Software for creating and analyzing semantic representations

Finn Årup Nielsen and Lars Kai Hansen

Cognitive Systems, DTU Compute, Technical University of Denmark

Abstract. In this chapter we describe some of the software packages for learning distributed semantic representation in the form of word and graph embeddings. We also describe several Python natural language processing frameworks which can prepare a corpus for the embedding software. We furthermore point to the Wikidata software as a tool for collaborative construction of explicit semantic representation and to tools to embed such data.

- Many natural language processing packages exist, including several written in the Python programming language.
- The popular scikit-learn, Gensim and Keras Python packages provide means for quickly building and training machine learning models on large corpora.
- Several word embedding software packages exist for efficiently building distributed representations of words.
- Many of the software packages have associated pre-trained models in multiple languages trained on very large corpora and the models are distributed freely for others to use.
- Software for collaboratively building large semantic networks and knowledge graphs exist and these graphs can be converted to a distributed representation.

1 Introduction

Since the first description and publication of modern efficient word embedding tools around 2013, they have seen wide-spread use in many areas of text mining that need semantic information. The free software license, the easy setup and its often very good performance on semantic tasks have no doubt been important for their adoption.

Natural language processing (NLP) software may be necessary for corpus preparation as dedicated semantic representation software does not handle different forms of textual input. For instance, Wikipedia derived text should usually be stripped of the special wiki markup present in MediaWiki-based wikis. Word embedding software may expect the input to be tokenized and white-space separated, so word tokenization and perhaps a sentence segmentation is necessary.

For morphological rich language, handling of the word variations may be important to avoid too sparse data when estimating a semantic model. Stemmers

and lemmatizers may be used to convert word variations to a basic form. Languages with many and rare compound nouns (e.g., German) can face an issue with many out-of-vocabulary words, so decompounding can be advantageous.

Datasets and pre-trained models are important parts of natural language processing, including semantic modeling. Python packages, NLTK and polyglot, have built-in facilities for easy download of their associated data and pre-trained models making the setup of a working system less of a hassle. These packages also saves the data in predefined directories, making subsequent setup simpler. The researchers using word embedding software have also distributed a number of pre-trained model for others to use.

Many text mining packages exist. Below we will focus on a few Python natural language processing packages: NLTK, spaCy, Pattern and polyglot for general natural language processing packages. Then we will describe word embedding software and some other software to create semantic representations.

2 Natural Language Processing Toolkit, NLTK

NLTK, short for Natural Language Processing Toolkit, is a Python package and perhaps one of the most popular Python NLP packages. It is documented in a detailed book [4] available in print as well as on the Internet at http://www.nltk.org/book/ under a Creative Commons license. NLTK packages a large number of models for a range of NLP tasks and furthermore includes methods for downloading and loading several corpora and other language resources.

Pre-trained sentence tokenizers exist for several languages, and different word tokenizers are also implemented. For instance, the `word_tokenize` function uses the default word tokenizer, which in NLTK version 3.2.5 is a modified version of the regular expression-based `TreebankWordTokenizer`. An example application of this default tokenizer for the sentence "I don't think you'll see anything here: http://example.org :)" is:

```
from nltk.tokenize import word_tokenize

text = ("I␣don't␣think␣you'll␣see␣anything␣here:␣"
        "http://example.org␣:)")
token_list = word_tokenize(text)
```

This tokenization yields a list with the individual words identified, but with the URL and the emoticon not well handled: `['I', 'do', "n't", 'think', 'you', "'ll", 'see', 'anything', 'here', ':', 'http', ':', '//example.org', ':', ')']`. Both the URL and the emoticon are split into separate parts. The NLTK's `TweetTokenizer` handles such tokens better. An example application of this method reads

```
from nltk.tokenize import TweetTokenizer

tokenizer = TweetTokenizer()
token_list = tokenizer.tokenize(text)
```

The result is `['I', "don't", 'think', "you'll", 'see', 'anything', 'here', ':', 'http://example.org', ':)']`, where the URL and the emoticon are detected as tokens.

NLTK has various methods to deal with morphological variations of words. They are available from the `nltk.stem` submodule. For English and some other languages, NLTK implements several stemmers. Variations of the Snowball stemmer by Martin Porter work for a range of languages, here an example in French

```
from nltk.stem.snowball import FrenchStemmer
from nltk.tokenize import word_tokenize

text = ("La cuisine française fait référence à divers "
        "styles gastronomiques dérivés de la "
        "tradition française.")
stemmer = FrenchStemmer()
[stemmer.stem(word) for word in word_tokenize(text)]
```

This yields `['la', 'cuisin', 'français', 'fait', 'référent', 'à', 'diver', 'styl', 'gastronom', 'dériv', 'de', 'la', 'tradit', 'français', '.']`. For English, a WordNet-based lemmatizer is available.

Part-of-speech taggers are implemented in the `nltk.tag` submodule. The default tagger available with the function `pos_tag` in NLTK 3.2.5 uses the Greedy Averaged Perceptron tagger by Matthew Honnibal [13]. Trained models for English and Russian are included.

