OOV Handling by Learning Subword using CNN based N-grams

Yonathan Purbo Santosa

Matriculation Number: 2993106

XXXX XX, XX

Master thesis

Computer Science

Supervisors:

Prof. Jehn Lehmann

Dr. Giulio Napolitano

Institut für Informatik III

Rheinische Friedrich-Wilhelms-Universität Bonn

Declaration of Authorship

I, Student name, declare that this thesis, titled "OOV Handling by Learning Subword using CNN based N-grams", and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my own work. I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Acknowledgements

blah blah blah

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Word embeddings is a method for word representation mainly used in natural language processing domain. Text data represented differently in machine unlike images and sounds data. Generally image represented as two-dimensional to four-dimensional (given channels and alpha value) matrix with finite number of cell elements containing numerical value to represent color intensity on each location (Tyagi, 2018). For example RGB image represented with 3-dimensional matrix. Each dimension represents the intensity of red color, green color, and blue color respectively in form of two-dimensional matrix. On the other hand, sound is represented as one-dimensional signal or stack of those signals (given several channels, it becomes two-dimensional) representing air pressure in the ear canal (for instance one channel for the left ear and one for the other) (Rocchesso, 1995). 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). Hence word embedding was introduced to give ability for representing text as a mathematical model namely a vector.

Text data consists of sequence of characters that is represented by codes that is standardize, as an example ASCII. In ASCII each character is represen-

ted by a number from 0 to 127 that later on extended until 255. Combinations of these number then translated by computer to represent a character. Originally, in natural language processing text data can only be modeled using one-hot vector. This vector is a one-dimensional vector that has d-dimension given d words that are known or used. Each word is represented by one dimension and depends on the used word entries in the sentence, the correspondent dimension's value is 1 and the rest is 0 hence the name one-hot vector. The one-hot vector then stacked with another one-hot vectors to represent order of usage in a sentence. Similar representation also used for representing characters. The problem with one-hot vector is that it does not have any information that infers connection between one to the other. It only encodes that certain word is used in a certain sentence in a certain sequence. Instead of sparse representation for each word, dense vector representations that maps semantic and syntactic information between words given one-hot vectors in a Euclidean space is introduced (Li and Yang, 2017; Mikolov et al., 2013b). Hopefully, 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 (Mikolov et al., 2013b; Al-Rfou et al., 2013; Tissier et al., 2017a). These word embeddings then can be used to do many downstream tasks, such as POS-tagging, 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 preferred because the size of the vocabularies will be higher and more words connection can be inferred from the corpus (Kutuzov and Kunilovskaya, 2018). On top of that, multiple usage of a vocabulary might also be used as a training data as an examples of cases of vocabulary usage in a sentence. However, it is not possible to have enough corpus since the language itself is creative and changing overtime (Forrester, 2008; Jurafsky and Martin, 2009). Fur-

thermore, there are cases of typographical error, especially on social media platform where anybody willingly write text over these platforms (Liu, 2010) and some of these words may not be present in the corpus hence not included in the vocabulary even though there exist another word that has the exact same meaning to those words thus it should more or less has close distance to the standard or original word(Eisenstein et al., 2012). In addition with the increased number of smartphone which uses touch screen, increased number in typographical error is to be expected (Ghosh and Kristensson, 2017). All this words that are non-existent in the corpus because of inability to collect such corpus or simply because of emergence of new slang or typographical error making the embedding to be unknown are called as *out-of-vocabulary* (OOV) words. One may use simple approach by assigning unique random embedding for every OOV or by replacing OOV with an unknown $<UNK>$ token with randomly initialized embedding (Garneau et al., 2019), that later hoped to be generalized in training. Despite the fact that it can be used for OOV, 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 embedding for OOV words will be generated by the model. Previous *state-of-the-art* used LSTM to infer OOV embeddings called Mimick (Pinter et al., 2017). For this model to infer OOV embedding, character embedding was first randomly initialized with a set of characters as its vocabulary. The character embedding then used to transform sequence of characters in a word into sequence of embedding. The sequence of the character embeddings then forwarded into bidirectional Long-Short Term Memory (bi-LSTM) then to fully connected layer. In language model, bi-LSTM generally works by separating sub-word by remembering and forgetting previous sequence from both end. How LSTM works in connection to this problem will be further explained in chapter 4. The last hidden state of the bi-LSTM then will be used to infer its embedding.

By architecture of LSTM, the gates inside the hidden neuron might drop previous information. Hence a 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 sub-words considered important, hence the information is incomplete since the previous information will be dampened by LSTM hidden neuron interiors if the next sequence of sub-words are considered more important. This problem will be explained further in chapter 4. From the explanation above, proposal of a new method to handle OOV is created.

