Identifying Gene-Disease Relationships from Journal Abstracts

Kayla Johnson and Anna Yannakopoulos

Introduction

The biomedical research community generates an ever-increasing amount of data each year, continuously adding to the already vast repository of freely accessible, peer-reviewed research publications. These publications contain a wealth of valuable patterns and knowledge, but because this information is hidden in unstructured natural language, it is difficult to computationally extract meaningful data from them. As a result, many have leveraged various natural language processing (NLP) frameworks to mine biologically-relevant relationships from these texts [3]. These relationships are generally lower-quality than those determined by expert curation, but may have value as input to weighted machine learning algorithms.

In this research, we have endeavored to determine whether using Named Entity Recognition (NER) to link biomedical synonyms before using an algorithm to generate word embeddings can improve the ability to find gene-disease relationships. Furthermore, previous research has shown that using non-biological sources of text in addition to biomedical text for input can improve biological relationship findings within other NLP frameworks, perhaps because a more general linguistic understanding can be encoded in word embeddings with a larger, more general corpus [5]. In order to test this observation, we compared the use of a corpus composed of both PubMed abstracts and English Wikipedia articles to the use of a corpus consisting of PubMed abstracts alone.

Background

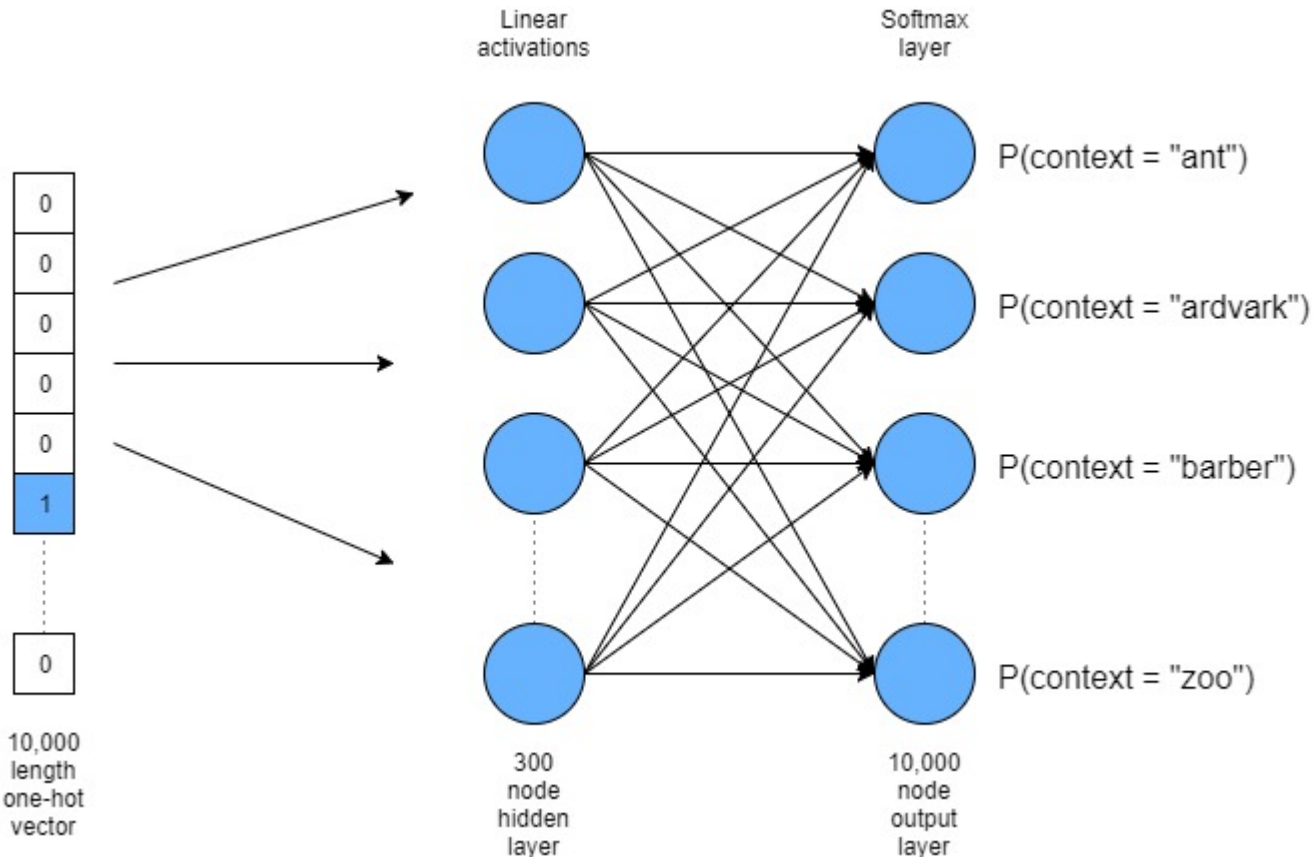
Two conceptual challenges which have proved problematic for NLP approaches are addressed by this research. First, words of interest must be distinguished from the rest of the text using NER. For our application, we were interested in biomedical word categories such as genes, proteins, mutations, diseases, etc. One difficulty associated with this step is that in a biological context, there are many terms which are synonymous. Furthermore, in many cases phrases are more than the sum of their parts; the phrase as a whole is much more significant than the individual words that make it up. Second, computationally determining the relationships between words can be difficult. There are a variety of approaches to this problem, but none is reliable for all corpuses and applications.