Trained models are usually saved as Python's pickle format. This format has the unfortunate problem of posing a security issue, as the pickle files can contain executable code, that gets executed if the file is loaded. It means that pickle data should only be downloaded from trusted sources. Trusted models distributed by the NLTK developers can fairly easily be downloaded by functions provided by the toolkit. By default they will be saved under the `~/nltk_data/` directory. For instance, the Danish tokenizer is saved `~/nltk_data/tokeniers/punkt/danish.pickle`. The functions of NLTK that require these models for setup will automatically read such files.

NLTK has also methods for handling semantic representations in the form of word nets. Access to the Princeton WordNet (PWN) [21] is readily available in a submodule imported, e.g., by:

```
from nltk.corpus import wordnet as wn
```

With this at hand, synsets and lemmas can be found based on the word representation. Semantic similarities can be analyzed with one of several similarity methods that transverse the word net taxonomy. Below are the similarities between pairs of 4 nouns (*dog*, *cat*, *chair* and *table*) computed with the `path_similarity` method which computes the similarity based on shortest path in the hypernym/hyponym graph:

```
import numpy as np
words = 'dog cat chair table'.split()
similarities = np.zeros((len(words), len(words)))
```

	dog	cat	chair	table
dog	1.00	0.20	0.08	0.07
cat	0.20	1.00	0.06	0.05
chair	0.08	0.06	1.00	0.07
table	0.07	0.05	0.07	1.00

Table 1. NLTK WordNet similarities

```
for n, word1 in enumerate(words):
    for m, word2 in enumerate(words):
        synset1 = wn.synsets(word1, pos='n')[0]
        synset2 = wn.synsets(word2, pos='n')[0]
        similarities[n, m] = synset1.path_similarity(synset2)
```

The result is displayed in Table 1, where the words *cat* and *dog* are found to have the highest similarity, while *cat* and *table* the lowest. For *chair*, the word *dog* has a higher similarity than *table* which in usual contexts would not be right. Each word may have multiple synsets, e.g., *dog* has 7 synsets. In the code above, the most common synset is selected.

3 spaCy

spaCy (https://spacy.io/) is a relatively new Python package with an initial release in 2015. It offers a range of NLP methods, such as tokenization, POS-tagging, sentence segmentation, dependency parsing, named entity recognition and semantic similarity computation. A 2015 paper showed that its dependency parser was the fastest among 13 systems evaluated [7].

spaCy has a number of pre-trained models for several languages. A command-line method downloads and installs the model packages. The packages with models for a specific language ship in various sizes, and for working with distributional semantics, the largest package should be downloaded for best result. For instance, one of the large English packages is downloaded with the command:[1]

```
python -m spacy download en_core_web_lg
```

This package provides models for the tagger, parser, named-entity recognizer and distributional semantic vectors trained on OntoNotes Release 5 and the Common Crawl dataset. An even larger package (`en_vectors_web_lg`) contains 300-dimensional distributional semantic vectors for 1.1 million tokens trained on the Common Crawl dataset with GloVe [26]. Once installed, the models can be used from within Python:

```
import spacy
nlp = spacy.load('en_core_web_lg')
doc = nlp(u'A␣lion␣is␣a␣large␣cat.␣It␣lives␣in␣Africa')
```

[1] A list of the various models is available at https://spacy.io/models/.

The resulting object, here named `doc`, has a number of attributes which analyze the text with lazy evaluation. For instance, `doc.sents` returns a generator that yields the sentences of the text as so-called `Span` objects, while `doc.ents` returns named entities, here 'Africa', and `doc.noun_chunks` returns the `Span` objects representing 'A lion', 'a large cat', 'It' and 'Africa'. The object itself iterates over tokens. An embedding of the text is available as `doc.vector`. This is created as the average of the word embedding. It can also be computed by iterating over the tokens and computing the average over the word embedding of each token. The example below compares the overall embedding with the computed average of the individual word embeddings of each token:

```
token_vectors = [token.vector for token in doc]
average_vector = sum(token_vectors) / len(token_vectors)
sum(doc.vector - average_vector) == 0
```

The last line yields true. The dimension of the subspace is 300 in this case, while `token_vectors` is a list of 11 300-dimensional vectors (the punctuation is also embedded).

As of 2018, spaCy distributes pre-trained models for full support of English as well as a few European language, see https://spacy.io/models/. French and Spanish models also support embeddings, while German, Portuguese, Italian and Dutch have less support. A multilingual model can be used for name entity detection.

4 Pattern

Open source *Pattern* Python package provides methods for processing text data from the web [27]. Exposed in the `pattern.web` submodule, it has download methods for a range of web services including popular web search engines,[2] Twitter, individual Wikipedia articles and news feeds. It also features a web crawler and an object to retrieve e-mail messages via IMAP.

Pattern has natural language processing methods for a few European language, English, German, French, Italian and Dutch, with varying degree of implementation. For English there are, e.g., a part-of-speech tagger, lemmatizer, singularization/pluralization, conjugation and sentiment analysis. It also contains a small set of word lists (academic, profanity, time and a basic list with 1000 words) as well as an interface to WordNet.

The `pattern.vector` submodule exposes various methods based around the vector space model representation, word count, tf-idf, latent semantic analysis. This module also features K-means and hierarchical document clustering algorithms and supervised classification algorithms.

