Emotion and Affect Representation
in Sentence Embeddings

Author:
Luis Alberto
BARRADAS CHACÓN
278183

Supervisors:
Prof. Dr. Christian WARTENA
MSc. Rafael DRUMOND

19th May 2020

Thesis submitted for
MASTER OF SCIENCE IN DATA ANALYTICS

WIRTSCHAFTSINFORMATIK UND MASCHINELLES LERNEN
STIFTUNG UNIVERSITÄT HILDESHEIM
UNIVERSITATSPLÄTZ 1, 31141 HILDESHEIM

Statement as to the sole authorship of the thesis:

Emotion and Affect Representation in Sentence Embeddings.
I hereby certify that the master's thesis named above was solely written by me and that no assistance was used other than that cited. The passages in this thesis that were taken verbatim or with the same sense as that of other works have been identified in each individual case by the citation of the source or the origin, including the secondary sources used. This also applies for drawings, sketches, illustration as well as internet sources and other collections of electronic texts or data, etc. The submitted thesis has not been previously used for the fulfilment of a degree requirements and has not been published in English or any other language. I am aware of the fact that false declarations will be treated as fraud.

19th May 2020, Hildesheim

Abstract

Word embeddings exist to abstract features of text into a numeric space. In this project we explore the representation of different emotions and affects from established labeled datasets in common word embeddings.

Acknowledgements

Acknowledgements Here

Contents

iii

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Emotions and Affect

Within the context of this project it is important to distinguish between emotion and affect. Affect, in the context of this project will be treated as a term to associate predisposition towards stimuli. Thus, affect is in a sense, a general term that can be even used to describe animal, and other non-human entities. Emotions, on the other hand, are treated in this project as a state inherent to humans. This state is multidimensional, and every dimension, or emotion, can either be present in a certain amount, or not be present at all. Emotions present an affect value, but not necessarily otherwise.

Models of Emotions

When trying to detect emotion, it is relevant to know what emotions to look for. This is called a model of emotions, and is still a very discussed subject is psychology. Although there are several models of emotions the most persistent are Ekman's model, Plutchikc's model. The main assumption this work does follows the theory of constructed emotions, which recognizes affect as a physiological response to positive or negative stimuli, but emotions as a cognitive form of context-giving.

Why is it important to study emotion? Emotions are considered a human state that influences behaviour and decision making. Many times, when expressing thoughts in a written or spoken form, one or several emotions are present. Detecting these emotions is an important task for human interaction. Automatic emotion detection on text is thus a machine learning task required for comprehensive human-computer interaction.

Eckman's model of Emotions

What are emotions? What is affect? Why study emotions? The role of emotions in communication Measurements of emotions:

- Facial Expressions

- Biosignals

- Language or self report

Constructed theory of emotions Emotions in Language Semantic Fields Emotion Lexicon Emotion Networks Learning word semantics from context

1.2 Emotion and Machine Learning

Language in Machine Learning NLP workflow? Language representation Tokenization The problem of large dictionaries Dimensionality reduction/ autoencoders Deep learning for language: automatic feature extraction Machine Learning meaning from context Word Embeddings Transformers

1.3 Problem setting

1.4 Project Description

1.5 Objective

Objetivo general: Análizar la representación de emociones en modelos del lenguaje en machine learning. Esto de manera objetiva y medible, y subjectiva, pero disponible al público para fomentar discución sobre la representación de emociones en modelos de machine learning.

Objectivos secundarios Plantear una metodología para análisar un dataset de emociones, basado en modelos de lenguaje previamente entrenados.

Cómo consecuencia del objetivo anterior, a marco teoríco y práctico se puede crear para el análisis de eficiencia de modelos previamente entrenados en datasets de clasificación de textos cortos, acompañados de una sola etiqueta. Ya que el resultado no está vinculado a la semántica de la etiqueta: (emociones). Pero la representatividad adecuada de las etiquetas en el espacio abstracto propuesto por el modelo, resulta en la más fácil clasificación de las observaciones y sus respectivas etiquetas.

1.6 Justification

Word Embeddings Numerically representing words is commonly known as word embedding. This allows for Machine Learning (ML) models to easily manipulate text data that would otherwise be an arbitrary encoding of text. With the advances in machine learning automatic word embedding became a possible solution to avoid crowdsourcing. Several machine learning approaches try to automatically learn the best numeric representation for characters, words, or sentences given the context of a dataset. Recently there have been many efforts from research institutions to generalize these embeddings though the use of powerful models, and bigger datasets. Such is the case of BERT, a model created by Google with massive datasets.

