Semantic Entity Retrieval Toolkit

Van Gysel, C.; de Rijke, M.; Kanoulas, E.

Published in:
Neu-IR: Workshop on Neural Information Retrieval

Semantic Entity Retrieval Toolkit*

Christophe Van Gysel
University of Amsterdam
Amsterdam, The Netherlands
cvangysel@uva.nl

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
derijke@uva.nl

Evangelos Kanoulas
University of Amsterdam
Amsterdam, The Netherlands
e.kanoulas@uva.nl

ABSTRACT

Unsupervised learning of low-dimensional, semantic representations of words and entities has recently gained attention. In this paper we describe the Semantic Entity Retrieval Toolkit (SERT) that provides implementations of our previously published entity representation models. The toolkit provides a unified interface to different representation learning algorithms, fine-grained parsing configuration and can be used transparently with GPUs. In addition, users can easily modify existing models or implement their own models in the framework. After model training, SERT can be used to rank entities according to a textual query and extract the learned entity/word representation for use in downstream algorithms, such as clustering or recommendation.

KEYWORDS

Neural information retrieval; Entity retrieval; Toolkit

1 INTRODUCTION

The unsupervised learning of low-dimensional, semantic representations of words and entities has recently gained attention for the entity-oriented tasks of expert finding [9] and product search [8]. Representations are learned from a document collection and domain-specific associations between documents and entities. Expert finding is the task of finding the right person with the appropriate skills or knowledge [1] and an association indicates document authorship (e.g., academic papers) or involvement in a project (e.g., annual progress reports). In the case of product search, an associated document is a product description or review [8].

In this paper we describe the Semantic Entity Retrieval Toolkit (SERT) that provides implementations of our previously published entity representation models [8, 9]. Beyond a unified interface that combines different models, the toolkit allows for fine-grained parsing configuration and GPU-based training through integration with Theano [3, 6]. Users can easily extend existing models or implement their own models within the unified framework. After model training, SERT can compute matching scores between an entity and a piece of text (e.g., a query). This matching score can then be used for ranking entities, or as a feature in a downstream machine learning system, such as the learning to rank component of a search engine. In addition, the learned representations can be extracted and used as feature vectors in entity clustering or recommendation tasks [10]. The toolkit is licensed under the permissive MIT open-source license; see the footnote on the first page.

*The toolkit is licensed under the permissive MIT open-source license and can be found at https://github.com/cvangysel/SERT.

SIGIR 2017 Workshop on Neural Information Retrieval (Neu-IR'17), August 7–11, 2017, Shinjuku, Tokyo, Japan

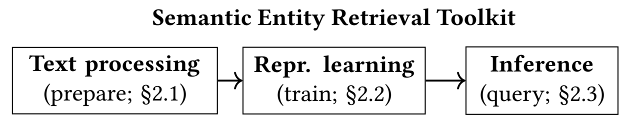

Figure 1: Schematic overview of the different pipeline components of SERT. The collection is parsed, processed and packaged in a numerical format using the *prepare* (§2.1) utility. Afterwards, the *training* (§2.2) utility learns representations of entities and words and the *query* (§2.3) utility is used to compute matching scores between entities and queries.

2 THE TOOLKIT

SERT is organized as a pipeline of utilities as depicted in Fig. 1. First, a collection of documents and entity associations is processed and packaged using a numerical format (§2.1). Low-dimensional representations of words and entities are then learned (§2.2) and afterwards the representations can be used to make inferences (§2.3).

2.1 Collection parsing and preparation

To begin, SERT constructs a vocabulary that will be used to tokenize the document collection. Non-significant words that are too frequent (e.g., stopwords), noisy (e.g., single characters) and rare words are filtered out. Words that do not occur in the dictionary are ignored. Afterwards, word sequences are extracted from the documents and stored together with the associated entities in the numerical format provided by NumPy [7]. Word sequences can be extracted consecutively or a stride can be specified to extract non-consecutive windows. In addition, a hierarchy of word sequence extractors can be applied to extract skip-grams, i.e., word sequences where a number of tokens are skipped after selecting a token [4]. To support short documents, a special-purpose padding token can be used to fill up word sequences that are longer than a particular document.

After word sequence extraction, a weight can be assigned to each word sequence/entity pair that can be used to re-weight the training objective. For example, in the case of expert finding [9], this weight is the reciprocal of the document length of the document where the sequence was extracted from. This avoids a bias in the objective towards long documents. An alternative option that exists within the toolkit is to resample word sequence/entity pairs such that every entity is associated with the same number of word sequences, as used for product search [8].

```python
class ExampleModel(VectorSpaceLanguageModelBase):

    def __init__(self, *args, **kwargs):
        super(ExampleModel, self).__init__(
            *args, **kwargs)

        # Define model architecture.
        input_layer = InputLayer(
            shape=(self.batch_size, self.window_size))

        ...

        def loss_fn(pred, actual, _):
            # Compute symbolic loss between
            # predicted/actual entities.

        # The framework deals with underlying boilerplate.
        self._finalize(loss_fn, ....)

    def get_representations(self):
        # Returns the representations and parameters
        # to be extracted.
```

Code snippet 1: Illustrative example of the SERT model interface. The full interface supports more functionality omitted here for brevity. Users can define a symbolic graph of computation using the Theano library [6] in combination with Lasagne [3].

