W4995 Applied Machine Learning

Word Embeddings

04/10/17

Andreas Müller

Beyond Bags of Words

Limitations of bag of words:

- Semantics of words not captured
- Synonymous words not represented
- Very distributed representation of documents

Last Time

- Latent Semantic Analysis
- Non-negative Matrix Factorization
- Latent Dirichlet Allocation
- All embed documents into a continuous, corpus-specific space.
- Today: Embed words in a “general” space.

Idea

- Unsupervised extraction of semantics using large corpus (wikipedia etc)
- Input: one-hot representation of word (as in BoW).
- Use auxiliary task to learn continuous representation.

Skip-Gram models

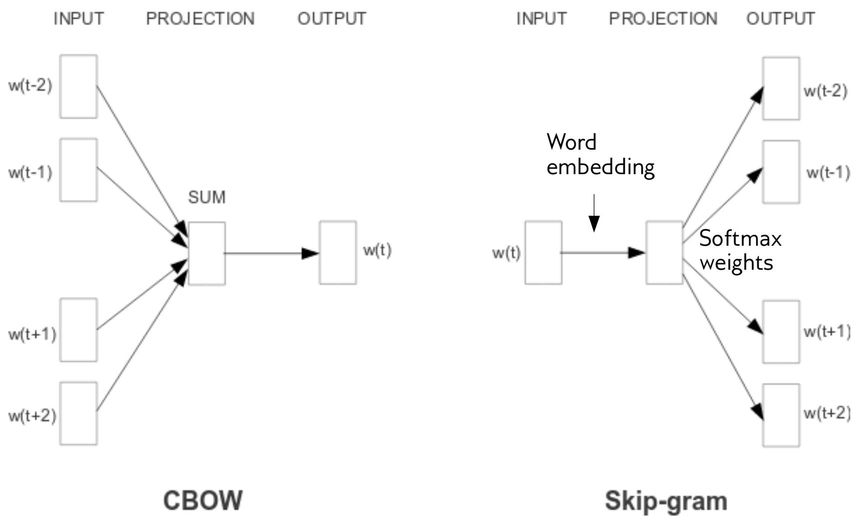
- Given a word, predict surrounding word
- Supervised task, each document yields many examples
- Not interested in performance for this task, just want to learn representations.

[“What is my purpose?”
“You pass the butter.”]

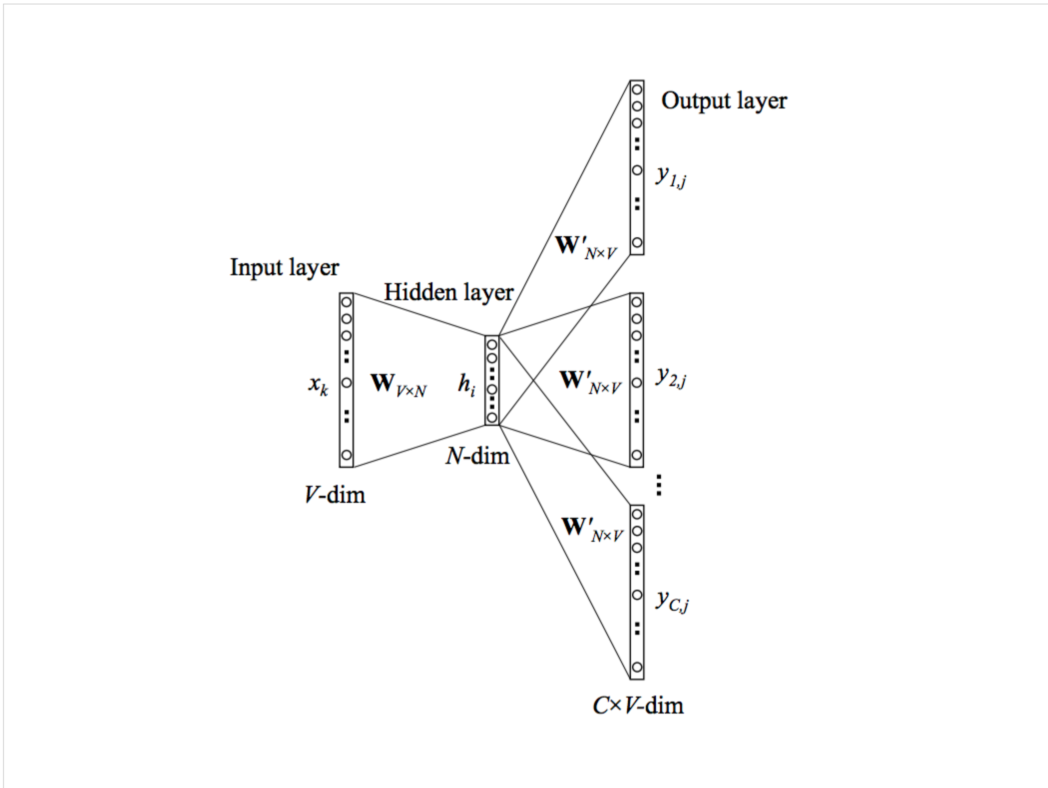
Using context windows of size 1 (in practice 5 or 10):