5 Polyglot

Polyglot is a multilingual NLP Python package [2]. It can be applied both as a Python module and as a command-line script. As shown in the documenta-

[2] Some of the web search engines are paid services, requiring a license key.

tion at https://polyglot.readthedocs.io, it has language detection, tokenization, part-of-speech tagging, word embedding, named entity extraction, morphological analysis, transliteration and sentiment analysis for many languages. Some of the methods require the download of files from the web. The command-line polyglot program will automatically download the files and in a manner similar to NLTK store them locally in directory such as `~/polyglot_data`. For instance, the command

```
polyglot download embeddings2.de
```

will download the German word embeddings file. Word embeddings are generated from the different language versions of Wikipedia and trained on a Theano-implementation of a curriculum learning-inspired method [2]. Apart from working with its own word embedding format, the polyglot word embeddings are also able to load pre-trained Gensim, Word2vec and GloVe models.[3] A series of commands that find the four nearest words to the German word 'Buch' (book) may read

```
from os.path import expanduser
from polyglot.mapping import Embedding

directory = expanduser('~/polyglot_data/embeddings2/de/')
filename = directory + 'embeddings_pkl.tar.bz2'
embedding = Embedding.load(filename)
words = embedding.nearest_neighbors('Buch', 4)
```

These commands yield `['Werk', 'Schreiben', 'Foto', 'Archiv']`. The embedding vector is available as `embedding['Buch']`. The German word embedding has a vocabulary of 100004 tokens and the dimension of the embedding space is 64. Note that the polyglot embeddings distinguish between upper and lower case letter. Out-of-vocabulary case-variations of words can be handled with polyglot's 'case expansion'.

6 MediaWiki processing software

Widely used as a free, large and multilingual text corpus [18], Wikipedia poses a special challenge to parse. The entire text is distributed as large compressed XML dump files where the text is embedded with the special MediaWiki markup for rendering of, e.g., tables, citations, infoboxes and images. There exist a few tools to convert the raw MediaWiki markup to a form suitable for standard NLP tools. The Python package `mwparserfromhell` parse the MediaWiki markup and can filter the individual parsed components to text in various ways. The Python code below downloads the 'Denmark' article from the English Wikipedia and strips any formatting.

```
import requests, mwparserfromhell
```

[3] See, e.g., https://polyglot.readthedocs.io/en/latest/Embeddings.html.

```
url = 'https://en.wikipedia.org/w/index.php'
response = requests.get(url,
    params={'title': 'Denmark', 'action': 'raw'})
wikicode = mwparserfromhell.parse(response.content)
text = wikicode.strip_code()
```

The Gensim package (see below) has its own dedicated wiki extractor: the `make_wiki` script.

7 Scikit-learn

Scikit-learn, also called sklearn, is a popular Python package for machine learning [25]. Its popularity may stem from implementation of a wide variety of machine learning algorithms and a consistent application programming interface (API). Though not specifically targeted at text processing, Scikit-learn does have a number of ways to handle text and convert it to a numerical matrix representation. The `sklearn.feature_extraction.text` submodule defines the `CountVectorizer` which tokenizes a list of texts into words or characters and counts the occurrences of the tokens, returning a bag-of-words or bag-of-n-grams representation. `TfidfTransformer` does the popular *tf-idf* normalization of a count matrix, while `TfidfVectorizer` combines the `CountVectorizer` and `TfidfTransformer` into one model. The `HashingVectorizer` applies hashing to the words and has a default size of 1048576 features. This particular hashing will make hash collisions, e.g., between "wearisome" and "drummers" and between "funk", "wag" and "deserters".

Several of scikit-learn's unsupervized learning models can be used to project high-dimensional representations to a two- or three-dimensional representation useful for a visualization. Relevant models are the ones in the `sklearn.decomposition` submodule, such as the principal component analysis (`PCA`) as well as the models in the `sklearn.manifold` submodule where we find the t-distributed Stochastic Neighbor Embedding (`TSNE`).

Yet other models in scikit-learn may be used for topic modeling. Apart from principal component analysis, scikit-learn implements non-negative matrix factorization with `sklearn.decomposition.NMF`, while latent Dirichlet allocation is implemented with `sklearn.decomposition.LatentDirichletAllocation`.

Andrej Karpathy's Arxiv Sanity web service with the code available from https://github.com/karpathy/arxiv-sanity-preserver provides an interesting working example of the use of scikit-learn's `TfidfVectorizer`. Karpathy downloads bibliographic information and PDFs from the arXiv preprint server at https://arxiv.org/, extracts the text from the PDF with the `pdftotext` command-line script, builds a word-bigram tfidf-weighted model with scikit-learn and computes document similarities based on inner products. The web service at http://www.arxiv-sanity.com can then use such similarities for ranking similar scientific papers given a specific paper.

