Going Beyond T-SNE: Exposing `whatlies` in Text Embeddings

Vincent D. Warmerdam
Rasa
Schönhauser Allee 175
10119 Berlin
v.warmerdam@rasa.com

Thomas Kober
Rasa
Schönhauser Allee 175
10119 Berlin
t.kober@rasa.com

Rachael Tatman
Rasa
Schönhauser Allee 175
10119 Berlin
r.tatman@rasa.com

Abstract

We introduce `whatlies`, an open source toolkit for visually inspecting word and sentence embeddings. The project offers a unified and extensible API with current support for a range of popular embedding backends including spaCy, tfhub, huggingface transformers, gensim, fastText and BytePair embeddings. The package combines a domain specific language for vector arithmetic with visualisation tools that make exploring word embeddings more intuitive and concise. It offers support for many popular dimensionality reduction techniques as well as many interactive visualisations that can either be statically exported or shared via Jupyter notebooks. The project documentation is available from https://rasahq.github.io/whatlies/.

1 Introduction

The use of pre-trained word embeddings (Mikolov et al., 2013a; Pennington et al., 2014) or language model based sentence encoders (Peters et al., 2018; Devlin et al., 2019) has become a ubiquitous part of NLP pipelines and end-user applications in both industry and academia. At the same time, a growing body of work has established that pre-trained embeddings codify the underlying biases of the text corpora they were trained on (Bolukbasi et al., 2016; Garg et al., 2018; Brunet et al., 2019). Hence, practitioners need tools to help select which set of embeddings to use for a particular project, detect potential need for debiasing and evaluate the debiased embeddings. Simplified visualisations of the latent semantic space provide an accessible way to achieve this.

Therefore we created `whatlies`, a toolkit offering a programmatic interface that supports vector arithmetic on a set of embeddings and visualising the space after any operations have been carried out. For example, Figure 1 shows an example of how representations for *queen*, *king*, *man*, and *woman* can be projected along the axes $v_{\text{queen}-\text{king}}$ and $v_{\text{man}|\text{queen}-\text{king}}$ in order to derive a visualisation of the space along the projections.

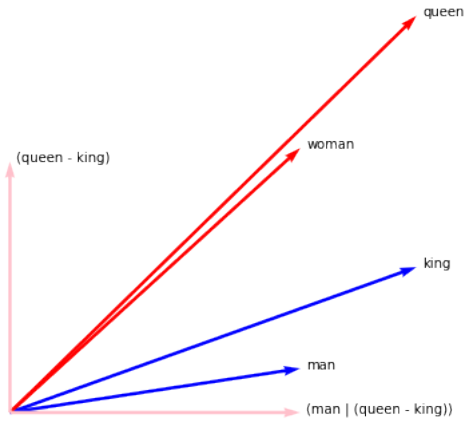

Figure 1: Projections of w_{king}, w_{queen}, w_{man}, $w_{\text{queen}} - w_{\text{king}}$ and w_{man} projected away from $w_{\text{queen}} - w_{\text{king}}$. Both the vector arithmetic and the visualisation were done using the `whatlies`. The support for arithmetic expressions is integral in `whatlies` because it leads to more meaningful visualisations and concise code.

Perhaps the most widely known tool for visualising embeddings is the tensorflow projector[1] which offers 3D visualisations of any input embeddings. The visualisations are useful for understanding the emergence of clusters and the neighbourhood of certain words and the overall space. However, the projector is limited to dimensionality reduction as the sole preprocessing method. More recently, Molino et al. (2019) have introduced parallax which allows explicit selection of the axes on which to project a representation. This creates an additional level of flexibility as these axes can also be derived from arithmetic operations on the embeddings.

The major difference between the tensorflow pro-

[1] https://projector.tensorflow.org/

Proceedings of Second Workshop for NLP Open Source Software (NLP-OSS), pages 52–60

jector, parallax and whatlies is that the first two provide a non-extensible browser-based interface, whereas whatlies provides a programmatic one. Therefore whatlies can be more easily extended to any specific practical need and cover individual use-cases. The goal of whatlies is to offer a set of tools that can be used from a Jupyter notebook with a range of visualisation capabilities that goes beyond the commonly used static T-SNE (van der Maaten and Hinton, 2008) plots. whatlies can be installed via pip, the code is available from https://github.com/RasaHQ/whatlies[2] and the documentation is hosted at https://rasahq.github.io/whatlies/.

