Temporal Word Embeddings with a Compass Documentation

Release 1.1.6

Federico Bianchi

Apr 29, 2020

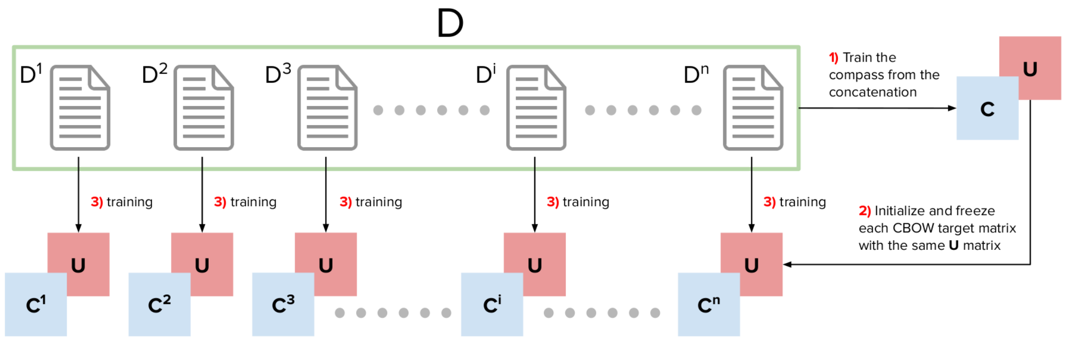
Contents:

CHAPTER 1

Compass-aligned Distributional Embeddings

This package contains Python code to generate compass aligned distributional embeddings. Comparing word vectors in different corpora requires alignment. We propose a method to aligned distributional representation based on word2vec. This method is efficient and it is based on a simple heuristic: we train a general word embedding, **the compass** and we use this embedding to freeze one of the layers of the CBOW architecture.

See the AAAI and Arxiv pre-print papers for more details.


D
D1
D2
D3
Di
Dn
1) Train the compass from the concatenation
U
C
3) training
3) training
3) training
3) training
3) training
2) Initialize and freeze each CBOW target matrix with the same U matrix
U
C1
U
C2
U
C3
U
Ci
U
Cn

1.1 Reference

This work is based on the following papers: AAAI and Arxiv-preprint

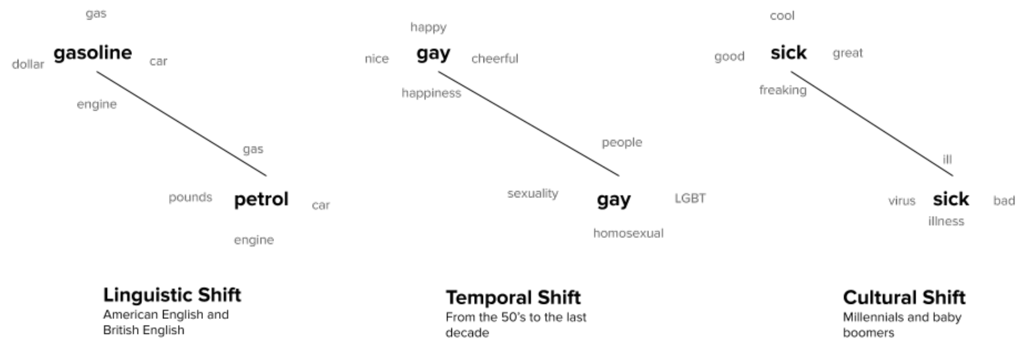
- Bianchi, F., Di Carlo, V., Nicoli, P., & Palmonari, M. (2020). **Compass-aligned Distributional Embeddings for Studying Semantic Differences across Corpora**. Arxiv. https://arxiv.org/abs/2004.06519
- Di Carlo, V., Bianchi, F., & Palmonari, M. (2019). **Training Temporal Word Embeddings with a Compass**. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), 6326-6334. https://doi.org/10.1609/aaai.v33i01.33016326

1.2 Abstract

Word2vec is one of the most used algorithms to generate word embeddings because of a good mix of efficiency, quality of the generated representations and cognitive grounding. However, word meaning is not static and depends on the context in which words are used. Differences in word meaning that depends on time, location, topic, and other factors, can be studied by analyzing embeddings generated from different corpora in collections that are representative of these factors. For example, language evolution can be studied using a collection of news articles published in different time periods. In this paper, we present a general framework to support cross-corpora language studies with word embeddings, where embeddings generated from different corpora can be compared to find correspondences and differences in meaning across the corpora. CADE is the core component of our framework and solves the key problem of aligning the embeddings generated from different corpora. In particular, we focus on providing solid evidence about the effectiveness, generality, and robustness of CADE. To this end, we conduct quantitative and qualitative experiments in different domains, from temporal word embeddings to language localization and topical analysis. The results of our experiments suggest that CADE achieves state-of-the-art or superior performance on tasks where several competing approaches are available, yet providing a general method that can be used in a variety of domains. Finally, our experiments shed light on the conditions under which the alignment is reliable, which substantially depends on the degree of cross-corpora vocabulary overlap.

