University of Sussex

My Theory of Everything

How it all works

Joe Bloggs

Submitted for the degree of Doctor of Philosophy

University of Sussex

September 2008

Declaration

I hereby declare that this thesis has not been and will not be submitted in whole or in part to another University for the award of any other degree.

Signature:

Joe Bloggs

UNIVERSITY OF SUSSEX

Joe Bloggs, Doctor of Philosophy

My Theory of Everything
How it all works

Summary

Acknowledgements

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Word embeddings is a method of word representation mainly used for natural language processing. Text data appears differently in computer from images and sounds. Both images and sounds can be easily represented as mathematic models either using analog or digital signals but not with text data (Li and Yang, 2017). Text data consists of strings that can only be modeled using one-hot vector (from given d-dimension for d words that are known, only one dimension is one and the rest are zeroes). This one-hot vector does not have any information that infers connection between one to the other. Hence that, vector representations that maps semantic and syntactic information given one-hot vectors in a euclidean space is introduced (Li and Yang, 2017). This positional information then can be used to infer interconnection between words, whether its similarities or usage of the word in a sentence (S. Harris, 1954). To obtain word embeddings, a model is created to extract features of a word from a corpus and map its location in euclidean space based on the features found. These word embeddings then can be used to do many downstream tasks, such as postagging, named entity recognition (NER), and sentiment analysis (Ling et al., 2015; Lample et al., 2016). In general, large corpus with many words and examples of word usage is prefered because the size of the vocabularies will increases and more words connection can be inferred from the corpus (Kutuzov and Kunilovskaya, 2018). However, it is not posible to have such corpus since the language itself is changing overtime (Forrester, 2008). Furthermore, there are cases of typographical error, especially after internet and social media are booming where anybody willingly write text over these platforms and these words maybe not present in the corpus hence not included in the vocabulary thus making many typographical error while in its correct form it actually is. In addition with the increased number of smartphone which uses touch screen, some typographical error is expected Ghosh and Kristensson (2017). All this words that are non-existent in the corpus

and its embedding cannot be inferred are called as *out-of-vocabulary* (OOV) words. One may use simple approach by asigning unique random embedding or by replacing OOV with an unknown $<UNK>$ embedding. While in some cases using these simple approaches and continue on the training on the downstream tasks can produce acceptable results {CITE}, further improvement on downstream tasks can be achieved by using machine learning method to infer OOV embeddings.

To infer OOV embeddings, the model is built over quasi-generative perspective. Only knowing the vocabularies and its embedding, the model tried to generate embedding for OOV words. Previous *state-of-the-art* used lstm to infer OOV embeddings (Pinter et al., 2017). Character embedding is used to transform sequence of characters then forwarded into bi-lstm then to fully connected layer. In language model, bi-lstm generally works by separating sub-word by remembering and forgeting previous sequence from both end. Those sub-word then will be used to infer its embedding. The problem might arise when there are more than two important sub-words and they are not in sequence, meaning that there exist at least one character between two important sub-words, hence the information is incomplete since lstm will dampen the previous sequence if the next sequence sub-words are considered more important. Instead of taking the whole words as a sequence and considering its importance based on time, by using n-grams and picking which grams that are considered to be important in theory should gives better results since the information is complete and only left for the model to pick which n-grams are more important. The only problem is that for the model to pick which grams that should be included in the model is impossible to do since there are many word combinations making the model needs to accept huge number of inputs. Instead of handpicking the features of n-grams, convolutional neural network (CNN) can be used to pick which features needs to be considered by using character embedding and treat the character sequence embedding as an image.

Chapter 2

Literature Review

Word2vec is one of word embedding that is trained using skip-gram model (Mikolov et al., 2013). A word $w(t)$ used as an input and its context word, for example context word with windows of 4 are $w(t-2), w(t-1), w(t+1)$, and $w(t+2)$, used as the target and the projection from input to the output is used as the representation of the input $w(t)$ that is usefull to predict the context words. This model is highly dependant on the corpus completeness. More examples and vocabulary a corpus has the better the representation of the embeddings since more information will be able to be learned. Word2vec model has no oov handling, meaning either random vector or unknown $<UNK>$ embedding will be used.