word	context
“is”	[“what”, “my”]
“my”	[“is”, “purpose”]
“pass”	[“you”, “the”]
“the”	[“pass”, “butter”]

CBOW vs Skip-gram



Efficient Estimation of Word Representations in Vector Space https://arxiv.org/pdf/1301.3781.pdf



Output layer
$y_{1,j}$
$\mathbf{W}'_{N\times V}$
Input layer
Hidden layer
x_k
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W}'_{N\times V}$
$y_{2,j}$
N-dim
V-dim
$\mathbf{W}'_{N\times V}$
$y_{C,j}$
$C\times V$-dim

Softmax Training

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

Output weights

Word embedding

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^\top v_{w_I}\right)}$$

Normalize over whole vocabulary!
[We want to do stochastic gradient descent / minibatch learning]
Monte-Carlo estimate: use some "noise words"

http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Implementations

- Gensim
- Word2vec
- Tensorflow
- For small applications don't train yourself

Gensim – topic models for humans

- Multiple LDA implementations
- Wrappers for Mallet and vopal wabbit
- Tools for analyzing topic models
- No supervised learning
- Uses list-of-tuples instead of sparse matrices to store documents.

Introduction to gensim

```
docs = ["What is my purpose", "You bring butter"]
texts = [[token for token in doc.lower().split()] for doc in docs]
print(texts)
```

```
[['what', 'is', 'my', 'purpose'], ['you', 'bring', 'butter']]
```

```
from gensim import corpora
dictionary = corpora.Dictionary(texts)
print(dictionary)
```

```
Dictionary(7 unique tokens: ['you', 'bring', 'butter', 'what', 'is']...)
```

```
new_doc = "what butter"
dictionary.doc2bow(new_doc.lower().split())
```

```
[(0, 1), (4, 1)]
```

```
corpus = [dictionary.doc2bow(text) for text in texts]
corpus
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1)], [(4, 1), (5, 1), (6, 1)]]
```

Use NLTK or spacy or a regexp for tokenization in real applications. This method here doesn’t even handle punctuation.

In gensim, a document is represented as list of tuples (index, frequency), corpus is a list of these.

Converting to/from sparse matrix

```
import gensim
corpus
```

```
[[(0, 1), (1, 1), (2, 1), (3, 1)], [(4, 1), (5, 1), (6, 1)]]
```

```
gensim.matutils.corpus2csc(corpus)
```

```
<7x2 sparse matrix of type '<class 'numpy.float64'>'
	with 7 stored elements in Compressed Sparse Column format>
```

Transposed!

```
X = CountVectorizer().fit_transform(docs)
X
```

```
<2x7 sparse matrix of type '<class 'numpy.int64'>'
	with 7 stored elements in Compressed Sparse Row format>
```

```
sparse_corpus = gensim.matutils.Sparse2Corpus(X.T)
print(sparse_corpus)
print(list(sparse_corpus))
```

```
<gensim.matutils.Sparse2Corpus object at 0x7faf015e3e48>
[[(4, 1), (3, 1), (2, 1), (5, 1)], [(1, 1), (0, 1), (6, 1)]]
```

most transformations is gensim are lazy:

They yield a generator (like Sparse2Corpus) which can be converted to a list.