Affect learning and its relevance The relevance of affect in text has increased since the popularization of text-based social networks, like twitter. There, individuals and organizations openly express their opinions. This creates an environment where implicit feedback about entities is present. An easy way to abstract popular opinion about a named entity is learning the affect expressed in text, such as a tweet. Affect can be a multidimensional phenomenon, but the most important dimension of it is valence: whether a text expresses positive or negative emotion.

Figure 1.1: Example of an image

Followed by equation 1.1

$$A = \{'a' : ['d',' e',' f',' g'],$$
$$'b' : ['d',' e',' f',' g'],$$
$$'c' : ['d',' e',' f',' g'],$$
$$'d' : ['h'],$$
$$'e' : ['h'],$$
$$'f' : ['h'],$$
$$'g' : ['h'],$$
$$'h' : []\} \tag{1.1}$$

Chapter 2

Related Work

2.1   Lexicons

Mohamad and Turney created an Emotion Lexicon through crowdsourcing [Mohammad and Turne
In this way an emotional word embedding was created by subjectively asking
participants whether or not a word was related to a specific emotion.

2.2   Automatic Approaches

Vo and Zhang created an automatic approach to learning sentiment lexicons
for short texts through the use of emojis [Vo and Zhang, 2016]. This method
uses the intrinsic usage of emojis to express positive or negative valence in
a sentence, and exploded this to expand that valence to words used in the
same context.

Maas et al.created a method to learn word vectors for sentiment ana-
lysis [Maas et al., 2011].

By applying machine learned automatic embeddings, the creation of word
embeddings based only on text data was open as a possibility. This is also a
method that later became the popular Word2Vec method [Mikolov et al., 2013].

A refining of word embeddings has been suggested by Yu et alby means
of a clustering algorithm on the vector space [Yu et al., 2017].

Rothe et alsuggested an orthogonal transformation to word embeddings
used on SemEval2015 which yielded ultradense word embeddings for affect
[Rothe et al., 2016].

A further exploration of transformations of a word vector space was done
by Hollis et alby means of component analysis, thus creating models of se-
mantics from text. These were applied to affect[Hollis and Westbury, 2016].

These studies have mostly been done with affect: positive and negative valences, but have mostly ignored other emotional dimensions.

2.3 Word Embeddings

2.4 Language Models

For word embeddings, different pre-trained models will be used:

- Fasttext

- Word2Vec

- GloVe

- BERT

A word embedding can also be trained from data. This is considered an optional model to analyze.

Selected Language Models

FastText

Word2Vec

GloVe

BERT

2.5 Analysis Algorithms

The following techniques have been used:

- TSNE

- PCA

2.6 Datasets

There is the possibility of using the datasets from the unified dataset of emotion in text. (An Analysis of Annotated Corpora forEmotion Classification in Text). This includes:

- AffectiveText

- Blogs

- CrowdFlower

- DailyDialogs

- Electoral-Tweets

- EmoBank

- EmoInt

- Emotion-Stimulus

- fb-valence-arousal

- Grounded-Emotions

- ISEAR

- Tales

- SSEC

- TEC

The link to these datasets can be found under the github repository for the unified emotion datasets. https://github.com/sarnthil/unify-emotion-datasets/tree/master/datasets

During the exploratory phase of this project, a subset of these datasets will be selected, giving priority to those with a more standardized, cleaner, or easier to access data. Models

2.7 Research Question

As a general research question we propose to answer the following: When using pre-trained models for word and sentence embedding, **is the information about the emotional and affective content or context of the word or sentence represented in the vector space?**

This question can be approached in three different ways:

- Is there a direct correlation between any of the dimensions of the vector space and human-labeled emotions and affect?

- Is there a linear transformation that will yield a direct correlation to the same human-labeled emotions?

- Is there a hierarchical structure that accurately represents the embedding of said labels?

Chapter 3

Methodology

To analyze the representation of emotions in different word embeddings, a high emphasis on dimensionality-reducing visualizations was done.

The steps to do so are the following:

1. First, the selected datasets must be embedded in their vector space. This will allow for a faster processing of the data, but requires making the technical decision of how the word vectors will be turned into sentence vectors to share the label in the case of datasets that contain a label for every sentence.

2. A supervised clustering must yield accuracy similar to that of a classification task. This means testing the accuracy of a classifier on the emotion recognition task. Even though this step could be seen as optional, failing to reproduce the baseline accuracy scores for this simple task might mean that the embeddings are not even capturing the basic information about the task.