Polyglot embedding .....

Dict2vec is yet another embedding that is trained by looking up definitions of words from cambridge dictionary Tissier et al. (2017a). This embedding was created because the previous method is trained with unsupervised manner, meaning that there is no supervision between pairs of words. There might exists pair of words that are actually related but do not appear enough inside a corpus making it harder for the model to find connection. Thus, this model is trained by creating sets of strong and weak pairs of words, then move both pairs closer and further respectively based on the pairs. The model then evaluated using several word similarity tasks to show imporvements over vanilla implementation of word2vec and fasttext.

Part-of-Speech-Tagging (postagging) is a process of determining grammatical category (tag) of given word in a certain sentence. In English, exist words that has ambiguous grammatical category, such as word "tag" can be either noun or verb depends on the usage of it (Cutting et al., 1992). To tackle this problems, many researchers proposed to use mathematical models or statistical models namely hidden markov model (Cutting

et al., 1992), n-grams (Brants, 2000), and neural network model (Ling et al., 2015). In this research, the neural network model will be implemented to serve as the downstream task. This model is a recurrent neural network (RNN) that took sequence of word embeddings representing a sentence or parts of sentence then categorize each word embedding for its tag.

As aforementioned above, some word embedding model such as Word2vec has no oov handling, thus creating a model to predict such word becomes research interest. One of the model that tries to tackle this problem successfully used bi-LSTM to predict embedding given sequence of characters from a word from a pretrained embedding (Pinter et al., 2017). As a result, oov embeddings are able to be predicted without the needs of knowing lexicon or model used for creating the word embedding. The results then tested to do downstream task namely postagging.

N-grams often used to capture word features. N-grams relies on characters that make up a word, later on a sentence. With character embedding and convolution neural network (CNN), n-grams can be calculated by convoluting sets of characters embedding with the kernel size of $n \times d$ for n is the number of grams and d is the dimension of the embeddings. This CNN n-grams then can be used to create a neural language model (Kim et al., 2015).

Chapter 3

Method

3.1 Out-of-Vocabulary Model

3.1.1 Sequence Feature Extraction

OOV problem is handled from quasi-generative perspective as aforementioned in chapter 1 by using neural language model under assumption that there is a form that could generate embedding for the original embedding. Hence that, the original embedding is used for training the model to generate the embedding. In chapter 1, reasons why lstm could perform worse is because the hidden states is controlled by cell gates making the information that is carried on is the most recent after the cell gates decided to forget inputs at certain time t.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.1}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.2}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.3}$$

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{3.4}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.5}$$

$$h_t = o_t \times tanh(C_t) \tag{3.6}$$

Formally, when $C_t = 0$ from equation 3.4, hidden state from equation 3.6 will be reset to 0 rendering hidden states prior to time t gone. This problem can be solved by using bi-lstm, since bi-lstm process sequence in forward and reverse order making both early and later sequence held by the last hidden state for each forward lstm and reverse lstm respectively. Another problem might arise when we need to divide sequence into more

$un|recogniz|able$

$inter|national|ities$

$oto|rhino|laryngolog|ical$

$hepatico|chol|angio|gastro|stomy$

Figure 3.1: Word examples with three or more subsequences

$unrecognizable$:$unre|nrec|reco|ecog|cogn|ogni|gniz|niza|izab|zabl|able$

$internationalities$:$inte|nter|tern|erna|rnat|nati|atio|tion|iona|onal|$
$nali|alit|liti|itie|ties$

$otorhinolaryngological$:$otor|torh|orhi|rhin|hino|inol|nola|olar|lary|$
$aryn|ryng|yngo|ngol|golo|olog|logi|ogic|gica|ical$

$hepaticocholangiogastrostomy$:$hepa|epat|pati|atic|tico|icoc|coch|ocho|chol|hola|olan|$
$lang|angi|ngio|giog|ioga|ogas|gast|astr|stro|$
$tros|rost|osto|stom|tomy$

Figure 3.2: 4-grams examples

than three subsequence as shown on figure 3.1. Hence another approach is needed since intermediate subsequence might get deleted or carried along with the later sequence even with bi-lstm.