Model	Corpus	Tokens	Vocabulary	Dim.	Comment
Word2vec[4]	Wikipedia		10K–50K	300	29 languages
GloVe[5]	Common Crawl	840G	2.2M	300	Cased, English
GloVe	Common Crawl	42G	1.9M	300	Uncased, English
GloVe	Wikipedia+Gigaword	6G	400K	300	Uncased, English. Models with dimension 50, 100 and 200 are also available
GloVe	Twitter	27G	1.2M	200	25, 50 and 100 dimensional models are also available.
fastText[6]	Wikipedia		up to 2.5M	300	294 different language versions are available
fastText[7]	Common Crawl + Wikipedia			300	157 different language versions

Table 2. Selection of important pre-trained word embedding models.

8 Word embedding

Word embeddings represent words in a continuous dense low-dimensional space. Although a number of systems exists for training, Word2vec, GloVe and fastText are probably the most popular. Below we describe these software packages

8.1 Word2vec

The original word2vec [19] word embedding software is available from https://code.google.com/archive/p/word2vec. The Apache license software is written in C and compiles to a multi-threaded command-line program. A small demonstration program uses the automatically downloaded 100 MB Wikipedia-derived text8 corpus to train a model in a few minutes time on a modern computer. Pre-trained word2vec models based on multiple languages of Wikipedia and Gensim, has been made available by Kyubyong Park at https://github.com/Kyubyong/wordvectors, see also Table 2.

The `word2vec` program has several parameters for the estimation of the model. Apart from parameters for reading the corpus and writing the trained model, some of the parameters are: *size*, which determines the dimension of the low-dimensional space. The default is 100 and researchers tend to set it to 300 for large corpora. *window* controls the size of the context. *cbow* switches between

[4] https://github.com/Kyubyong/wordvectors
[5] https://nlp.stanford.edu/projects/glove/
[6] https://fasttext.cc/docs/en/pretrained-vectors.html
[7] https://fasttext.cc/docs/en/crawl-vectors.html

the skip-gram and the continuous bag of words model. *min-count* discards words occuring less often. Setting this parameter to a high value will results in a smaller vocabulary.

8.2 GloVe

Another widely used system is GloVe [26] with the reference implementation available from https://nlp.stanford.edu/projects/glove/. Like word2vec, GloVe is distributed under the Apache license and written in C with a demonstration program using the text8 corpus. Several large pre-trained word embeddings are available from the homepage. In [26], the researchers reported performance for the so-called 6B (trained on Wikipedia and Gigaword corpora with 6 gigatokens) and 42B (trained on Common Crawl with 42 gigatokens). On word similarity and word analogy the model trained on the largest corpus (42B) proved the best, also in comparison with the word2vec variations and other examined approaches.

8.3 FastText

FastText is an open embedding library and program from Facebook AI Research. It is available from GitHub at https://github.com/facebookresearch/fastText under a BSD license and documented at https://fasttext.cc/ and in several scientific papers [5, 16, 20, 11]. The program is implemented in C++ with interface in Python. FastText requires a whitespace-separated tokenized corpus as input. Word phrases should be preprocessed. The ability to handle both words and character n-grams sets fastText apart.

The fastText research group distributes open pre-trained models. The so-called "Wiki word vectors" available from https://fasttext.cc/docs/en/pretrained-vectors.html are trained on 294 different language versions of Wikipedia using an embedding dimension of 300 and the skip-gram model. Another line of pre-trained models were initially only available in English but trained on very large datasets [20]. These models have produced strong results on benchmark datasets, including word and phrase analogy datasets, rare word similarity and a question answering system, so may likely be the current state-of-the-art. In 2018, the researchers used language detection on the very large dataset which enabled them to train separate models on 157 different languages. This improved the performance of fastText on a word analogy benchmark dataset, — for some language the improvement was considerable [11].

Apart from the unsupervised learning of a word embedding, fastText can also work in a supervised setting with a labeled corpus. For supervised training each line in the input should be prefixed with a string indicating the category.

8.4 Other word embedding software

The Multivec package implements bilingual word embeddings that should be trained on a parallel corpus [6]. It is available from

https://github.com/eske/multivec, and implements training as well as functions to compute similarity and find the semantically closest words across languages. The documentation shows a training with a 182,761 sentence large French–English parallel corpus and with this pre-trained model, an associated Python package can compute similarity values, e.g., between the French and English word for *dog*:

```
from multivec import BilingualModel

model = BilingualModel('news-commentary.fr-en.bin')
model.similarity('chien', 'dog')  # 0.7847443222999573
model.similarity('dog', 'chien')  # 0.0
model.similarity('chien', 'chien')  # 0.0
model.similarity('dog', 'dog')  # 0.0
```

Here the resulting similarities are shown after the method call. Only when the source embedding is French and the target English, does the model report a non-zero similarity. In the other cases, the words are out-of-vocabulary in one or two of the languages and the returned similarity value from the method is zero.