3. A correlational analysis will be done between every dimension of the vector space and the emotions present in the dataset. This will tell us about any linear representation of emotions in the vector space.

4. A second study must show if a linear transformation of the vector space dimensions will yield a better correlation with the emotions of the datasets. This means either LDA, or PCA.

5. A hierarchical clustering will be used to analyze any possible structure in the embedded dataset in relation to emotions.

6. A final approach to this problem can be done through the study of oppositeness.

The mentioned methodology is based on answering the research question with progressive approximations. It is highly unlikely that a simple embedding model represents emotions in a single dimension in a linear manner, but it is increasingly more likely that some correlation is found with a linear transformation of the aforementioned. In case these two approaches present no information about emotions, a hierarchical clustering can extract the intrinsic information of affect in emotions. Since previous works have already shown that affect can be represented in vector spaces, created with a linear transformation of word embeddings[**?**], it would be contradictory to not find a hierarchical structure of emotions in this last step.

3.1 Preliminaries

This research was managed as both, a research project, and a software development project. As a setup for this project, two steps were taken. Setting up the physical and virtual environments for the experiments, and converging the different datasets and model embeddings into a single data source.

With scientific rigor, order, and reproducibility in mind, a git repository has been setup, where not only the working environment is provided, but also the history of the project development.

Environment Setup

Organizational

The project planning was layed out throught three months: March, April and May of 2020. A total of twelve weeks were divided into four equal sprints, where the four main tasks in the project were equaly separated in time: Exploration and Preparation, Programming, Experiments, and Writing. The four sprints were described by tasks, further divided by sub-tasks. These were kept in track and followed me, and both Supervisors through the Asana application[**?**].

The repository is accessible through github: `TODO`

Hardware

This project was implemented and executed in my personal computer: A Manjaro Linux x86˙64, with Kernel 5.6.11-1-MANJARO. The available CPU is an Intel i7-8700K (12) @ 5.000GHz, and the GPU is an NVIDIA GeForce GTX 1080 Ti. A total of 15937MB of RAM memory were available for experiments, as well as 16GB of swap disk. Although much of the technology

available for these experiments is more than necesary, the execution of some BERT models is not possible with these technical specifications. This influenced the selection of the BERT model to be used, and played a big part in selecting a pre-embedding of models.

Software

As mentioned, the development and execution were on a Linux Operating Systetm (OS). The Distribution used was Manjaro. This OS is a rolling release distribution, so the version used changed allong the development. This is one of the reasons why virtual development and execution environments were used: to keep reproductability, and ensure a stable testing. The only reason for this OS to be used is that it is my personal computer. As a Version Control System (VCS), git was added to the repository. This enables distributed access and historical revision for anyone trying to reproduce or supervise the project. Several development tools were used. For text editing and script execution, Atom 1.46.0 [**?**] was used. Within the Atom environment, community packages were used to simplify the workflow: Hydrogen 2.14.1 [**?**], for example, allows the execution of python code from within the text editor, and can even show output of the lines executed. A list of the used packages is provided in **??** For some exploratory analysis, Jupyter Notebooks [**?**] were used. To run these, a specific virtual environment was created with Docker 19.03 [**?**] and NVIDIA-Docker. A docker image for theses notebooks was created. The dockerfile of this image contains the libraries used for data exploration. The downloading of the BERT models ran in TensorFlow is also contained in this dockerfile. The description and an initialization script for the virtual container are included in the project folder called "TF". While the notebooks provided were used for data exploration, and visualization. Most of the development was done on the text editor. For this, python virtual environments were created with the help of the `virtualenv` and `virtualenvwrapper` python libraries. For these, a "requirements.txt" file was provided with the libraries used, and their versions. When developping, the desired virtualenvironment was activated. After this, the atom editor is open on the desired folder. By doing so, the Hydrogen library takes the virtual environment for the execution of the code in the project. By developing in this way, the whole project is available from the folder view on Atom. Code can be executed, and tested on the run, as if it were a Jupyter notebook, but changes are inmediately integrated into the code repository. This specific development environment was seleted to avoid conflicts between Jupyter Notebooks, and the VCS. The explorations are stored as notebooks, but cannot really represent the development of the

project. The Python programming language was used for the programming of the current project. This is due to it's incredible flexibility, access to the main ML libraries, and the predisposition of the Wirtschaftsinformatik und Maschinelles Lernen Institut. Under Python's umbrella of libraries, several were specifically added to enable this study. A List of the used libraries is provided in the appendix ??.