We can find many examples especially from more technical areas as shown on figure 3.1. Since on MIMICK Pinter et al. (2017) implementation only the last hidden state is used for inferencing word embedding, another approach is proposed.

For all subsequence to be processed, we need a method that accounts for all sequence yet still able to divides the whole sequence into subsequences. Consequently, n-grams is chosen because this method splits word into sequence of characters depends on the chosen window size as shown on figure 3.2. Those sequences of characters then feed into learning algortihm. This idea is similar to how human tries to recognize an unseen word by reading subword that is understandable beforhand when no explanation or context were given.

This model then will be used to estimate OOV embeddings. In other words, given sets of vocabulary $\mathcal{V}$ with size $|\mathcal{V}|$ and pretrained embeddings $\mathcal{W}^{|\mathcal{V}|\times d}$ for each word $w_i \in \mathcal{V}$

that is represented as a vector e_i with d dimension, the model is trained to map function $f : \mathcal{V} \rightarrow \mathbb{R}^d$ that minimizes $|f(w_i) - e_i|^2$. This approach is similar to MIMICK Pinter et al. (2017) approach. The text input is represented as a sequence of character $[c_1, c_2, \ldots, c_m]$ for $c_i \in \mathcal{C}$. Those sequence then transformed as sequence of vectors g_i with b dimension by using character embeddings $\mathcal{G}^{|\mathcal{C}| \times b}$. For simplicity, sequence of $[g_1, g_2, \ldots, g_m]$ will be called $[\hat{g}]$. $[\hat{g}]$ becomes 2-dimensional matrix that has size of $m \times b$. In summary, given word w will be transformed using function h into $[\hat{g}]$ as shown on equation 3.7.

$$h : w \rightarrow [\hat{g}] \tag{3.7}$$

To process $[\hat{g}]$ like an n-grams, CNN is used. CNN is basically a method to do convolution on matrix by using a kernel $k_i^{n \times b} \in K$. This operation is represented with $*$ symbol. To mimick n-grams, kernel with size of $n \times b$ is used, producing another vector $\hat{l}$ that represents the value of each grams, then non-linearity is applied to this vector by using ReLU activation function as shown in equation 3.8.

$$f(x) = ReLU(x) = max(0, x) \tag{3.8}$$

Several kernel is used to learn several features for producing embeddings. Each of these kernel will be responsible to find grams that are affecting the results, thus the vector $\hat{l_i}$ that are results of convolution $[\hat{g}] * k_i$ will be maxpooled to produce one number. In details, from given sequence of character embedding $[\hat{g}]$, only gram that produces the highest value when convoluted by using kernel k_i will be processed. Since, there are $|K|$ number of filter, $|K|$ number of grams will be considered to be important to the results. Furhtermore, by using several window sizes for n-grams (bigram, trigram, etc.), more features will be able to be learned.

3.1.2 Embedding Generation

After the features are able to be extracted, those features then concatenated and fed into fully connected layer with output size matching the pretrained embedding $\mathcal{W}$ dimension d with non-linear activation function matching the maximum and minimum bound of the pretrained embedding $\mathcal{W}$ resulting a new embedding vector $\tilde{e}$.

The complete process from input word, feature extraction, until predicting embedding is shown on figure 3.3.