9 Other embedding software

Other forms of embedding software go beyond embedding of words. Of particular interest is graph embedding software that takes a graph (rather than a corpus) and embeds the nodes or both nodes and typed links between the nodes in the continuous embedding space. The tools may generate a 'sentence' of nodes (and possible links) by a random walk on the graph and submit the 'sentence' to conventional (word) embedding software. The node2vec package[8] embeds nodes via such random graph walks. The simple reference implementation uses the Python NetworkX package and Gemsim's implementation of Word2vec. In the accompanying paper [12], the node2vec embedding was trained on a Wikipedia-derived word co-occurrence corpus. RDF2vec[9] is a similar tool and also uses Gensim's Word2vec implementation, but works for relational graphs. Wembedder uses a simple RDF2vec-inspired approach to embed a relational graph in the form of the Wikidata knowledge graph. Wembedder embeds both Wikidata items (the nodes in the knowledge graph) as well as the Wikidata properties (the links) and makes similarity computations based on the trained model available as a web service with an associated API [24].

10 Gensim

Gensim is an open source Python package for topic modeling and related text mining methods going back to at least 2010 [29]. Distributed from

[8] https://snap.stanford.edu/node2vec/
[9] http://data.dws.informatik.uni-mannheim.de/rdf2vec/

https://radimrehurek.com/gensim/, the package has various methods to handle large corpora by streamed reading and to convert a corpus to a vector space representation.

Gensim implements several popular topic modeling methods: *tf-idf* (`TfidfModel`), latent semantic analysis (`LsiModel`), random projections (`RpModel`), latent Dirichlet allocation (`LdaModel`) and hierarchical Dirichlet process (`HdpModel`). It provides access to computed vectors so similarity computations can be made between the documents or between the documents and a query.

Gensim reimplements the word2vec models including training of the model from a text stream, returning the vector representation of a word and computing similarity between words. Several other word embedding algorithms are also implemented: doc2vec, fastText and Poincaré embedding.

Furthermore, Gensim wraps a few programs so they are accessible from within the Python session, although the programs need to be installed separately to work. They include WordRank [14] and VarEmbed [3] as well as Dynamic Topic Models and LDA models implemented in Mallet and Vowpal Wabbit. FastText was also wrapped but with the direct implementation in Gensim 3.2, the wrapper is now deprecated.

Gensim has a simple class to read the POS-tagged Brown Corpus from an NLTK installation, which can be used as an example. Provided that this small dataset is downloaded via NLTK, training a Gensim word2vec model may be done as follows

```
from os.path import expanduser, join
from gensim.models.word2vec import BrownCorpus, Word2Vec

dirname = expanduser(join('~', 'nltk_data', 'corpora',
    'brown'))
model = Word2Vec(BrownCorpus(dirname))
```

Here the POS-tags are included as part of the word. The training with this small corpus takes no more than a few seconds. A similarity search on the noun "house" (postfix with the POS-tag "nn" for nouns) with `model.similar_by_word('house/nn')` may yield the nouns "back", "room" and "door" as the most similar words.

11 Deep learning

Several free software packages for deep learning are available: Google's TensorFlow [1], Microsoft's CNTK,[10] Theano, PyTorch, Keras, MXNet, Caffe [15] and Caffe2. Model architectures and trained models may differ between the frameworks, but there are efforts towards a standardized format for interchange. One such effort is Open Neural Network Exchange (ONNX) described at https://github.com/onnx.

[10] https://docs.microsoft.com/en-us/cognitive-toolkit/

AllenNLP is a open source framework for natural language understanding using deep learning from the Allen Institute for Artificial Intelligence [10]. Available from http://allennlp.org/, it is built on top of PyTorch and spaCy and runs with Python 3.

11.1 Keras

Keras[11] is a high-level deep learning library. It runs on top of either TensorFlow, CNTK, or Theano. It is among the most popular deep learning frameworks.

Keras enables the deep learning programmer to easily build deep neural networks by chaining layers of simple classes representing, e.g., layers of weights (such as `Dense` or `Conv1D`) or activation functions (such as ReLU or softmax). It has a range of layers for recurrent neural networks, such as the popular long short-term memory (LSTM) unit. Of special interest for text mining are the text preprocessing functions and an embedding layer.

Under the `keras.application` submodule, Keras has a number of pre-trained deep learning models. Currently, they are all trained for image classification with the ImageNet dataset, so not of direct relevance in a text mining context, except in cases with combined image and text analysis, such as image captioning. It is possible to load pre-trained embedding models, such as GloVe models, as a Keras embedding layer with the `keras.layers.Embedding` class. The set up of pre-trained embedding models requires a verbose setup,[12] but the third-party *KerasGlove* package makes the set up simpler.

In the `keras.datasets` submodule, a couple of text corpora are readily available — mostly for benchmarking and not particular relevant for establishing general semantic models: The *Large Movie Review Data* ([17], by Keras referred to as *IMDB Movie reviews*) with 25,000 movie reviews in the training set labeled by sentiment and the *Reuters newswires* dataset with 11,228 newswires labeled over 46 topics. Keras loading functions format the data as a sequence of indices, where the indices point to words. The indices are offset by 3 to make room for indices representing special 'words': start, out-of-vocabulary and padding. Loading the *Large Movie Review Data* into a training and a test set as well as translating the indexed based data back to text can be performed with the following code:

```
from keras.datasets.imdb import get_word_index, load_data

# Read the text corpus as indices
(x_train, y_train), (x_test, y_test) = load_data(
    num_words=5000)

# Read the index to word mapping
index_to_word = {v: k for k, v in get_word_index().items()}
```

[11] https://keras.io/

[12] See the blog post "Using pre-trained word embeddings in a Keras model" at https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html.

```
# Add special translation indices for padding, start
# and out-of-vocabulary indicators
index_to_word.update({-3: 'PAD', -2: 'START', -1: 'OOV'})

# Translate indices to words and concatenate
review = "␣".join([index_to_word[index - 3]
                   for index in x_train[1]])
```

The beginning of the second review in the dataset (the `review` variable) will read "START big hair big OOV bad music and a giant safety OOV these are the words ..." Here the case, punctuation and markup have been removed from the original text: The original file[13] reads "Big hair, big boobs, bad music and a giant safety pin.....these are the words ..."

