Embeddings and Deep Learning

Exercises

17-21 July, ESSLLI 2017

0 Requirements

Exercise 0.1 *Installation*

To do the following exercises you will need certain python packages. This first exercise is about installing them. You will need `sklearn`, `nltk`, `numpy`, `gensim`. Please make sure you have installed them (by your distribution's package manager, pip, anaconda, . . .) and check your installation by trying to import them:

```
import sklearn
import nltk
import numpy
import gensim
```

1 Wordspace

Exercise 1.1 *First steps with Wordspace*

In `wordspace.py` you find some convenience functions to extract a word cooccurrence matrix from text. Run the following script and evaluate the embeddings by looking at the nearest neighbors of some words.

```
from wordspace import cooccurrence_matrix,\
    nearest_neighbor_loop

with open('brown.txt', 'r') as f:
    brown = f.read()

matrix, vocabulary = cooccurrence_matrix(brown)
nearest_neighbor_loop(matrix, vocabulary)
```

Exercise 1.2 *Model improvements (I)*

One simple way to improve a basic counting model is transforming the word counts by, e.g., applying the square root afterwards.
Modify the script from exercise 1.1 by using `numpy.sqrt` to do so.

Exercise 1.3 *Model improvements (II)*

Next let us examine the parameters of the function `cooccurrence_matrix`. You can modify the `window_size` and/or try a different `vectorizer` than the standard `CountVectorizer` to compute the cooccurrence scores. Try `sklearn.feature_extraction.text.TfidfVectorizer`!

```
cooccurrence_matrix(
    text, window_size=2, max_vocab_size=20000,
    same_word_zero=False, vectorizer=CountVectorizer
)
```

2 Singular Value Decomposition

Exercise 2.1 *Lower dimensionality*

With Singular Value Decomposition (SVD) you can reduce the dimensionality of your embeddings. Try `sklearn.decomposition.TruncatedSVD` and see how your embeddings change! Consider the following usage example:

```
C, V = cooccurrence_matrix(some_text)
svd = TruncatedSVD(
    n_components=100, algorithm="randomized",
    n_iter=5, random_state=42, tol=0.
)
new_C = svd.fit_transform(C)
```

`n_components`	desired embedding dimension
`algorithm`	SVD solver to use; either “arpack” or “randomized”
`n_iter`	number of iterations for randomized SVD solver (not used by ARPACK)
`random_state`	seed for pseudo-random number generator
`tol`	toleranze for ARPACK. Ignored by randomized SVD solver

3 Word2Vec

Exercise 3.1 *How to train your word2vec*

Use the following code snippets to train your own word2vec model on the brown corpus (or any other large text file you have). `semantic_tests.py` contains some tests for your embeddings. Feel free to add more!

```
from semantic_tests import semantic_tests
from gensim.models.word2vec import Word2Vec
import nltk.data
from nltk.tokenize import word_tokenize
import logging
logging.basicConfig(
    format='%(asctime)s: %(levelname)s: %(message)s',
    level=logging.INFO
)

sent = nltk.data.load(
    'tokenizers/punkt/english.pickle'
)
with open('brown.txt', 'r') as f:
    sentences = sent.tokenize(f.read())
sentences = map(lambda s: word_tokenize(s), sentences)

model = Word2Vec(
    sentences, size=100, window=5,
    min_count=5, hs=0, negative=5,
    cbow_mean=1, iter=5, workers=3
)

semantic_tests(model.wv)
```

Hint: The keyword arguments of `Word2Vec` should look familiar to you. You can use them as you would use the command line arguments of the `word2vec` script.

Exercise 3.2 *Load pretrained embeddings*

Instead of training your own word2vec model, you can also download pre-trained embeddings and load them into `gensim`. Are they doing better in your `semantic_tests`?

```python
from gensim.models import KeyedVectors
from semantic_tests import semantic_tests

model = KeyedVectors.load_word2vec_format(
    'path/to/GoogleNews-vectors-negative300.bin.gz',
    binary=True
)

semantic_tests(model)
```

Exercise 3.3 *Bonus exercise: Phrase embeddings*

`gensim` also includes a module for phrase detection (i.e. two or more words belonging together like *New York*). If you have time, you can try to train embeddings for these, too!

```python
from gensim.models.phrases import Phrases, Phraser

bigram = Phraser(Phrases(sentences))
model = Word2Vec(list(bigram_transformer[sentences]))

print(model.wv.most_similar(['New_York']))
```

4 FastText

Exercise 4.1 *How to train your fasttext*

For this exercise you need to download and build FastText[1] as `gensim` only provides a wrapper around the actual fasttext library. Then you can use it like this:

```python
from gensim.models.wrappers import FastText
from semantic_tests import semantic_tests

model = FastText.train(
    "path/to/fasttext",
    corpus_file='brown.txt'
)
semantic_tests(model)
```

[1] https://github.com/facebookresearch/fastText

5 Deep Learning

Exercise 5.1 *New requirements*

You will need to install `torch` and `torchtext` for the last exercise.

`http://cis.lmu.de/esslli2017/convolutional.tar.gz` contains a shell script `install_requirements.sh` that can do this for you (Anaconda and using the script with `./install_requirements.sh conda` is recommended).

In case you need to troubleshoot your installation, please make sure you tried to install it before the last course session.

Exercise 5.2 *Sentiment Classification*

Download the pytorch implementation of a convolutional neural network for text classification from
`http://cis.lmu.de/esslli2017/convolutional.tar.gz`.

Try different hyperparameters. You can also modify

(I) if the word embeddings should be randomly initialized or loaded from word2vec (cf. exercise 3.2) and

(II) if the embeddings should be kept static or be fine-tuned during training.