1.2 Objectives

For OOV to be inferred, a model that are able to generate embedding for the correspondence word has to be created. In MIMICK, the whole sequence is processed by a bi-LSTM. By the problem mentioned earlier, instead of taking the whole words as a sequence and considering its importance based on time and occurrence using bi-LSTM, n-grams will be used to pick which grams (set of sequence) that are considered to be important. In theory this method should gives better results in downstream tasks since the information fed is complete and only left for the model to pick which n-grams features 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 learn the existed features and pick which features needs to be considered by using character embedding and treat the character sequence embedding as two-dimensional matrix. The features picked then will be processed to predict the word embedding for the input word using feedforward network.

1.3 Contributions

1. An improved OOV handling model for downstream task
2. Evaluation on different settings for baseline model and the proposed model

1.4 Thesis Structure

The remainder of this document is structured as follows. In the Related Work chapter, the previous works that are in relatives with research done in this documents are mentioned especially the baseline used in this research. In the following, Preliminaries chapter, base theories for feedforward neural network, recurrent neural network, and n-grams that are considered to be needed are explained in details here. On top of that, the problem of the previous *state-of-the-art* will be explained in depth here. In the Method chapter, the method of solution proposed to the problem and the testing method for analyzing the results are explained. In the Implementation chapter, the method of solution proposed to the problem are described in technical way to show how it is implemented and tested. In the Results and Discussion chapter, the results are shown and discussed with the relation with the previous *state-of-the-art*. Lastly, Conclusion chapter talks about the conclusion that are able to be pulled from this research.

Chapter 2

Related Work

Polyglot is one of word embedding that focused on multilingual application (Al-Rfou et al., 2013). A total of one hundred and seventeen languages word embeddings were generated to give availability of different language models to be trained. Previously, specific language features were hand crafted by experts of specific language(Al-Rfou et al., 2013). This makes applying a language model that are trained using commonly available language features harder, hence the creation of Polyglot word embedding. This embedding was trained on Wikipedia article and has no OOV handling if there exist word that does not used in Wikipedia. This pretrained embedding is also used in the baseline model MIMICK for generating OOV embedding in many languages.

Word2vec is another word embedding that is trained using skip-gram model (Mikolov et al., 2013a). The available language choice for this pretrained embedding is English. 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. The model tried to project the input $w(t)$ to the output to predict the context words (Mikolov et al., 2013a). This model is highly dependent 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 for it.

Dict2vec is yet another embedding that is trained by looking up definitions of words from Cambridge dictionary (Tissier et al., 2017b). 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 (Tissier et al., 2017b). 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 improvements over vanilla implementation of word2vec and fasttext (Tissier et al., 2017b).

As aforementioned above, those word embedding has no way of handling OOV words rather than assigning some unknown token $<UNK>$ or let the downstream tasks model to train from the random generated embeddings. This case fueled MIMICK, an OOV handling model to be created (Pinter et al., 2017). MIMICK uses bidirectional Long-Short Term Memory (bi-LSTM) to process characters of an OOV word to produce embeddings. The OOV embedding generation process is taken from quasi-generative perspective, meaning that the original embedding assumed to has some form that could generate the embeddings (Pinter et al., 2017).

Kim et al. (2015) created a language model for several languages. This model used character n-grams by using character embeddings and processed with CNN like an image. The results from these process then passed through a highway network. A highway network controls whether the information from the input would also be carried to the next process or to be changed with something else. The output of the highway network then passed through LSTM to predict the next word. In those research, the results of the model for OOV nearest neighbor trained with or without highway network were compared. The

results was that the one trained without highway network has closer distance to a word that has smallest edits while the one that trained with the highway network has closer distance to a word that has similar orthographic (Kim et al., 2015).

Part-of-Speech-Tagging (POS-tagging) is a process of determining grammatical category called a 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 problem, 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 bi-LSTM that took sequence of word embeddings representing a sentence or parts of sentence then categorize each word embedding for its tag based on the usage in that sentence.

As aforementioned above, some word embedding model such as Word2vec, Polyglot, and Dict2vec has no OOV handling, thus creating a model to predict such word becomes a research interest. One of the model that tries to tackle this problem successfully by using bi-LSTM to predict embedding given sequence of characters from a word from a pretrained embedding named Mimick (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 POS-tagging.

In spite of the performance of Mimick, there are evidence that CNN that used for sequence modeling can outperform LSTM and RNN architecture (Bai et al., 2018). Moreover, CNN converge faster than LSTM, RNN, and GRU based on the number of iteration despite of the sequence length (Bai et al., 2018). The model called temporal convolution network (TCN) is basically a multilayer CNN with different dilation to factor the collection of input dimension for each layer. The model was evaluated using several sequence modelling

tasks.