The DeepMoji pre-trained Keras model has been trained to predict emojis based on a very large English Twitter dataset [9].[14] The model predicts across 64 common emojis. `deepmoji_feature_encoding` loads the neural network excluding the last softmax layer so a prediction generates a 2304-dimensional feature space. The layers of the model can be used in other systems for semantic task. A recent system for emotion classification used the softmax layer and the attention layer together with domain adaptation as the winning entry in a competition for prediction of affect in messages from Twitter [8].

The Keras.js is a JavaScript library that support running Keras models in the web browser. This browser version may use the GPU through WebGL thus running at reasonable speeds.

12 Explicit creation of semantic representation

Software for construction of ontologies, e.g., for explicit semantic representations exists with, e.g., Protegé [22].[15] A recent development is Wikibase and its prime instance Wikidata [28], which is a collaborative environment for multilingual structured data, implemented around the Wikipedia software (MediaWiki). Users are able to describe concepts and, through properties, describe relations between the concepts. A new feature enables users to describe lexemes and their orthographic forms. Users can make properties of different type. In Wikidata, some of the properties are 'instance of', 'subclass of' and 'part of', so users of Wikidata can create multilingual concept hierarchies. A database engine is setup where users can query the knowledge graph with complex queries in the SPARQL query language, e.g., generating semantic networks on-the-fly. The SPARQL listing below generates the semantic hypernym network from the concept *chair*, see also Fig. 1.

```
#defaultView:Graph
```

[13] The original text can be found as a text file in *Large Movie Review Dataset* distributed from http://ai.stanford.edu/ amaas/data/sentiment/.
[14] https://github.com/bfelbo/DeepMoji
[15] https://protege.stanford.edu/.

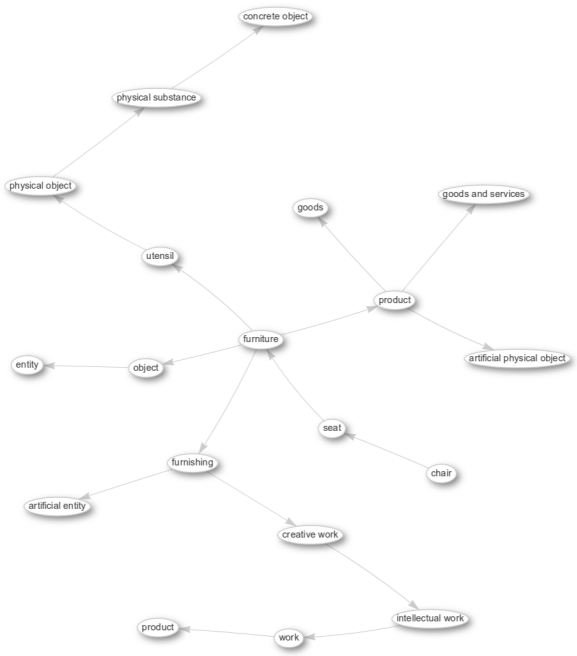

Fig. 1. Wikidata semantic hypernym network from the concept 'chair'.

```
SELECT ?child ?childLabel ?parent ?parentLabel
WHERE {
  SERVICE gas:service {
    gas:program gas:gasClass
      "com.bigdata.rdf.graph.analytics.BFS" ;
                gas:in wd:Q15026 ;
                gas:traversalDirection "Forward" ;
                gas:out ?parent ;
                gas:out2 ?child ;
                gas:linkType wdt:P279 ;
  }
  SERVICE wikibase:label {
    bd:serviceParam wikibase:language "en" . }
}
```

If the semantic graph representation is not used directly, then graph embedding can embed Wikidata items and properties in a low dimensional space via, e.g., node2vec, RDF2vec or Poincaré embedding [23]. Fig. 2 shows the result of a Poincaré embedding of the *furniture* hyponym network from Wikidata where the embedding space have been selected to have just two dimensions. The Python code below constructs this plot, first formulating a SPARQL query for the *fur-*