To overcome the first challenge, we employed an NER tool called EXTRACT, which is capable of identifying genes/proteins, chemical compounds, organisms, environments, tissues, diseases, and phenotypes and providing a unique identifier from each term's governing ontology [1]. For example, if EXTRACT encounters the strings “myocardial infarction”' and “heart attack,” it will link both to the corresponding ID from the Disease Ontology or Gene Ontology [4, 6]. This has allowed our NLP method to combine the contextual information provided by both of these terms into a single word, improving our contextual understanding of the disease as a whole.

To overcome the second challenge, we used the word2vec algorithm to embeds each word in our corpus into a high-dimensional vector space in which each word vector's geometric location relative to other word vectors implies something about its meaning. Similar words are expected to occupy similar regions in this vector space, so closeness can be used as a measure of relationship strength.

Word2vec trains one shallow neural network per distinct word in the corpus, where the input is a one-hot vector representing the word and the output is the word’s normalized context vector, each with dimensionality equal to the number of distinct words. Between the input and the output, there is a hidden layer with dimensionality equal

to that of the word embedding vector space. The structure of the network for a corpus with vocabulary size 10000 is described in the figure below [12]. In each iteration of the word2vec training, the weights of the hidden layer are adjusted so that the output more closely resembles the word's context vector as determined from the corpus. When the training is complete, the word embedding in the vector space is defined by the weights of the hidden layer.



Methods

Building Corpuses

The first corpus consists of all abstracts in the PubMed citation database, which includes over 28 million biomedical texts from 1966 to the present, as of 2018. A dump of the PubMed database was downloaded from the U.S. National Library of Medicine's FTP server as a collection of 928 gzipped XML files, each containing tens of thousands of citations [7]. We chose to construct our corpus from only those citations that have abstracts, with one abstract per paragraph.

The second corpus consists of the PubMed abstract corpus and all 5.5 million published English Wikipedia articles, containing about three billion words. A dump of the current revision of English Wikipedia was downloaded as a compressed XML file from Wikimedia. The raw text is formatted in Wiki Markdown, so wiki2text [9] was used to clean it. We then segmented the Wikipedia dump into several hundred separate files so that our computations could be more easily parallelized.

Preprocessing

Although there are only two different corpuses, there are 4 different test cases; in order to test the efficacy of using NER before creating word embeddings, we consider the PubMed corpus, the PubMed + Wikipedia corpus, the PubMed corpus with EXTRACT applied, and the PubMed + Wikipedia corpus with EXTRACT applied. In parallel,

each of our clean corpuses was sent to the EXTRACT API in chunks of less than 10 MB to detect biomedical terms. These biomedical terms were then replaced with their corresponding ontology IDs to create our EXTRACTed corpuses.

All four corpuses were then preprocessed, removing capitalization and words of three or fewer characters. Additionally, English stop words, such as “and” and “the” as defined in the NTLK Python package [10] were removed. This preprocessing reduces the number of words that our models need to be trained on, reducing the time taken to build each model. At the end of this step, we have four different corpuses on which to train word2vec.