2 What lies in whatlies — Usage and Examples

Embedding backends. The current version of whatlies supports word-level as well as sentence-level embeddings in any human language that is supported by the following libraries:

- BytePair embeddings (Sennrich et al., 2016) via the BPemb project (Heinzerling and Strube, 2018)
- fastText (Bojanowski et al., 2017)
- gensim (Řehůřek and Sojka, 2010)
- huggingface (Wolf et al., 2019)
- sense2vec (Trask et al., 2015); via spaCy
- spaCy[3]
- tfhub[4]

Embeddings are loaded via a unified API:

```
from whatlies.language import \
SpacyLanguage, FasttextLanguage, \
TFHubLanguage, HFTransformersLangauge

# spaCy
lang_sp = SpacyLanguage('en_core_web_md')
emb_king = lang_sp["king"]
emb_queen = lang_sp["queen"]

# fastText
ft = 'cc.en.300.bin'
lang_ft = FasttextLanguage(ft)
emb_ft = lang_ft['pizza']

# TF-Hub
tf_hub = 'https://tfhub.dev/google/'
model = tf_hub + 'nnlm-en-dim50/2'
lang_tf = TFHubLanguage(model)
emb_tf = lang_tf['whatlies is awesome']

# Huggingface
bert = 'bert-base-cased'
lang_hf = HFTransformersLanguage(bert)
emb_hf = lang['whatlies rocks']
```

Retrieved embeddings are python objects that contain a vector and an associated named. It comes with extra utility methods attached that allow for easy arithmetic and visualisation.

The library is capable of retreiving embeddings for sentences too. In order to retrieve a sentence representation for word-level embeddings such as fastText, whatlies returns the summed representation of the individual word vectors. For pre-trained encoders such as BERT (Devlin et al., 2019) or ConveRT (Henderson et al., 2019), whatlies uses its internal [CLS] token for representing a sentence.

Similarity Retrieval. The library also supports retrieving similar items on the basis of a number of commonly used distance/similarity metrics such as cosine or Euclidean distance:

```
from whatlies.language import \
SpacyLanguage

lang = SpacyLanguage('en_core_web_md')

lang.score_similar("man", n=5,
                   metric='cosine')
[(Emb[man], 0.0),
 (Emb[woman], 0.2598254680633545),
 (Emb[guy], 0.29321062564849854),
 (Emb[boy], 0.2954298257827759),
 (Emb[he], 0.3168887495994568)]
 # NB: Results are cosine _distances_
```

Vector Arithmetic. Support of arithmetic expressions on embeddings is integral in any whatlies functions. For example the code for creating Figure 1 from the Introduction highlights that it does not make a difference whether the plotting functionality is invoked on an embedding itself or on a representation derived from an arithmetic operation:

```
import matplotlib.pylab as plt
from whatlies import Embedding

man   = Embedding("man", [0.5, 0.1])
woman = Embedding("woman", [0.5, 0.6])
king  = Embedding("king", [0.7, 0.33])
queen = Embedding("queen", [0.7, 0.9])
man.plot(kind="arrow", color="blue")
woman.plot(kind="arrow", color="red")
king.plot(kind="arrow", color="blue")
queen.plot(kind="arrow", color="red")
```

[2] Community PRs are greatly appreciated ☺.
[3] https://spacy.io/
[4] https://www.tensorflow.org/hub

```
diff = (queen - king)
orth = (man | (queen - king))

diff.plot(color="pink",
          show_ops=True)
orth.plot(color="pink",
          show_ops=True)
# See Figure 1 for the result :)
```

This feature allows users to construct custom queries and use it e.g. in combination with the similarity retrieval functionality. For example, we can validate the widely circulated analogy of Mikolov et al. (2013b) on spaCy's medium English model in only 4 lines of code (including imports):

$$w_{\text{queen}} \approx w_{\text{king}} - w_{\text{man}} + w_{\text{woman}}$$

```
from whatlies.language import \
SpacyLanguage

lang = SpacyLanguage('en_core_web_md')

> e = lang["king"] - lang["man"] + \
lang["woman"]
> lang.score_similar(e, n=5,
                     metric='cosine')
[(Emb[king], 0.19757413864135742),
 (Emb[queen], 0.2119154930114746),
 (Emb[prince], 0.35989218950271606),
 (Emb[princes], 0.37914562225341797),
 (Emb[kings], 0.37914562225341797)]
```

Excluding the query word *king*[5], the analogy returns the anticipated result: *queen*.