Corpus Transformations

```
tfidf = gensim.models.TfidfModel(corpus)
tfidf[corpus[0]]
```

```
[(0, 0.5), (1, 0.5), (2, 0.5), (3, 0.5)]
```

```
print(tfidf[corpus])
print(list(tfidf[corpus]))
```

```
<gensim.interfaces.TransformedCorpus object at 0x7faf015e3ef0>
[[(0, 0.5), (1, 0.5), (2, 0.5), (3, 0.5)], [(4, 0.5773502691896258),
(5, 0.5773502691896258), (6, 0.5773502691896258)]]
```

Word2Vec in Gensim

```
from gensim import models
w = models.KeyedVectors.load_word2vec_format(
    '../GoogleNews-vectors-negative300.bin', binary=True)
```

```
w['queen'].shape
```

```
(300,)
```

```
w.syn0.shape
```

```
(3000000, 300)
```

Prepare document

```
vect_w2v = CountVectorizer(vocabulary=w.index2word)
vect_w2v.fit(text_train_sub)
docs = vect_w2v.inverse_transform(vect_w2v.transform(text_train_sub))
docs[0]
```

```
array(['in', 'for', 'that', 'is', 'the', 'at', 'not', 'as', 'it', 'by',
       'are', 'have', 'an', 'this', 'they', 'but', 'one', 'which', 'do',
       'than', 'over', 'just', 'some', 'like', 'only', 'did', 'because',
       'off', 'being', 'my', 'very', 'much', 'go', 'under', 'does', 'got',
       'top', 'come', 'really', 'lot', 'find', 'thing', 'once', 'offer',
       'feel', 'film', 'medical', 'terms', 'rather', 'certain', 'felt',
       'consider', 'watch', 'parts', 'heavy', 'towards', 'enjoy',
       'feeling', 'maybe', 'piece', 'fear', 'myself', 'stuff', 'handed',
       'brings', 'movies', 'hospitals', 'rare', 'lots', 'skin', 'eating',
       'intense', 'somewhat', 'liked', 'afraid', 'tribute', 'horror',
       'revenge', 'brave', 'whilst', 'sympathy', 'assaulted', 'satisfying',
       'hatred', 'viewer', 'awhile', 'pardon', 'delightful', 'disgust',
       'imitation', 'pudding', 'cringe', 'jaded', 'pun', 'pangs', 'doesn',
       'junctures', 'hellraiser', 'appologize'],
      dtype='<U98')
```

This is a silly way to tokenize the input and using only words that appear in the vocabulary used in the pre-trained model.

Represent doc by average

```
X_train = np.vstack([np.mean(w[doc], axis=0) for doc in docs])
```

```
X_train.shape
```

```
(18750, 300)
```

```
docs_val = vect_w2v.inverse_transform(vect_w2v.transform(text_val))
X_val = np.vstack([np.mean(w[doc], axis=0) for doc in docs_val])
```

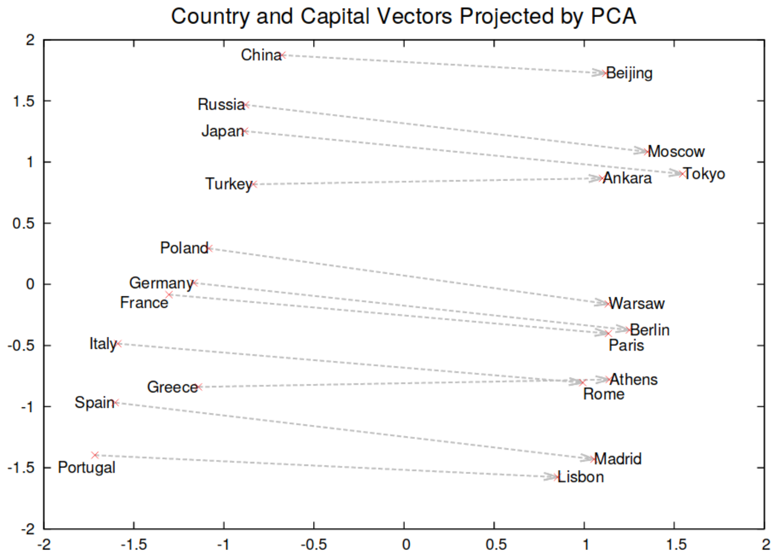
```
lr = LogisticRegression(C=100).fit(X_train, y_train_sub)
lr.score(X_train, y_train_sub)
```

```
0.8676266666666666
```