Two main ML frameworks were selected for the current project: Tensor-Flow 2.1.0 [?] (TF), and PyTorch 1.4.0 [?] (Also called Torch, for simplicity.). TF was selected specifically for it's access to a pre-trained BERT library [] for embedding sentences. This was very usefull, since, compared to the Transformers library [?], it must not be fine-tunned. TF confronts developers with two main compatibility issues:

- The cuda library being used most be a specific version. Most TF libraries will only work under CUDA library 9.2. Some might run under 10.1, but not under 10.2. Since the development environment is a rolling release linux distribution, the latest version of libraries is provided. Installing multiple versions brings problems to the day-to-day usage. Since the environment is also my personal computer, a virtual environment with containers were used instead, and for these, NVIDIA-Docker.

- at the moment of the development of this project, TF is undergoing a major version change, from 1.x to 2.x. Many reference libraries, and all code I have creted, used, or studied in my masters is depricated. The techniques learned during my studies need to be updated, and in many cases, re-learned. This is not an uncommon problem in technology, but it opens the opportunity for changing the work framework.

For all ML programming requirements that did not use the pre-trained BERT library, PyTorch was used. Certain algorithms were not programmed, but simply integrated from their implementation on python:

- FastText: This algorithm was not implemented. It's python library from the implementaiton of Facebook Research was used. [?]

- MulticoreTSNE: The TSNE algorithm was not implemented. Since it has heavy requirements on hardware, its implementation using distributed computing was used. [?]

- Normalize: The sklearn version of the normalization algorithm was used due to its optimization. [?]

- PCA: SKlearn version was used. [?]

- Tokenization: Part of the embedding pipeline requires the tokenization of the sentences. This was done with the Spacy library, and the "en˙core˙web˙sm" model.[**?**]

The Datasets

Access

http://www.crowdflower.com/wp-content/uploads/2016/07/text˙emotion.csv

Storage

The datasets were downloaded and stored under the project folder "data". Since every dataset is provided in different format and under different folder structures, every dataset is simply stored inside a folder with it's name. Under the datasets folder, every selected dataset is accompanied by folders with the embedding model used to embed the dataset. Thus every dataset folder has several subfolders. On these subfolders, a python script called "embed.py". This script varies for every model and dataset. In general terms, it extracts the text and label from the dataset, embeds the text into the desired model, and stores it in a "csv" file under the same folder. The "csv" file is stored under the name "embedded.py", except for the FastText model. In this case, there are two embedding approaches, one supervised and one unsupervised. Thus the names of the FastText embedding files are "embeddings˙supervised.csv", and "embeddings˙unsupervised.csv". Every other script creates a single "csv" file called "embeddings.csv".

Since every model embedds text in different

Thie file structure of the embedded files allows for exploration and experimental scripts to access the embedded data of different datasets, by building a single string with the dataset and model selected. This string must be prepended by the "./data/" folder name, and appended with the "embeddings.csv" string to generate a path that creates accesibility to the different datasets via a python coma-sepparated-value library, such as the built in `csv`, or Pandas [**?**] and it's `read_csv` function. This effectively create a data source to be used in a data pipeline. This approach was selected due to it's simplicity.

Embedding

The comparisson of the representation of different language models in this project requires a convergence of many different techniques. For this reason

it was chosen to embed the datasets into an intermediate format, to later use them in experiments.

The embedding of the datasets is comprised of 5 steps:

1. Loading model, text, and labels.

2. Tokenizing text.

3. Embedding every token into the model latent space.

4. Average the given embedded words.

5. Store the average sentence embedding.

Loading text, and labels was done with either the CSV[**?**] or the JSON[**?**] python library, depending on the format of the data.

Tokenizing was done with Spacy's "en˙core˙web˙sm" model, which allows access to the tokens via an iterator on the model, and the sub-compontent "text". An small snippet showing this process is shown in 3.1. This snippet considers a model has been loaded as a dictionary on tokens.

Listing 3.1: Tokenizing with Spacy

```python
import spacy
nlp = spacy.load("en_core_web_sm")
for token in nlp("This is a sentence in English"):
  word_embedding = model[token.text]
```

Every token is embedded in this way, but some models might not contain some tokens. In this case, the token is simply skipped. Some tokens with relevant information can be lost with using pretrained models that don't contain the complete vocabulary of the dataset, but it is expected, that the information distribution converge to the real distribution when large number of samples are integrated. This problem is later discussed in relation to training a model from zero **??**.