The library also allows the user to add/subtract embeddings but also project unto (via the $>$ operator) or away from them (via the $|$ operator). This means that the user is very flexible when it comes to retrieving embeddings.

Multilingual Support. `whatlies` supports any human language that is available from its current list of supported embedding backends. This allows us to check the royal analogy from above in languages other than English. The code snippet below shows the results for Spanish and Dutch, using pre-trained fastText embeddings[6].

```
from whatlies.language import \
FasttextLanguage
es = FasttextLanguage("cc.es.300.bin")
nl = FasttextLanguage("cc.nl.300.bin")

emb_es = es["rey"] - es["hombre"] + \
es["mujer"]
emb_nl = nl["koning"] - nl["man"] + \
nl["vrouw"]

es.score_similar(emb_es, n=5,
                 metric='cosine')
[(Emb[rey], 0.04499000310897827),
(Emb[monarca], 0.24673408269882202),
(Emb[Rey], 0.2799408435821533),
(Emb[reina], 0.2993239760398865),
(Emb[príncipe], 0.3025314211845398)]

nl.score_similar(emb_nl, n=5,
                 metric='cosine')

[(Emb[koning], 0.48337286710739136),
(Emb[koningen], 0.5858825445175171),
(Emb[koningin], 0.6115483045578003),
(Emb[Koning], 0.6155656576156616),
(Emb[kroonprins], 0.658723771572113)]
```

While for Spanish, the correct answer *reina* is only at rank 3 (excluding *rey* from the list), the second ranked *monarca* (female form of *monarch*) is getting close. For Dutch, the correct answer *koningin* is at rank 2, surpassed only by *koningen* (plural of *king*). Another interesting observation is that the cosine distances — even of the query words — vary wildly in the embeddings for the two languages.

Sets of Embeddings. In the previous examples we have typically only retrieved single embeddings. However, `whatlies` also supports the notion of an "Embedding Set", that can hold any number of embeddings:

```
from whatlies.language import \
SpacyLanguage

lang = SpacyLanguage("en_core_web_lg")

words = ["prince", "princess", "nurse",
         "doctor", "man", "woman",
         "sentences also embed"]
# NB: 'sentences also embed' will be
#     represented as the mean of the
#     3 individual words. This behavior
#.    is driven by spaCy currently.

emb = lang[words]
```

It is often more useful to analyse a set of embeddings at once, rather than many individual ones. Therefore, any arithmetic operations that can be applied to single embeddings, can also be applied to all of the embeddings in a given set.

The `emb` variable in the previous code example represents an `EmbeddingSet`. These are collections of embeddings which can be simpler to analyse than many individual variables. Users can, for example, apply vector arithmetic to the entire `EmbeddingSet`.

```
new_emb = emb | (emb['man'] - emb['woman'])
```

[5] As appears to be standard practice in word analogy evaluation (Levy and Goldberg, 2014).

[6] The embeddings are available from https://fasttext.cc/docs/en/crawl-vectors.html.

Visualisation Tools. Any visualisations in `whatlies` are most useful when performed on `EmbeddingSets`. They offer a variety of methods for plotting, such as the distance map in Figure 2:

```
words = ['man', 'woman', 'king', 'queen',
         'red', 'green', 'yellow']
emb = lang[words]
emb.plot_distance(metric='cosine')
```

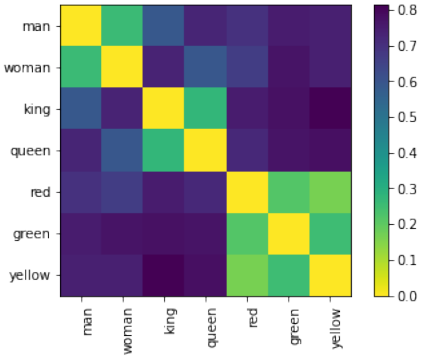

Figure 2: Pairwise distances for a set of words using cosine distance.

`whatlies` also offers interactive visualisations using "Altair" as a plotting backend[7]:

```
emb.plot_interactive(x_axis="man",
                     y_axis="yellow",
                     show_axis_point=True)
```

The above code snippet projects every vector in the `EmbeddingSet` onto the vectors on the specified axes. This creates the values we can use for 2D visualisations. For example, given that *man* is on the x-axis the value for 'yellow' on that axis will be:

$$v(\text{yellow} \rightarrow \text{man}) = \frac{w_{\text{yellow}} \cdot w_{\text{man}}}{w_{\text{man}} \cdot w_{\text{man}}}$$

which results in Figure 3.