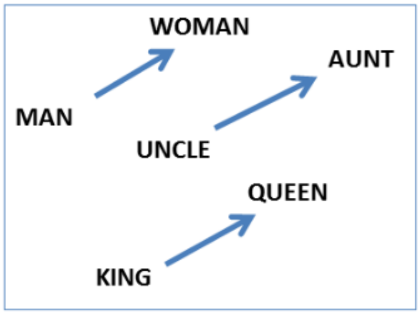
```
lr.score(X_val, y_val)
```

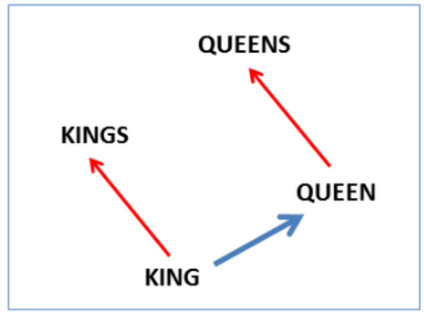
```
0.8571199999999999
```

Analogues and Relationships



http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf





Answer “King is to Kings as Queen is to ?”:
Find closest vector to vec(“Queen”) + (vect(“Kings”) - vec(“King”))

http://www.aclweb.org/anthology/N13-1090

Examples with Gensim

```
w.most_similar(positive=['woman', 'king'], negative=['man'], topn=3)
```

```
[('queen', 0.7118192911148071),
 ('monarch', 0.6189674139022827),
 ('princess', 0.5902431607246399)]
```

```
w.most_similar(positive=['woman', 'he'], negative=['man'], topn=3)
```

```
[('she', 0.8492251634597778),
 ('She', 0.6329933404922485),
 ('her', 0.6029669046401978)]
```

```
w.most_similar(positive=['Germany', 'pizza'], negative=['Italy'], topn=3)
```

```
[('bratwurst', 0.5436394810676575),
 ('Domino_pizza', 0.5133179426193237),
 ('donuts', 0.5121968984603882)]
```

Input	Result Produced
Chicago : Illinois : : Houston	Texas
Chicago : Illinois : : Philadelphia	Pennsylvania
Chicago : Illinois : : Phoenix	Arizona
Chicago : Illinois : : Dallas	Texas
Chicago : Illinois : : Jacksonville	Florida
Chicago : Illinois : : Indianapolis	Indiana
Chicago : Illinois : : Austin	Texas
Chicago : Illinois : : Detroit	Michigan
Chicago : Illinois : : Memphis	Tennessee
Chicago : Illinois : : Boston	Massachusetts

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Tolga Bolukbasi[1], Kai-Wei Chang[2], James Zou[2], Venkatesh Saligrama[1,2], Adam Kalai[2]
[1]Boston University, 8 Saint Mary's Street, Boston, MA
[2]Microsoft Research New England, 1 Memorial Drive, Cambridge, MA
tolgab@bu.edu, kw@kwchang.net, jamesyzou@gmail.com, srv@bu.edu, adam.kalai@microsoft.com

$$\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{king}} - \overrightarrow{\text{queen}}$$

$$\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{computer programmer}} - \overrightarrow{\text{homemaker}}.$$