Generating Word Embeddings

We used the Gensim [8] Python implementation of the word2vec algorithm to generate word embeddings from each of our corpuses. We chose to use the skip-gram model rather than the continuous bag of words model and set the dimensionality of our word embedding space to 300. We passed each paragraph in the corpus to word2vec as a “sentence,” ensuring that each abstract was considered as a whole. All other parameters were left as the defaults. The models took between 15 and 40 hours using 28 workers per model, depending on the corpus.

After obtaining word embeddings from word2vec, the PubMed and PubMed + Wikipedia models and vocabularies that had not previously been run through EXTRACT were converted to ontology IDs where applicable so that the genes and diseases in the model were easily identifiable. This is distinct from the case where EXTRACT was run before the models were trained, as the context of each word was changed by running EXTRACT pre-training, resulting in different word embeddings.

Validation

In our evaluation, only protein-coding genes which appeared in all four models were considered, leaving us with a total of 10181 genes that we could query. The protein-encoding genes are based on those contained in the Search Tool for the Retrieval of Interacting Genes/Proteins (STRING) [2]. For each disease that we studied, we generated a list of similarities between that disease and all genes in our models. This similarity is defined as the cosine similarity between the respective word vectors for the disease and each gene. We expect diseases and genes that are closely related to be located in roughly the same area of the word embedding vector space and hence have a high cosine similarity.

In [29]:
```
import gensim as gs
model_path = '/mnt/research/compbio/krishnanlab/projects/nlp/pubmed_clean.
txt'
model = gs.models.KeyedVectors.load_word2vec_format(model_path)
```

In [31]:
```
model.most_similar('brain')
```

Out[31]:
```
[('cerebrum', 0.7514523863792419),
 ('cerebellum', 0.7017686367034912),
 ('brains', 0.6887757778167725),
 ('hippocampus', 0.6431123614311218),
 ('brainstem', 0.641647219657898),
 ('subcortical', 0.63283371925354),
 ('cerebral', 0.6313273310661316),
 ('nonbrain', 0.6276805996894836),
 ('whole-brain', 0.626050591468811),
 ('midbrain', 0.6213588714599609)]
```

For each disease we investigated, we downloaded a list of associated genes from the GWAS catalog [11] to serve as our positive labels. This resource is a collection of published GWAS studies which have related genes to a particular disease or condition. Our negative labels are also derived from the GWAS catalog in combination with the Disease and Gene Ontologies. Using a pared-down version of each ontology consisting only of broad GO slim terms, we determined which disease and which genes were annotated to each GO slim term. If a gene did not share any annotations with the disease, it was considered a negative label.

Results

Below is code to generate results for genes linked to Alzheimer's disease based on the process described in Validation.

In [1]:
```
import matplotlib.pyplot as plt
from sklearn import metrics
import pandas as pd
import re, os, sys

# import utility functions from elsewhere in our project
sys.path.append('../src/')
from evaluate import get_pattern_vocab
from EXTRACT_corpus import EXTRACT_list
from EXTRACT_model import get_vocab
```

First, we generate our list of genes that can be queried in all models.

In [2]:
```
# define location of models and vocabularies
project_dir = '/mnt/research/compbio/krishnanlab/projects/nlp'
vocab_filenames = ['pubmed_EXTRACT_fixed_vocab.txt',
                   'pubmed_wikipedia_EXTRACT_fixed_vocab.txt',
                   'pubmed_clean_then_EXTRACT_vocab.txt',
                   'pubmed_wikipedia_clean_then_EXTRACT_vocab.txt']
vocab_files = [os.path.join(project_dir, filename) for filename in vocab_f
ilenames]

vocabs = [get_vocab(vocab_file) for vocab_file in vocab_files]

# search for words in the vocabulary that look like IDs for protein-coding
 genes (ex. ENSP00000369460)
gene_pattern = re.compile('ensp\d+$')
genes = get_pattern_vocab(vocabs, gene_pattern)  # contains the intersecti
on of matching genes for all vocabs

print('{} queriable genes found.'.format(len(genes)))
```

```
Reading vocab file /mnt/research/compbio/krishnanlab/projects/nlp/pubmed_EX
TRACT_fixed_vocab.txt...
Reading vocab file /mnt/research/compbio/krishnanlab/projects/nlp/pubmed_wi
kipedia_EXTRACT_fixed_vocab.txt...
Reading vocab file /mnt/research/compbio/krishnanlab/projects/nlp/pubmed_cl
ean_then_EXTRACT_vocab.txt...
Reading vocab file /mnt/research/compbio/krishnanlab/projects/nlp/pubmed_wi
kipedia_clean_then_EXTRACT_vocab.txt...
8079 queriable genes found.
```

Next, we read in data on Alzheimer's downloaded from the GWAS database and form a list of non-duplicate associated genes.