These plots are built on top of Altair (VanderPlas et al., 2018) and are fully interactive. It is possible to click and drag in order to navigate through the embedding space and zoom in and out. These plots can be hosted on a website but they can also be exported to `png`/`svg` for publication. It is furthermore possible to apply any vector arithmetic operations for these plots, resulting in Figure 4:

```
e = emb["man"] - emb["woman"]
emb.plot_interactive(x_axis=e,
                     y_axis="yellow",
                     show_axis_point=True)
```

[7]Examples of the interactive visualisations can be seen on the project's github page: https://github.com/RasaHQ/whatlies

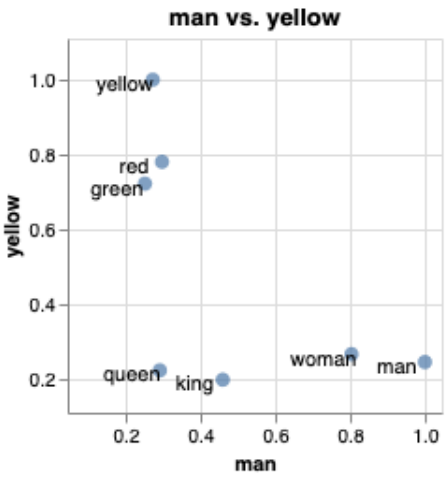

Figure 3: Plotting example terms along the axes *man* vs. *yellow*. Note how the title/axes automatically update.

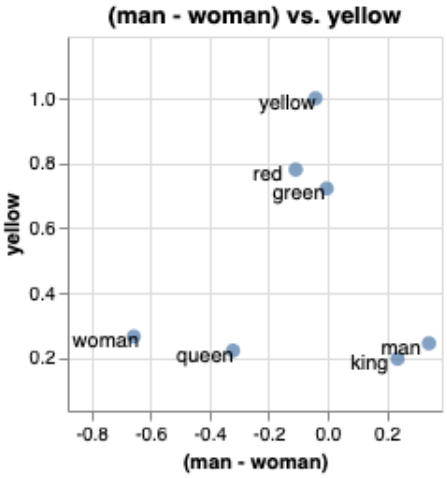

Figure 4: Plotting example terms along the transformed *man* - *woman* axis and the *yellow* axis.

Transformations. `whatlies` also supports several techniques for dimensionality reduction of `EmbeddingSets` prior to plotting. This is demonstrated in Figure 5 below.

```
from whatlies.transformers import Pca
from whatlies.transformers import Umap

p1 = (emb
      .transform(Pca(2))
      .plot_interactive())
p2 = (emb
      .transform(Umap(2))
      .plot_interactive())
p1 | p2
```

Transformations in `whatlies` are slightly different than for example scikit-learn transformations because in addition to dimensionality reduction, the transformation can also add embeddings that represent each principal component to the

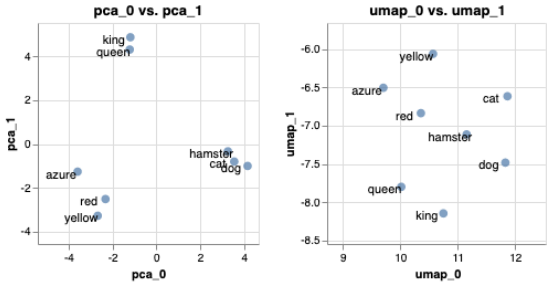

Figure 5: Demonstration of PCA and UMAP transformations.

`EmbeddingSet` object. As a result, they can be referred to as axes for creating visualisations as seen in Figure 5.