On figure 3.3, starting and ending token is added at the beginning and the end of the word respectfuly. Furthermore, padding token is added if the input word is shorter

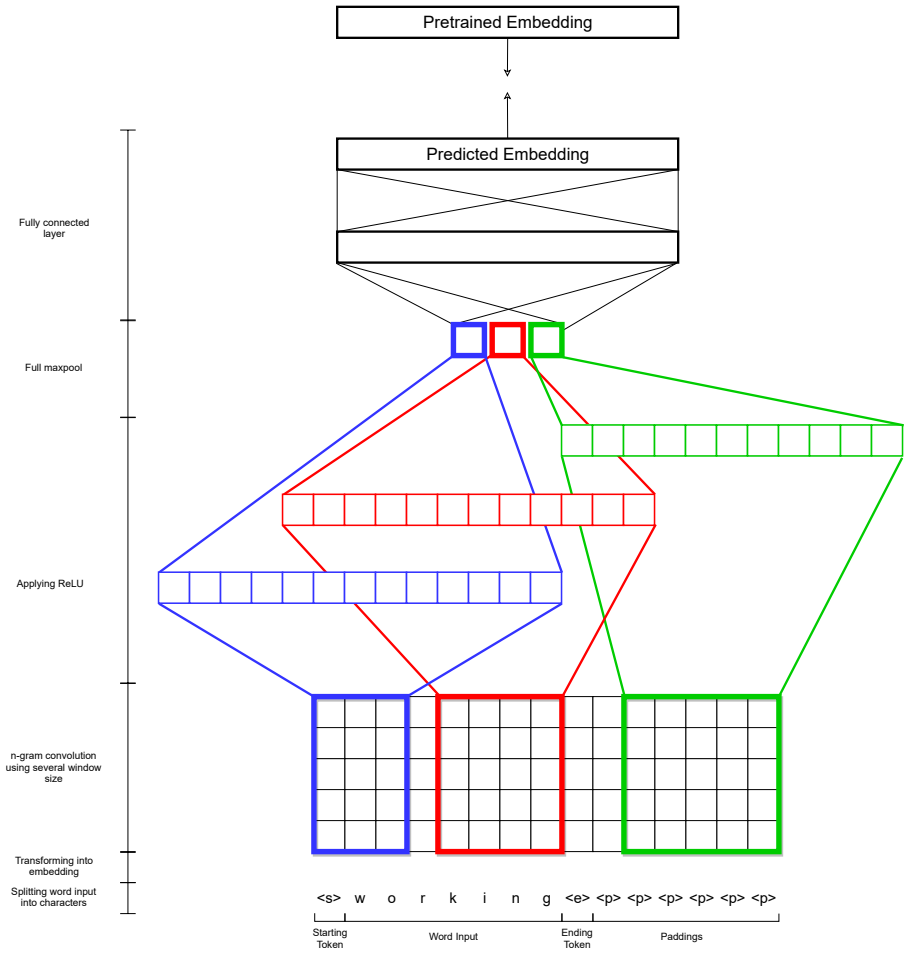

Figure 3.3: OOV Inferencing Model

than the length of the input size of the model. The padding token is a zero vector $\vec{0}$ and $\vec{0} * k = \vec{0}$ for any k. This is to ensure that part of input that got padded does not goes through maxpool layer since only grams that has highest value can goes through the next layer and minimum value of ReLU is 0.

3.1.3 Error and Backpropagation

The output of the neural network $\tilde{e}$ then compared with the original embedding e to learn new parameters for the neural network using mean squard error function as shown on 3.9

$$Error = \frac{1}{2}\|e - \tilde{e}\|^2 \tag{3.9}$$

The error then backpropagated to fine-tune the neural network parameters.

3.2 Measuring Performance on Donwstream Tasks

In natural language modeling (NLP), there are several tasks that make used of word embedding. Hence that, the generated embeddings from the model can be evaluated by using those downstream tasks. The results then compared with the state-of-the-art OOV handling model MIMICK Pinter et al. (2017).

3.2.1 Part-of-Speech Tagging

Part-of-speech tagging or POStagging is a task of classifying words in sentence or corpus based on the grammatical usage of the word {cite}, for example: verb, noun, adverb, etc. Given sentence $S = \{w \in \mathcal{V} | ((w_1, t_1), (w_2, t_2), \ldots, (w_n, t_n)\}$ with its POS-tag t_i, each word $w_i \in S$ is converted into sequence of character then each character is converted into character embedding $[\hat{g}]_i$ using model represented in equation 3.7.