1.3 What's this About?

Different words assume different meaning in different contexts. Think for example of how people once used the word *amazon* to mainly refer to the forest. Or think about the differences between American and British English. This is what we usually call meaning shift. See some examples of meaning shifts:



Why not using standard word embeddings? Well, long story short, different embeddings generated from different corpora are not comparable: they need to be aligned!

With CADE we provide a method to align different corpora (in the same language) and to compare them. Alignment allow us to compare different word embeddings in different corpora using cosine similarity!

Here are some example of mappings between text about Pokemons (from the Reddit board) and text about Scientific stuff (again, Reddit) that you can learn with CADE.

For example, you can take the vector of the word Arceus, from the Pokemon corpus and find that it is very similar to the word *god* in the Science corpus. You wonder why? Arceus is the good of Pokemons! See some examples of mapping like this in the table, where we show the top-5 nearest neighbors of the mapped space!

Mapping	Word	Top-5
SCI → POK	move	'**walk**', 'go', 'hop', 'sit', 'rotate'
POK	move	'tm', 'moves', 'moven', 'superpower', 'attacks'
POK → SCI	arceus	'**gods**', 'sakes', 'worship', 'creator', 'god'
POK	arceus	'mew', 'deoxys', 'giratina', 'celebi', 'regigigas'
SCI → POK	ash	'lava', 'moisture', 'air', 'ocean', 'clouds'
POK	ash	'brock', 'misty', 'serena', 'giovanni', 'gary'

1.4 Installing

We use a custom/edited implementation of gensim, this **WILL** clash with your gensim installation, consider installing this inside a virtual environment

```
pip install -U cade
```

REMEMBER TO USE A VIRTUAL ENVIRONMENT

```
pip install git+https://github.com/valedica/gensim.git
```

1.5 Guide

- **Remember**: when you call the training method of `CADE` the class creates a "model/" folder where it is going to save the trained objects. The compass will be trained as first element and it will be saved in that folder. If you want to overwrite it remember to set the parameter `overwrite=True`, **otherwise** it will reload the already trained compass.
- **What do you need**: Different corpora you want to compare (i.e., text from 1991, text from 1992 / text from the New York Times, text from The Guardian . . . etc. . .) and the concatenation of those text slices (the compass).
- **The compass** should be the concatenation of the slice you want to align. In the next code section you will see that we are going to use arxiv papers text from two different years. The "compass.txt" file contains the concatenation of both slices.

1.6 How To Use

- Training

Suppose you have corpora you want to compare text "arxiv_14.txt" and "arxiv_9.txt". First of all, create the concatenation of these two and create a "compass.txt" file. Now you can train the compass.

```
from cade.cade import CADE
from gensim.models.word2vec import Word2Vec
aligner = CADE(size=30)

```

(continues on next page)

(continued from previous page)

```
# train the compass: the text should be the concatenation of the text from the slices
aligner.train_compass("examples/training/compass.txt", overwrite=False) # keep an eye␣
↪on the overwrite behaviour
```

You can see that the class covers the same parameters the Gensim word2vec library has. After this first training you can train the slices:

```
# now you can train slices and they will be already aligned
# these are gensim word2vec objects
slice_one = aligner.train_slice("examples/training/arxiv_14.txt", save=True)
slice_two = aligner.train_slice("examples/training/arxiv_9.txt", save=True)
```

These two slices are now aligned and can be compared!

- Load Data

You can load data has you do with gensim.

```
model1 = Word2Vec.load("model/arxiv_14.model")
model2 = Word2Vec.load("model/arxiv_9.model")
```

and you can start comparing it with standard methods

```
from scipy.spatial.distance import cosine
print(1 - cosine(model1["like"], model["sign"]))
```

1.7 People

- Federico Bianchi - Bocconi University - (f.bianchi@unibocconi.it)
- Valerio Di Carlo - BUP Solutions
- Paolo Nicoli - University of Milano-Bicocca
- Matteo Palmonari - University of Milano-Bicocca - (matteo.palmonari@unimib.it)

1.8 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Compass-aligned Distributional Embeddings, run this command in your terminal:

```
$ pip install cade
```

This is the preferred method to install Compass-aligned Distributional Embeddings, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Compass-aligned Distributional Embeddings can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/vinid/cade
```

Or download the tarball:

```
$ curl -OJL https://github.com/vinid/cade/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Compass-aligned Distributional Embeddings in a project:

```
import cade
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/vinid/cade/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

4.1.4 Write Documentation

Compass-aligned Distributional Embeddings could always use more documentation, whether as part of the official Compass-aligned Distributional Embeddings docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/vinid/cade/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *cade* for local development.

1. Fork the *cade* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/cade.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv cade
   $ cd cade/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 cade tests
   $ python setup.py test or pytest
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.5, 3.6 and 3.7, and for PyPy. Check https://travis-ci.org/vinid/cade/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_cade
```

4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

Credits

5.1 Development Lead

- Federico Bianchi <f.bianchi@unibocconi.it.it>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2019-09-11)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search