Scikit-Learn Integration. To facilitate quick exploration of different word embeddings we have also made our library compatible with scikit-learn (Pedregosa et al., 2011). The Rasa library uses numpy (Harris et al., 2020) to represent the numerical vectors associated to the input text. This means that it is possible to use the `whatlies` embedding backends as feature extractors in scikit-learn pipelines, as the code snippet below shows[8]:

```
from whatlies.language import \
BytePairLanguage
from sklearn.pipeline import Pipeline

pipe = Pipeline([
    ("embed", BytePairLanguage("en")),
    ("model", LogisticRegression())
])

X = [
    "i really like this post",
    "thanks for that comment",
    "i enjoy this friendly forum",
    "this is a bad post",
    "i dislike this article",
    "this is not well written"
]

y = np.array([1, 1, 1, 0, 0, 0])

pipe.fit(X, y).predict(X)
```

This feature enables fast exploration of many different word embedding algorithms.[9]

3 A Tale of two Use-cases

Visualising Bias. One use-case of `whatlies` is to gain insight into bias-related issues in an embedding space. Because the library readily supports vector arithmetic it is possible to create an `EmbeddingSet` holding pairs of representations:

```
lang = SpacyLanguage("en_core_web_lg")

emb_of_pairs = EmbeddingSet(
    (lang["nurse"] - lang["doctor"]),
    (lang["nurse"] - lang["surgeon"]),
    (lang["woman"] - lang["man"]),
)
```

Subsequently, the new `EmbeddingSet` can be visualised as a distance map as in Figure 6, revealing a number of spurious correlations that suggest a gender bias in the embedding space.

```
emb_of_pairs.plot_distance(metric="cosine")
```

Visualising issues in the embedding space like this creates an effective way to communicate potential risks of using embeddings in production to non-technical stakeholders.

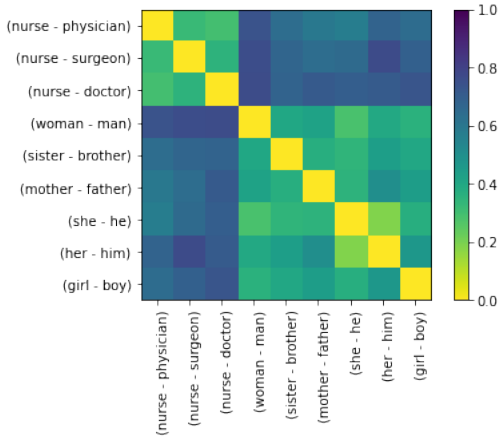

Figure 6: Distance map for visualising bias. If there was no bias then we would expect 'she-he' to have a distance near 1.0 compared to 'nurse-physician'. The figure shows this is not the case.

It is possible to apply the debiasing technique introduced by Bolukbasi et al. (2016) in order to approximately remove the direction corresponding to gender. The code snippet below achieves this by, again, using the arithmetic notation.

```
lang = SpacyLanguage("en_core_web_lg")

emb = lang[words]
axis = EmbeddingSet(
    (lang['man'] - lang['woman']),
    (lang['king'] - lang['queen']),
    (lang['father'] - lang['mother'])
).average()
emb_debias = emb | axis
```

Figure 7 shows the result of applying the debiasing technique, highlighting that some of the spurious correlations have indeed been removed.

[8] Note that this is an illustrative example and we do not recommend to train and test on the same data.

[9] At the moment, however, it is not yet possible to use the `whatlies` embeddings in conjunction with scikit-learn's grid search functionality.

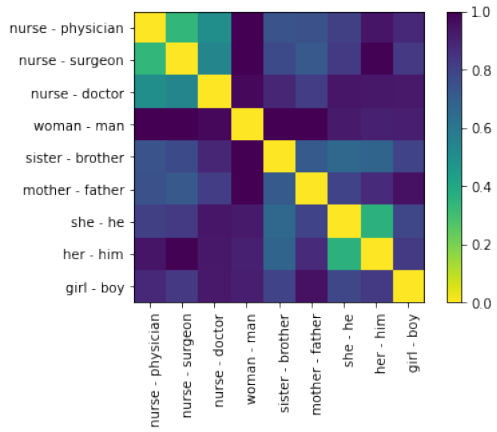

Figure 7: Distance map for visualising the embedding space after the debiasing technique of Bolukbasi et al. (2016) has been applied.

It is important to note though, that the above technique does not reliably remove all relevant bias in the embeddings and that bias is still measurably existing in the embedding space as Gonen and Goldberg (2019) have shown. This can be verified with `whatlies`, by plotting the neighbours of the biased and debiased space:

```
emb.score_similar("maid", n=7)

[(Emb[maid], 0.0),
 (Emb[maids], 0.18290925025939941),
 (Emb[housekeeper], 0.2200336456298828),
 (Emb[maidservant], 0.3770867586135864),
 (Emb[butler], 0.3822709918022156),
 (Emb[mistress], 0.3967094421386719),
 (Emb[servant], 0.40112364292144775)]

 emb_debias.score_similar("maid", n=7)

[(Emb[maid], 0.0),
 (Emb[maids], 0.18163418769836426),
 (Emb[housekeeper], 0.21881639957427979),
 (Emb[butler], 0.3642127513885498),
 (Emb[maidservant], 0.3768376111984253),
 (Emb[servant], 0.382546067237854),
 (Emb[mistress], 0.3955296277999878)]
```

As the output shows, the neighbourhoods of *maid* in the biased and debiased space are almost equivalent, with e.g. *mistress* still appearing relatively high-up the nearest neighbours list.