In [3]:
```
GWAS_file = '../data/gwas-association-downloaded_2018-04-21-Alzheimers dis
ease.tsv'
GWAS_data = pd.read_csv(GWAS_file, sep='\t', header=0)

positives = GWAS_data['REPORTED GENE(S)'].values  # use the genes reported
 by the authors
positives = [item.split(', ') for item in positives]  # some studies repor
t multiple genes
positives = [item for sublist in positives for item in sublist]

# run gene names through EXTRACT and only keep the ones we can query in al
l models
positives = [item.lower() for item in EXTRACT_list(positives, per_request=
10000)]
positives = [item for item in positives if gene_pattern.match(item) and it
em in genes]
positives = list(set(positives))  # many studies report duplicate genes

print('{} positive genes found.'.format(len(positives)))
```

```
Calling EXTRACT on 807 items...
155 positive genes found.
```

We then read the list of negative genes generated as described in Methods.

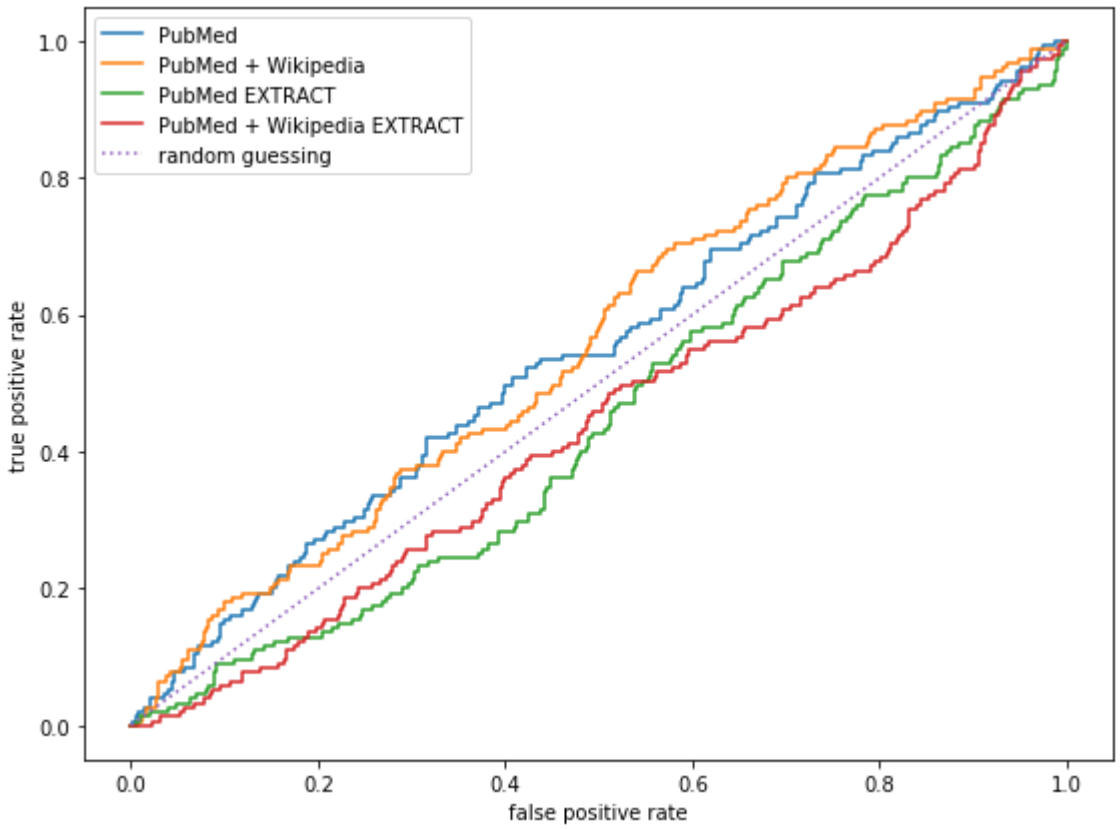
In [4]:
```
negatives_file = '../data/non-neurodegenerative-disease-genes-symbols.txt'
negatives = list(pd.read_csv(negatives_file, header=None)[0])

# apply the same EXTRACT processing as for the positive genes
negatives = [item.lower() for item in EXTRACT_list(negatives, per_request=
10000)]
negatives = [item for item in negatives if gene_pattern.match(item)
             and item in genes and item not in positives]
negatives = list(set(negatives))

print('{} negative genes found.'.format(len(negatives)))
```

```
Calling EXTRACT on 1252 items...
847 negative genes found.
```