Once every token has been embedded into the model's latent space. A simple average is done across the tokens, keeping the dimensionality of the vector representation, and effectively creating a sentence embedding, represented in the model's latent space. This technique was selected since it's the most common [**?**] method for sentence representation. With this method, the secuential nature of the tokens in the sentence is lost, in favor of providing a constant sized sentence embedding to compare between methods and datasets.

Lastly, the sentence embeddings are stored along with the label information. For this, the CSV format was selected, due to it's interoperabiliy, and accessibility. The statistics library Pandas has an excellent csv reader, but the data can also be imported into spreadsheet software, other statistical software, or very quickl loaded on to python with the CSV library. Every CSV file contains a header on the first row. The header is composed by $N+1$ columns where N is the number of latent dimensions in the model. The last column is the "Emotion" column, where the label is stored. The name of every column starts with the letter "d", and is followed by consecutive numbers.

Since every pre-trained model is different, there were specific requirements on loading the model and embedding the tokens:

FastText

As previously mentioned, the FastText algorithm is an exception in this project, since it is NOT a pretrained model. The model is trained based on the dataset given. This can be done in a supervised, or an unsupervised manner. Due to the two methods for the usage of the FastText python library, the process of embedding a dataset with it requires two extra text files one with a sentence per line, and a second one, which includes the label as the last word of every line, prepended by two underscores (__).

In both ways of training, the language model is being trained specifically for the dataset vocabulary. For this reason, all tokens will be available in the model's vocabulary, resulting in the most complete language model. This is at the cost of representing only the topics on the dataset. This is therefore also not a general languge model.

Word2Vec

Word2Vec is trained in a very similar way as fasttext. Therefore, the expected results are similar. Word2Vec is treated within the context of this experiments as the pre-trained equivalent of fast text. The same number of latent dimensions, and a similar training approach were used. In this case, if a word in the dataset is not contained in the Word2Vec model, it is droped, and its analysis won't be included in the results of this project. Word2Vec was trained with a very large corpus, it is therefore considered a general language model.

The pre-trained model has been stored under the project folder "./models/Word2Vec/GoogleN vectors-negative300.bin.gz". The gensim python library is used to load the model in binary format without having to decompress it. This model is

loaded as a dictionary. An example of this is shown in snippet 3.2 that considers an iterator over a tokenized sentence.

Listing 3.2: Loading Word2Vec

```python
import gensim
model =
    gensim.models.KeyedVectors.load_word2vec_format(model_path,
    binary=True)
for token in tokenized_sentence:
  word_embedding = model[token]
```

GloVe

Pre-trained GloVe models can be downloaded from the official website `TODO`. This model was downloaded and stored under the project folder "./models/GloVe/glove.6B/glove.6B.300d.txt". The name of this file contains two numbers: 6B is the number of words that are represented in this model, while 300d is the number of latent dimensions used to represent the vocabulary. This model has been trained for 50, 100, 200, and 300 dimensions. Since a smaller number of dimensions represents a lesser capability for representing complex language concepts [?], the larger version of this model was selected. This also concides with the number of dimensions used in Word2Vec, which makes results easier to compare.

BERT

Although BERT is a pretrained model, it's original distribution is considered to be only partialy trained. On the original paper [?], a fine-tunning task-specific phase is mentioned, and generally required for the model to work best. This fine-tunning also presents a great infrastructure challenge, since some pre-trained BERT models simply wont fit into a personal computer's RAM.

For this reason, the pre-trained BERT embedding library `https://github.com/imgarylai/bert-embedding` was used. This library allows for a selection of the BERT model, and the embedding of the whole sentence, without tokenization. The result is a json-like dictionary in Python that contains both the original sentence and the embedded sentence.

To be able to run the embedding notebook, provided under the project folder "exploration/Embedding with bert.ipynb", the following requirements should be met:

16

- Docker ¿=19.03

- NVIDIA Container Toolkit

- This Docker TF Image: 'tensorflow/tensorflow:2.1.0-gpu-py3-jupyter'

On Linux, the a correct installation of the nvidia-docker environment would yield a successful run of the following command: `docker run --gpus all --rm nvidia/cuda nvidia-smi`

To build the docker image for this project, one must open a terminal on the "TF" project folder and run the following docker instruction: `docker build -t bert .` where bert is the name of the image to be created. Once this image has been built, docker can create containers with it. So to run the container necessary for the BERT embedding, the following command is used inside the project folder: `docker run --gpus all -p 8888:8888 -v $(pwd):/tf -it bert`. This last command will run a docker container, based on the "tensorflow:2.1.0-gpu-py3-jupyter" image, connect it to the localhost port 8888, and integrate the project folder to the jupyter server running on the container.