Comparing Embedding Backends. Another use-case for `whatlies` is for comparing different embeddings. For example, we wanted to analyse two different encoders for their ability to capture the intent of user utterances in a task-based dialogue system. We compared spaCy and the Universal Sentence Encoder for their ability to embed sentences from the same intent class close together in space. Figure 8 shows that the utterances encoded with the Universal Sentence Encoder form more coherent clusters.

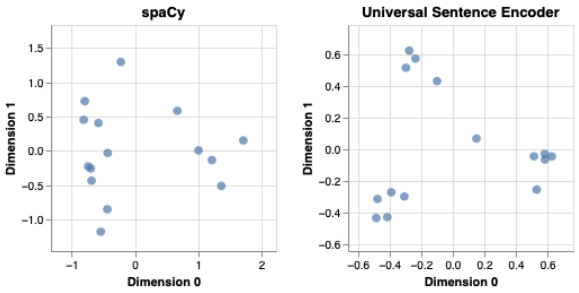

Figure 8: Side-by-side comparison of spaCy and Universal Sentence Encoder for embedding example sentences from 3 different intent classes. Universal Sentence Encoder embeds the sentences into relatively tight and coherent clusters, whereas class boundaries are more difficult to see with spaCy.

Figure 9 highlights the same trend with a distance map, where for spaCy there is barely any similarity between the utterances, the coherent clusters from Figure 8 are well reflected in the distance map for the Universal Sentence Encoder.

The superiority of Universal Sentence Encoder in comparison to spaCy for this example is expected, though, as it is aimed at sentences, but it is certainly useful to have a tool — `whatlies` — at one's disposal with which it is possible to quickly validate this.

4 Roadmap

`whatlies` is in active development. While we cannot predict the contents of future community PRs, this is our current roadmap for future development:

- We want to make it easier for people to research bias in word embeddings. We will continue to investigate if there are visualisation techniques that can help spot issues and we aim to make any robust debiasing techniques available in `whatlies`.
- We would like to curate labelled sets of word lists for attempting to quantify the amount of bias in a given embedding space. Properly labelled word lists can be useful for algorithmic bias research but it might also help understand clusters. We plan to make any evaluation resources available via this package.
- One limit of using Altair as a visualisation library is that we cannot offer interactive visualisations with many thousands of data points.