We can generate our results from the scored gene rankings we produced earlier.

In [7]:
```
results_names = ['PubMed', 'PubMed + Wikipedia', 'PubMed EXTRACT', 'PubMed
 + Wikipedia EXTRACT']
results_filenames = ['ALZ_results_pubmed_clean_then_EXTRACT.tsv',
                     'ALZ_results_pubmed_wikipedia_clean_then_EXTRACT.tsv'
,
                     'ALZ_results_pubmed_EXTRACT_fixed.tsv',
                     'ALZ_results_pubmed_wikipedia_EXTRACT_fixed.tsv']
results_files = [os.path.join(project_dir, filename) for filename in resul
ts_filenames]

# define list of positive/negative labels for genes
labels = [1] * len(positives) + [-1] * len(negatives)

# some pretty printing
line_length = max([len(name) for name in results_names]) + 8
print('Corpus' + ' ' * (line_length - 11) + 'auROC')
print('-' * line_length)

plt.figure(figsize=(8, 6))

for name, file in zip(results_names, results_files):
    results = pd.read_csv(file, sep='\t', header=None, names=['gene', 'sco
re'])

    # get scores from genes known to be positive and negative
    results_pos = results.loc[results['gene'].isin(positives)]
    results_neg = results.loc[results['gene'].isin(negatives)]

    scores = list(results_pos['score'].values) + list(results_neg['score']
.values)

    # generate ROC curve
    fpr, tpr, _ = metrics.roc_curve(labels, scores)

    # print auROC per model
    spacing = ' ' * (line_length - len(name) - 7)
    print('{}:{}{:.4f}'.format(name, spacing, metrics.auc(fpr, tpr)))

    plt.plot(fpr, tpr, label=name)

# in an ROC plot, a classifier with random guessing will produce a straigh
t line with slope 1
plt.plot(fpr, fpr, label='random guessing', linestyle=':')

plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.legend()
plt.tight_layout()
plt.show()
```

```
Corpus                          auROC
---------------------------------------
PubMed:                        0.5477
PubMed + Wikipedia:            0.5589
PubMed EXTRACT:                0.4473
PubMed + Wikipedia EXTRACT:    0.4415
```



These results indicate that, in ranking genes based on their association to Alzheimer's disease, adding Wikipedia to the corpus had little effect, and running the corpus through EXTRACT was actively harmful to the model's ability to rank genes by similarity score.

Discussion

If we were to start this project over from the beginning, we would have approached some of our preprocessing differently. For example, when constructing the PubMed corpus, we might have included the titles of each citation as the first sentence in the abstract, given that the title often serves as an extremely compact summary of the paper's results. Additionally, given that we are attempting to ask a very specific question ("Which genes are related to which diseases?"), we are interested in what the performance of a very specific model might look like, such as one trained on a corpus that only consists of EXTRACT terms. In this case, we might increase the window size to include the entire abstract at once, rather than the default of only the 5 words immediately before and after each word.

We might have chosen better and more extensive options available for preprocessing rather than just removing words that don't fit our criteria. For example, we might have implemented lemmatization to reduce alternate forms of words, such as "running" and "ran," down to their root form, in this case "run," which may have improved our word embeddings. A potentially useful form of preprocessing that we did not implement could include some biological information by incorporating the location of the EXTRACT-identified terms within their respective ontologies.