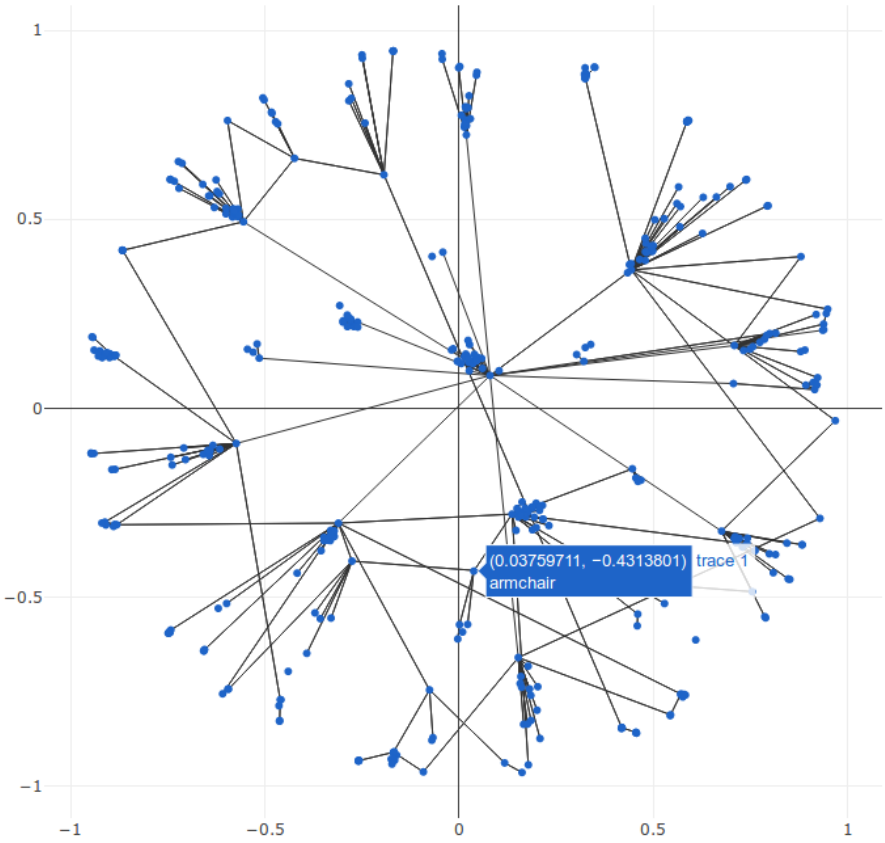

Fig. 2. Wikidata semantic network for *furniture* and its subclasses with concept position (the blue dots) determined by Gensim's Poincaré embedding where the dimension of the embedding space is two. For this particular trained model, the concept *armchair* appears at the coordinate $(0.04, -0.43)$ while the root concept *furniture* is close to the middle of the plot.

niture hyponym network, then downloading the data from the Wikidata Query Service and converting the results, lastly training and plotting with Gensim via its `PoincareModel`.

```
from gensim.models.poincare import PoincareModel
from gensim.viz.poincare import poincare_2d_visualization
from plotly.offline import plot
import requests

# Furniture graph query
sparql = """
```

```
SELECT
  ?furniture1 ?furniture1Label
  ?furniture2 ?furniture2Label
WHERE {
  ?furniture1 wdt:P279+ wd:Q14745 .
  {
    ?furniture2 wdt:P279+ wd:Q14745 .
    ?furniture1 wdt:P279 ?furniture2 .
  }
  UNION
  {
    BIND(wd:Q14745 AS ?furniture2)
    ?furniture1 wdt:P279 ?furniture2 .
  }
  ?furniture1 rdfs:label ?furniture1Label .
  ?furniture2 rdfs:label ?furniture2Label .
  FILTER (lang(?furniture1Label) = 'en')
  FILTER (lang(?furniture2Label) = 'en')
}"""

# Fetch data from Wikidata Query Service and convert
response = requests.get(
    "https://query.wikidata.org/sparql",
    params={'query': sparql, 'format': 'json'})
data = response.json()['results']['bindings']
relations = [
    (row['furniture1Label']['value'],
     row['furniture2Label']['value'])
    for row in data]

# Set up and train Poincare embedding model
model = PoincareModel(relations, size=2, negative=5)
model.train(epochs=100)

# Plot
plot_data = poincare_2d_visualization(model, relations, 'Furniture')
plot(plot_data)
```

We can further query the trained Gensim model for similarity, e.g., `model.kv.most_similar('armchair')` yields similar furniture for the *armchair* concept. With a particular trained model the most similar concepts are *recliner*, *Morris chair* and *fauteuil*, but also *gynecological chair* and *gas-discharge lamp*. The latter concept is in the hyponym network of furniture because *lamp* is regarded as a furniture.

The selection of the Poincaré embedding space to have just two dimensions is entire due to visualization. In the original work [23], the WordNet noun network

was modeled with embedding spaces from 5 to 100. Also note that the resulting embedding is depended upon the initialization and that the resulting configuration in Fig. 2 is not optimal (as some concept groups can be moved closer to higher concepts).