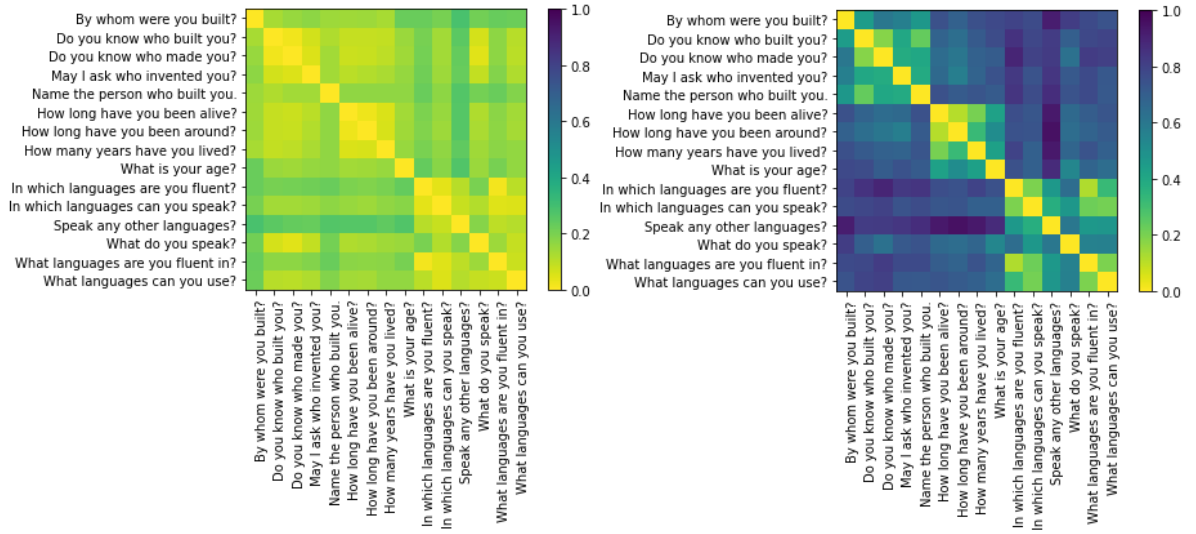

Figure 9: Side-by-side comparison of spaCy and Universal Sentence Encoder for embedding example sentences from 3 different intent classes. The distance map highlights the "clustery" behaviour of Universal Sentence Encoder, where class membership is nicely reflected in the intra-class distances. For spaCy on the other hand, there is less difference between intra-class vs. inter-class distances.

We might explore other visualisation tools for this library as well.

- Since we're supporting dynamic backends like BERT at the sentence level, we are aiming to also support these encoders at the word level, which requires us to specify an API for retrieving contextualised word representations within `whatlies`. We are currently exploring various ways for exposing this feature and are working with a notation that uses square brackets that can select an embedding from the context of the sentence that it resides in:

```
mod_name = "en_trf_robertabase_lg"
lang = SpacyLanguage(mod_name)
emb1 = lang['[bank] of the river']
emb2 = lang['money on the [bank]']
assert emb1.vector != emb2.vector
```

  At the moment we only support spaCy backends with this notation but we plan to explore this further with other embedding backends.[10]

- A related issue is that not every vocabulary based back-end uses the same method of pooling word-embeddings to represent a sentence. Some take the sum, while others take the mean and others introduce yet another standard. Our goal for vocabulary based back-ends is to allow the user to control this manually for consistency.

5 Conclusion

We have introduced `whatlies`, a python library for inspecting word and sentence embeddings that is very flexible due to offering a programmable interface. We currently support a variety of embedding models, including fastText, spaCy, BERT, or the Universal Sentence Encoder. This paper has showcased its current use as well as plans for future development. The project is hosted at `https://github.com/RasaHQ/whatlies` and we are happy to receive community contributions that extend and improve the package.

Acknowledgements

Despite being only a few months old the project has started getting traction on github and has attracted the help of outside contributions. In particular we'd like to thank Masoud Kazemi for many contributions to the project.

We would furthermore like to thank Adam Lopez for many rounds of discussion that considerably improved the paper.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

[10] Ideally we also introduce the necessary notation for retrieving the contextualised embedding from a particular layer, e.g. `lang['bank'][2]` for obtaining the representation of *bank* from the second layer of the given language model.

Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. 2016. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 4356–4364, USA. Curran Associates Inc.

Marc-Etienne Brunet, Colleen Alkalay-Houlihan, A. Anderson, and R. Zemel. 2019. Understanding the origins of bias in word embeddings. In *ICML*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Nikhil Garg, Londa Schiebinger, Dan Jurafsky, and James Zou. 2018. Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, 115(16):E3635–E3644.

Hila Gonen and Yoav Goldberg. 2019. Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 609–614, Minneapolis, Minnesota. Association for Computational Linguistics.

Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature*, 585:357–362.

Benjamin Heinzerling and Michael Strube. 2018. BPEmb: Tokenization-free pre-trained subword embeddings in 275 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Matthew Henderson, Iñigo Casanueva, Nikola Mrkvsi'c, Pei hao Su, Tsung-Hsien, and Ivan Vulic. 2019. Convert: Efficient and accurate conversational representations from transformers. *ArXiv*, abs/1911.03688.

Omer Levy and Yoav Goldberg. 2014. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 171–180, Ann Arbor, Michigan. Association for Computational Linguistics.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013a. Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013b. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

Piero Molino, Yang Wang, and Jiawei Zhang. 2019. Parallax: Visualizing and understanding the semantics of embedding spaces via algebraic formulae. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 165–180, Florence, Italy. Association for Computational Linguistics.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Andrew Trask, Phil Michalak, and John Liu. 2015. sense2vec - a fast and accurate method for word sense disambiguation in neural word embeddings. *ArXiv*, abs/1511.06388.

Jacob VanderPlas, Brian E. Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. 2018. Altair: Interactive statistical visualizations for python. *Journal of Open Source Software*, 3(32):1057.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.