2.2 Representation learning

After the collection has been processed and packaged in a machine-friendly format, representations of words and entities can be learned. The toolkit includes implementations of state-of-the-art representation learning models that were applied to expert finding [9] and product search [8]. Users of the toolkit can use these implementations to learn representations out-of-the-box or adapt the algorithms to their needs. In addition, users can implement their own models by extending an interface provided by the framework. Code snippet 1 shows an example of a model implemented in the SERT toolkit where users can define a symbolic cost function that will be optimized using Theano [6]. Due to the component-wise organization of the toolkit (Fig. 1), modeling and text processing are separated from each other. Consequently, researchers can focus on modeling and representation learning only. In addition, any improvements to the collection processing (§2.1) collectively benefits all models implemented in SERT.

2.3 Entity ranking & other uses of the representations

Once a model has been trained, SERT can be used to rank entities w.r.t. a textual query. The concrete implementation used to rank entities depends on the model that was trained. In the most generic case, a matching score is computed for every entity and entities are ranked in decreasing order of his score. However, in the special case when the model is interpreted as a metric vector space [2, 8], SERT casts entity ranking as a k-nearest neighbor problem and uses specialized data structures for retrieval [5]. After ranking, SERT outputs the entity rankings as a TREC-compatible file that can be used as input to the `trec_eval`[1] evaluation utility.

Apart from entity ranking, the learned representations and model-specific parameters can be extracted conveniently from the models through the interface[2] and used for down-stream tasks such as clustering, recommendation and determining entity importance as shown in [10].

3 CONCLUSIONS

In this paper we described the Semantic Entity Retrieval Toolkit, a toolkit that learns latent representations of words and entities. The toolkit contains implementations of state-of-the-art entity representations algorithms [8, 9] and consists of three components: text processing, representation learning and inference. Users of the toolkit can easily make changes to existing model implementations or contribute their own models by extending an interface provided by the SERT framework.

Future work includes integration with Pyndri [11] such that document collections indexed with Indri can transparently be used to train entity representations. In addition, integration with machine learning frameworks besides Theano, such as TensorFlow and PyTorch, will make it easier to integrate existing models into SERT.

Acknowledgments. The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This research was supported by Ahold Delhaize, Amsterdam Data Science, the Bloomberg Research Grant program, the Criteo Faculty Research Award program, the Dutch national program COMMIT, Elsevier, the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 312827 (VOX-Pol), the Google Faculty Research Award scheme, the Microsoft Research Ph.D. program, the Netherlands Institute for Sound and Vision, the Netherlands Organisation for Scientific Research (NWO) under project nrs 612.001.116, HOR-11-10, CI-14-25, 652.002.001, 612.001.551, 652.-001.003, and Yandex. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] K. Balog, Y. Fang, M. de Rijke, P. Serdyukov, and L. Si. Expertise retrieval. *Found. & Tr. in Inform. Retr.*, 6(2-3):127–256, 2012.
[2] L. Boytsov, D. Novak, Y. Malkov, and N. Eric. Off the beaten path: Let's replace term-based retrieval with k-nn search. In *CIKM*, 2016.
[3] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, et al. Lasagne: First release., Aug. 2015.
[4] D. Guthrie, B. Allison, W. Liu, L. Guthrie, and Y. Wilks. A closer look at skip-gram modelling. In *LREC*, 2006.
[5] A. M. Kibriya and E. Frank. An empirical comparison of exact nearest neighbour algorithms. In *ECMLPKDD*, pages 140–151. Springer, 2007.
[6] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.
[7] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2): 22–30, 2011.
[8] C. Van Gysel, M. de Rijke, and E. Kanoulas. Learning latent vector spaces for product search. In *CIKM*. ACM, 2016.
[9] C. Van Gysel, M. de Rijke, and M. Worring. Unsupervised, efficient and semantic expertise retrieval. In *WWW*. ACM, 2016.
[10] C. Van Gysel, M. de Rijke, and E. Kanoulas. Structural regularities in expert vector spaces. In *ICTIR*. ACM, 2017.
[11] C. Van Gysel, E. Kanoulas, and M. de Rijke. Pyndri: a python interface to the indri search engine. In *ECIR*, volume 2017. Springer, 2017.

[1] https://github.com/usnistgov/trec_eval

[2] See `get_representations` in Snippet 1.