The sequence of character embeddings $[\hat{g}]_i$ then fed into bi-lstm with logsoftmax activation function at the output to get the POS-tag. To ease up computation time, adaptive logsoftmax is used Grave et al. (2016). Instead of calculating the whole classification, the frequent and infrequent classes are separated thus there are many chance that only frequent classes needs to be calculated. The complete process of postagging process is shown in figure 3.4.

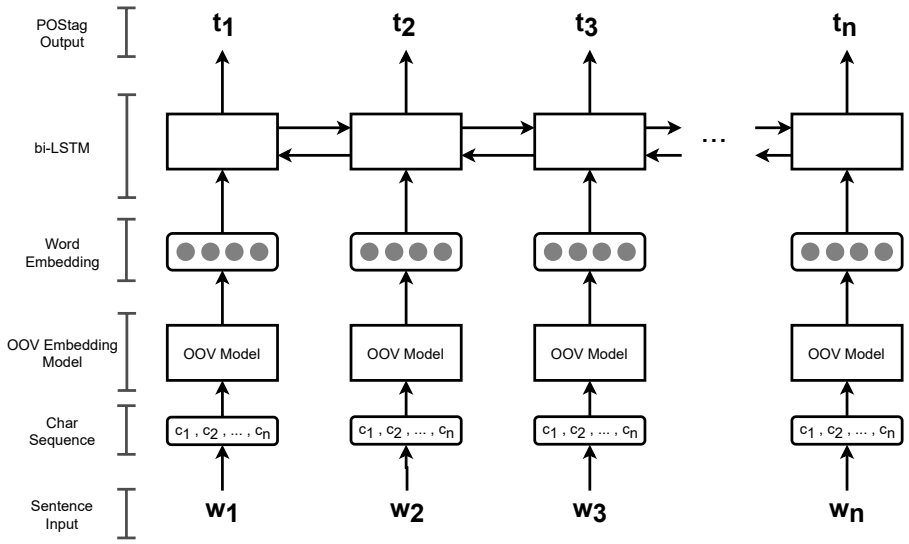

Figure 3.4: Postagging Process

3.2.2 Word Similarity Tasks

Word similarity tasks is basically task to evaluate the similarities between two words based on human given scores. Generally, several human subject were given pairs of words and asked to score its similarities. Those scores then will be used to determine the agreements between subjects that certain word pairs have stronger connection and the others are weaker. In order to calculate the agreements between the OOV generated embedding and the data that is scored by human, Spearman's rank correlation coefficient is used. Firstly, given a pair (w_1, w_2), the cosine distance of the embedding based on the generated OOV model between e_1 and e_2 for w_1 and w_2 respectfuly, are calculated using equation 3.10.

$$cosine\ similarity = \frac{e_1 \cdot e_2}{\|e_1\|\|e_2\|} \tag{3.10}$$

After all of the cosine distance for all pairs are calculated, Spearman rank's correlation from the dataset and the generated embedding are calculated by using equation 3.11 and by using equation 3.12 when no tied ranks exists.

$$\rho = \frac{n\sum_{i=1}^{n} u_i v_i - \left(\sum_{i=1}^{n} u_i\right)\left(\sum_{i=1}^{n} v_i\right)}{\sqrt{\left[n\sum_{i=1}^{n} u_i^2 - \left(\sum_{i=1}^{n} u_i\right)^2\right]\left[n\sum_{i=1}^{n} v_i^2 - \left(\sum_{i=1}^{n} v_i\right)^2\right]}} \tag{3.11}$$

$$= 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)} \text{ where } d_i = u_i - v_i \tag{3.12}$$

Chapter 4

Implementation

4.1 Datasets

4.1.1 Pretrained Word Embedding

In order to train the model, pretrained embedding is needed since the model will tries to predict embedding of from known vocabulary $v \in \mathcal{V}$ with its known embedding $w \in \mathcal{W}$. For this purpose, word2vec which trained on Google news dataset using skip-gram model containing 3 million words and phrases Mikolov et al. (2013). Word2vec contains phrases from negative sampling and for the purpose of this research those phrases did not included. The embedding has 300 dimensional vectors. Only top 40 thousands words with removal if the word is actually a phrase.