We are still unsure if the cosine similarity between two words is a good measure of the strength of their relationship. Essentially, the cosine similarity measures the similarity between the typical contexts of the two words. For example, in models trained on English text, the words "nighttime" and "daytime" often have an extremely high cosine similarity because they have very similar usages as two words describing times, even though they are technically opposites. One of the strengths of word2vec is its ability to construct analogies; using typical English word embeddings, the expression "king" - "man" + "woman" evaluates to a point in vector space that is very close to the embedding of "queen." It is possible that there is a better, more biologically-relevant analogy we might use to measure the relationship between a gene and a disease. Furthermore, perhaps a different kind of natural language model might better capture the relationship information we are interested in.

In this report, we only tested our word embeddings on a single disease due to time constraints. We would like to test our models on all diseases in the GWAS catalog with some minimum number of associated genes to more fully evaluate their performance on a broader set of genes and diseases. Additionally, we could test disease-disease relationships by creating a similarity matrix between all diseases in the Disease Ontology. We could evaluate such a matrix by using it to reconstruct a tree structure with hierarchical clustering and then comparing it to the canonical Disease Ontology.

Bibliography

1. Evangelos Pafilis, Rudolfs Berzins, and Lars Juhl Jensen. "EXTRACT 2.0: text-mining-assisted interactive annotation of biomedical named entities and ontology terms." In: bioRxiv (Feb. 2017), p. 111088. DOI: 10.1101/111088. URL: https://www.biorxiv.org/content/early/2017/02/23/111088 (https://www.biorxiv.org/content/early/2017/02/23/111088).
2. Jensen et al. Nucleic Acids Res (2009), 37.Database issue pp. D412-6. URL: https://string-db.org/ (https://string-db.org/)
3. Kyubum Lee et al. "Deep learning of mutation-gene-drug relations from the literature." In: BMC Bioinformatics 19.1 (Dec. 2018), p. 21. ISSN: 1471-2105. DOI: 10.1186/s12859-018-2029-1. URL: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-018-2029-1

(https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-018-2029-1).
4. Lynn Marie Schriml et al. "Disease Ontology: a backbone for disease semantic integration." In: Nucleic acids research 40.Database issue (Jan. 2012), pp. 940-6. ISSN: 1362-4962. DOI: 10.1093/nar/gkr972. URL: http://www.ncbi.nlm.nih.gov/pubmed/22080554 (http://www.ncbi.nlm.nih.gov/pubmed/22080554).
5. Maryam Habibi et al. "Deep learning with word embeddings improves biomedical named entity recognition." In: Bioinformatics 33.14 (July 2017), pp. D985-D994. ISSN: 1362-4962. DOI: 10.1093/bioinformatics/btx228. URL: http://www.ncbi.nlm.nih.gov/pubmed/28881963 (http://www.ncbi.nlm.nih.gov/pubmed/28881963).
6. Michael Ashburner et al. "Gene Ontology: tool for the unification of biology." In: Nature Genetics 25.1 (May 2000), pp. 25-29. ISSN: 1061-4036. DOI: 10.1038/75556. URL: https://www.nature.com/articles/ng0500_25 (https://www.nature.com/articles/ng0500_25).
7. National Center for Biotechnology Information. Entrez Programming Utilities Help Manual. Bethesda, MD: U.S. National Library of Medicine, 2010. URL: https://ncbi.nlm.nih.gov/books/NBK25501/ (https://ncbi.nlm.nih.gov/books/NBK25501/).
8. Radim Rehurek and Petr Sojka. "Software Framework for Topic Modelling with Large Corpora." In: Proceedings of the LREC w2010 Workshop on New challenges for NLP Frameworks. http://is.muni.cz/publication/884893/en (http://is.muni.cz/publication/884893/en). Valletta, Malta: ELRA, May 2010, pp. 45-50.
9. Rob Speer. “Wiki2text.” In: GitHub Repository (2015). URL: https://github.com/rspeer/wiki2text (https://github.com/rspeer/wiki2text)
10. Steven Bird, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O’Reilly Media Inc.
11. T Burdett et al. The NHGRI-EBI Catalog of published genome-wide association studies. URL: https://www.ebi.ac.uk/gwas (https://www.ebi.ac.uk/gwas).
12. Andy Thomas. "Word2Vec word embedding tutorial in Python and TensorFlow." In: Adventures in Machine Learning (2017). URL: http://adventuresinmachinelearning.com/word2vec-tutorial-tensorflow/ (http://adventuresinmachinelearning.com/word2vec-tutorial-tensorflow/)