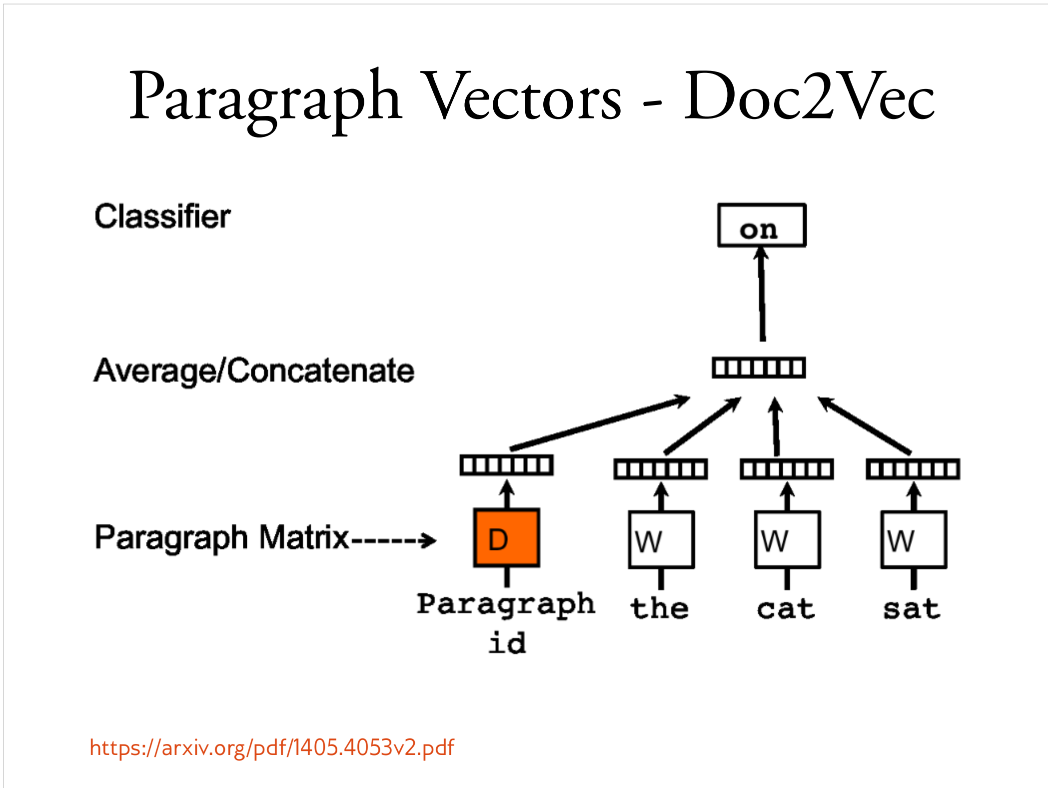
Going along he-she direction:

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



Add a vector for each paragraph / document, also randomly initialized.

To infer for new paragraph: keep weights fixed, do stochastic gradient descent on the representation D, sampling context examples from this paragraph.

Doc2Vec with gensim

```
def read_corpus(text, tokens_only=False):
    for i, line in enumerate(text):
        if tokens_only:
            yield gensim.utils.simple_preprocess(line)
        else:
            # For training data, add tags
            yield gensim.models.doc2vec.TaggedDocument(gensim.utils.simple_preprocess(line), [i])
```

```
train_corpus = list(read_corpus(text_train_sub))
test_corpus = list(read_corpus(text_val, tokens_only=True))
```

```
model = gensim.models.doc2vec.Doc2Vec(size=50, min_count=2, iter=55)
model.build_vocab(train_corpus)
```

```
model.train(train_corpus, total_examples=model.corpus_count)
```

https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/doc2vec-lee.ipynb

Encoding using doc2vec

```
vectors = [model.infer_vector(train_corpus[doc_id].words)
           for doc_id in range(len(train_corpus))]
```

```
X_train = np.vstack(vectors)
```

```
X_train.shape
```

```
(18750, 50)
```

```
test_vectors = [model.infer_vector(test_corpus[doc_id])
                for doc_id in range(len(test_corpus))]
```

```
X_test = np.vstack(test_vectors)
```

```python
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=100).fit(X_train, y_train_sub)
```

```python
lr.score(X_train, y_train_sub)
```

```
0.81738666666666671
```

```python
lr.score(X_val, y_val)
```

```
0.80320000000000003
```

Not working well here. Either not enough training data, sgd didn’t converge yet, or too low dimension.

GloVe: Global Vectors for Word Representation

- https://nlp.stanford.edu/projects/glove/
- Co-occurence not prediction

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 ,$$

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} .$$

Here X_ij is the coocurrance count matrix, counting how often word I and j appear in the same context – say a 5 word window.

We are learning a factorization of log(X_ij) into w_i and ~w_j, where w_i are the word-vectors we want to extract.

Very infrequent word-pairs are less important to get right, and are down-weighted using f(X_ij).

Word analogies

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	<u>67.5</u>	<u>54.3</u>	<u>60.3</u>
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	<u>64.8</u>	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	<u>80.8</u>	61.5	<u>70.3</u>
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	<u>67.4</u>	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	<u>77.4</u>	67.0	<u>71.7</u>
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**<u>81.9</u>**	**<u>69.3</u>**	**<u>75.0</u>**

https://nlp.stanford.edu/pubs/glove.pdf

Comparison of CBOW, SGD, skip-gram and Glove on the semantic (ie. state→capital) and syntactic (ie singular→ plural) word analogy tasks