Docker is used to comply with the complex requirements of TensorFlow, CUDA, and the bert-embeddings.Once the Jupyter server is running, the notebook can be opened, and executed. The loading of the model is shown in the following snippet 3.3:

Listing 3.3: Loading BERT

```
from bert_embedding import BertEmbedding
bert_embedding = BertEmbedding(model='bert_24_1024_16',
    dataset_name='book_corpus_wiki_en_cased')
```

Here, the selected model is shown. This is a model with 1024 latent dimensions, trained on the Wikipedia corpus, and with case sensitivity. This means that words lowercase and uppercase letters will be embedded differently.

Within the notebook, a function was created to embed the datasets. This receives three arguments: a list of the sentences, a list of the labels, and the name of the output file, as a string. The embedding function is shown here:

Listing 3.4: Embedding with BERT

```
def embed_and_save(X, Y, outpath):
    E = np.array([np.mean(t[1], axis=0) for t in bert_embedding(X)])
    with open(outpath, 'w', newline='') as f:
        fieldnames = [f"d{i}" for i in range(len(E[0]))] +
            ['emotion']
```

```
writer = csv.DictWriter(f, fieldnames=fieldnames)
writer.writeheader()
for e, l in zip(E, Y):
    writer.writerow(dict({f"d{i}": ei for i, ei in
        enumerate(e)}, **{'emotion': l}))"
```

Running the embeddings for the datasets reported the following data:

Dataset	User	System	Total	Wall
CrowdFlower	user 3h 57min 7s	13min 11s	4h 10min 18s	1h 5min 33s
EmotionPush	user 1h 28min 35s	5min 25s	1h 34min 1s	24min 31s
Friends	user 1h 26min 40s	5min 3s	1h 31min 44s	24min 9s

Table 3.1: Runtimes for embedding datasets with BERT

This is much less than the "several days" reported by classmates. This might be due to the use of pre-trained models, and not running back-propagation to fine-tune the language models.

While running the embeddings, almost no GPU memory was used. This signals that the library is actually not making use of the GPU resources available. This also might mean that the embedding of the datasets might be much faster if the correct hardware resources are used.

At the beginning of the Year 2020, the library seemed a reliable way of getting the embedding done quickly. It allowed for embedding of the complete datasets in matter of minutes. Since I had been warned BERT embeddings could take days, I saw this as a gread advantage, and kept the method. Unfortunately as of May 2020, this library has been depricated. It's unmantained, and has requirements that might only be achievable under very specific conditions. This will not be a problem for reproduction, as long as the library is still available, and the provided docker image is used.

3.2   Analysis

Correlational Analysis

Linear Dimentionality Reduction

Non-Linear Dimentionality Reduction

Clustering Analysis

3.3   Experiments

Emolex

Selecting Emotions

Class Inbalance

Valence

Chapter 4

Experiments

4.1 Performance

4.2 Results

Chapter 5

Conclusion

Bibliography

[Hollis and Westbury, 2016] Hollis, G. and Westbury, C. (2016). The principals of meaning: Extracting semantic dimensions from co-occurrence models of semantics. *Psychonomic Bulletin & Review*, 23(6):1744–1756.

[Maas et al., 2011] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011). Learning Word Vectors for Sentiment Analysis. page 9.

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. page 9.

[Mohammad and Turney, 2013] Mohammad, S. M. and Turney, P. D. (2013). Crowdsourcing a Word-Emotion Association Lexicon. *arXiv:1308.6297 [cs]*. arXiv: 1308.6297.

[Rothe et al., 2016] Rothe, S., Ebert, S., and Schütze, H. (2016). Ultradense Word Embeddings by Orthogonal Transformation. *arXiv:1602.07572 [cs]*. arXiv: 1602.07572.

[Vo and Zhang, 2016] Vo, D. T. and Zhang, Y. (2016). Don't Count, Predict! An Automatic Approach to Learning Sentiment Lexicons for Short Text. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 219–224, Berlin, Germany. Association for Computational Linguistics.

[Yu et al., 2017] Yu, L.-C., Wang, J., Lai, K. R., and Zhang, X. (2017). Refining Word Embeddings for Sentiment Analysis. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 534–539, Copenhagen, Denmark. Association for Computational Linguistics.