Another pretrained embedding is polyglot Al-Rfou et al. (2013) which contains multilingual embeddings. For this research, only English embedding that contains around 100 thousands with 60 dimensional vector representation. This pretrained embedding is also used in OOV handling model MIMICK Pinter et al. (2017). This model is used as baseline model.

Dict2vec is yet another word embedding that trained based on the definition of a word in dictionary Tissier et al. (2017b). Originally, this embedding was tested using word similarity tasks, hence this embedding is used as baseline model for word similarity tasks.

4.1.2 Word Similarity Dataset

Several word similarity dataset were used in order to increase the pair examples since for the datasets that is collected, the maximum number of pairs is 3000 pairs. Those datasets are asdf, sadf, asdf, asdf, asdf, asdf, sadg, and asdfsadf.

4.2 Programming Language and Tools

The model was using PyTorch 1.0.1 Paszke et al. (2017) on top of Python 3.6. Most of the basic model, for instance 2d convolution layer, 2d maxpool layer, fully connected layer, bi-lstm, and many activation functions and loss function, thus will serve enough for the purpose of this research.

4.3 Hardware

The model was trained on the freely available google colaboratory (links here) which gives randomized hardware specification, thus the exact hardware used cannot be determined. Nevertheless, this only affects the time needed to train the model not the results.

Chapter 5

Conclusion

I was right all along.

5.1 What was I right about?

I was right about the following things.

5.1.1 Previous theories were wrong

People thought they understood, but they didn't.

5.1.2 My new idea is right

Of course.

Bibliography

Al-Rfou, R., Perozzi, B., and Skiena, S. (2013). Polyglot: Distributed word representations for multilingual NLP. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 183–192, Sofia, Bulgaria. Association for Computational Linguistics. 11

Brants, T. (2000). Tnt: A statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, ANLC '00, pages 224–231, Stroudsburg, PA, USA. Association for Computational Linguistics. 4

Cutting, D., Kupiec, J., Pedersen, J., and Sibun, P. (1992). A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, ANLC '92, pages 133–140, Stroudsburg, PA, USA. Association for Computational Linguistics. 3

Forrester, J. C. (2008). A brief overview of english as a language in change. *CCSE*, abs/1508.06615(4). 1

Ghosh, S. and Kristensson, P. O. (2017). Neural networks for text correction and completion in keyboard decoding. *CoRR*, abs/1709.06429. 1

Grave, E., Joulin, A., Cissé, M., Grangier, D., and Jégou, H. (2016). Efficient softmax approximation for gpus. *CoRR*, abs/1609.04309. 9

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2015). Character-aware neural language models. *CoRR*, abs/1508.06615. 4

Kutuzov, A. and Kunilovskaya, M. (2018). *Size vs. Structure in Training Corpora for Word Embedding Models: Araneum Russicum Maximum and Russian National Corpus*, pages 47–58. 1

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR*, abs/1603.01360. 1

Li, Y. and Yang, T. (2017). *Word Embedding for Understanding Natural Language: A Survey*, volume 26. 1

Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., and Luis, T. (2015). Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal. Association for Computational Linguistics. 1, 4

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546. 3, 11

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*. 12

Pinter, Y., Guthrie, R., and Eisenstein, J. (2017). Mimicking word embeddings using subword rnns. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2, 4, 6, 7, 9, 11

S. Harris, Z. (1954). Distributional structure. *Word*, 10:146–162. 1

Tissier, J., Gravier, C., and Habrard, A. (2017a). Dict2vec : Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263, Copenhagen, Denmark. Association for Computational Linguistics. 3

Tissier, J., Gravier, C., and Habrard, A. (2017b). Dict2vec : Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263, Copenhagen, Denmark. Association for Computational Linguistics. 11

Appendix A

Code

```
10 PRINT "HELLO WORLD"
```