13 Acknowledgment

We would like to thank Innovation Fund Denmark for funding through the DABAI project.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I.J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Łukasz Kaiser, Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Vinyals, O., Warden, P., Wattenberg, M.M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (March 2016), https://arxiv.org/pdf/1603.04467.pdf
2. Al-Rfou, R., Perozzi, B., Skiena, S.: Polyglot: Distributed Word Representations for Multilingual NLP. Proceedings of the Seventeenth Conference on Computational Natural Language Learning pp. 183–192 (June 2014), https://arxiv.org/pdf/1307.1662.pdf
3. Bhatia, P., Guthrie, R., Eisenstein, J.: Morphological Priors for Probabilistic Neural Word Embeddings (September 2016), https://arxiv.org/pdf/1608.01056.pdf
4. Bird, S., Klein, E., Loper, E.: Natural Language Processing with Python (June 2009), http://www.nltk.org/book_1ed/
5. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching Word Vectors with Subword Information (July 2016), https://arxiv.org/pdf/1607.04606.pdf
6. Bérard, A., Servan, C., Pietquin, O., Besacier, L.: MultiVec: a Multilingual and Multilevel Representation Learning Toolkit for NLP. Proceedings of the 10th edition of the Language Resources and Evaluation Conference (2016), http://www.lrec-conf.org/proceedings/lrec2016/pdf/666_Paper.pdf
7. Choi, J.D., Tetreault, J., Stent, A.: It Depends: Dependency Parser Comparison Using A Web-based Evaluation Tool. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing pp. 387–396 (July 2015), http://www.aclweb.org/anthology/P15-1038
8. Duppada, V., Jain, R., Hiray, S.: SeerNet at SemEval-2018 Task 1: Domain Adaptation for Affect in Tweets (2018), https://static1.squarespace.com/static/58e3ecc75016e194dd5125b0/t/5aaabbc02b6a28802e380940/1521138626662/adaptation-affect-tweets.pdf
9. Felbo, B., Mislove, A., Søgaard, A., Rahwan, I., Lehmann, S.: Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing pp. 1616–1626 (August 2017), http://aclweb.org/anthology/D17-1169

10. Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N., Peters, M., Schmitz, M., Zettlemoyer, L.: AllenNLP: A Deep Semantic Natural Language Processing Platform (2017), http://allennlp.org/papers/AllenNLP_white_paper.pdf
11. Grave, E., Bojanowski, P., Gupta, P., Joulin, A., Mikolov, T.: Learning Word Vectors for 157 Languages. Proceedings of the 11th edition of the Language Resources and Evaluation Conference (February 2018), https://arxiv.org/pdf/1802.06893.pdf
12. Grover, A., Leskovec, J.: node2vec: Scalable Feature Learning for Networks. Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining 2016, 855–864 (August 2016), https://arxiv.org/pdf/1607.00653.pdf
13. Honnibal, M.: A Good Part-of-Speech Tagger in about 200 Lines of Python (September 2013), https://explosion.ai/blog/part-of-speech-pos-tagger-in-python
14. Ji, S., Yun, H., Yanardag, P., Matsushima, S., Vishwanathan, S.V.N.: WordRank: Learning Word Embeddings via Robust Ranking. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (September 2016), https://arxiv.org/pdf/1506.02761.pdf
15. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional Architecture for Fast Feature Embedding. Proceedings of the 22nd ACM international conference on Multimedia (June 2014), https://arxiv.org/pdf/1408.5093.pdf
16. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of Tricks for Efficient Text Classification (August 2016), https://arxiv.org/pdf/1607.01759.pdf
17. Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies pp. 142–150 (June 2011)
18. Mehdi, M., Okoli, C., Mesgari, M., Nielsen, F.Å., Lanamäki, A.: Excavating the mother lode of human-generated text: A systematic review of research that uses the Wikipedia corpus. Information Processing & Management 53, 505–529 (March 2017)
19. Mikolov, T., Chen, K., Corrado, G.S., Dean, J.: Efficient Estimation of Word Representations in Vector Space (January 2013), https://arxiv.org/pdf/1301.3781v3
20. Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., Joulin, A.: Advances in Pre-Training Distributed Word Representations (December 2017), https://arxiv.org/pdf/1712.09405.pdf
21. Miller, G.A.: WordNet: a lexical database for English. Communications of the ACM 38, 39–41 (November 1995)
22. Musen, M.A., Team, P.: The Protégé Project: A Look Back and a Look Forward. AI matters 1, 4–12 (June 2015)
23. Nickel, M., Kiela, D., Kiela, D.: Poincaré Embeddings for Learning Hierarchical Representations. Advances in Neural Information Processing Systems 30 (May 2017), https://arxiv.org/pdf/1705.08039.pdf
24. Nielsen, F.Å.: Wembedder: Wikidata entity embedding web service (October 2017), https://arxiv.org/pdf/1710.04099
25. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Édouard Duchesnay: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, 2825–2830 (October 2011), http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

26. Pennington, J., Socher, R., Manning, C.D.: GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) p. 1532–1543 (2014), http://www.emnlp2014.org/papers/pdf/EMNLP2014162.pdf
27. Smedt, T.D., Daelemans, W.: Pattern for Python. Journal of Machine Learning Research 13, 2031–2035 (2012), http://www.jmlr.org/papers/volume13/desmedt12a/desmedt12a.pdf
28. Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Communications of the ACM 57, 78–85 (October 2014), http://cacm.acm.org/magazines/2014/10/178785-wikidata/fulltext
29. Řehůřek, R., Sojka, P.: Software framework for topic modelling with large corpora. New Challenges For NLP Frameworks Programme pp. 45–50 (May 2010), https://radimrehurek.com/gensim/lrec2010_final.pdf