For training an OOV model, some kind of feature extraction method needs to be used. One of feature extraction method called n-grams often used to capture word features. N-grams can be applied in both character grams in a word or word grams in 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

Preliminaries

3.1 Feedforward Neural Network

Feedforward neural network or also known as multilayer perceptron are a mathematical model that inspired from how neuron works in biological body (Goodfellow et al., 2016). The model takes numerical input as the stimulus and produces numerical output as the response. This model is a simplification of organism nervous system. Originally it was a single layer of input and a single layer of output. The cells within input layer and output layer is called neuron. Those neuron are responsible for the numerical representation of the input and the output. Given an input vector $\mathbf{x_i}$ with n-dimension from a data-set $D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_i}, y_i)\}$, the model tries to predict a target y_i correctly given a set of weights $\mathbf{w}$. The weights act as stimuli intensity value, thus different weight values will produces different responses given the same amount of input stimulus. The output o_i is calculated by using the dot product between $\mathbf{x_i}$ and $\mathbf{w}$ as shown on equation 3.1.

$$o_i = \mathbf{x_i} \cdot \mathbf{w} \tag{3.1}$$

$$o_i = \sum_{j=1}^{n} (x_{ij} \times w_j) \tag{3.2}$$

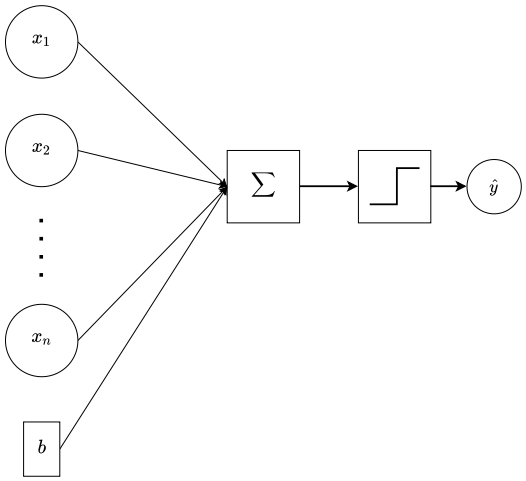

Figure 3.1: Perceptron

After calculating the output o_i and another parameter bias b is added, the result passed to activation function $f(o_i)$ to produce the perceptron output as shown on equation 3.3.

$$f(o_i) = \hat{y}_i = \begin{cases} 1 & \text{if } \; o_i + b > 0, \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

The input information fed through the model in chain with no feedback thus the name feedforward (Goodfellow et al., 2016). When the output are fed back into the model, the model becomes recurrent network and it will be explained in the next section. The perceptron model is depicted in figure 3.1.

The results then compared with the original output y_i in the dataset to learn the correct parameter weight $\mathbf{w}$ and bias b by using backpropagation algorithm. The algorithm defines a cost function and calculate the gradient of the current cost, then the gradient information is processed by another algorithm called stochastic gradient descent to try to find the optimal parameters that produced the minimum cost. If $\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}) = \mathbf{x} \cdot \mathbf{W}$ and the cost

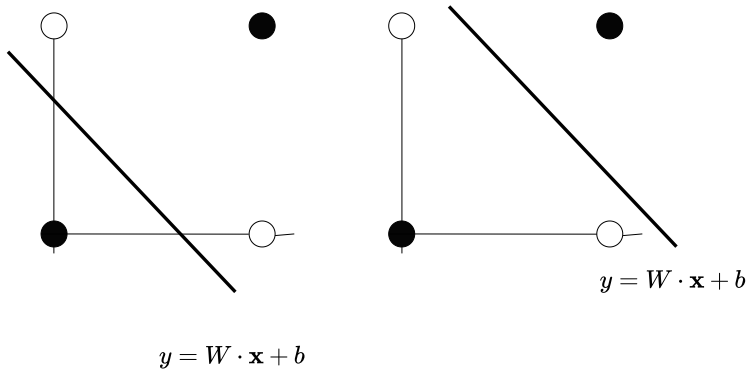

Figure 3.2: XOR Problem

$z = J(\mathbf{y})$, the simple calculation of the gradient can be calculated as follows,

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{3.4}$$

$$= \sum_j \frac{\partial z}{\partial y_j} \sum_k w_k \tag{3.5}$$

The correction on parameter $\mathbf{W}$ can be calculated by following calculation with some learning parameter η as shown on equation 3.6.

$$\hat{w_i} = -\frac{\partial z}{\partial x_i} \times \eta \times w_i \tag{3.6}$$

Notice that the gradient in equation 3.6 is multiplied by -1 because the gradient points uphill and to find the minimum cost, the parameters needs to traverse downhill on the cost function. The learning parameter η act as penalization factor for the gradient to avoid jumping over the optimal solution. Generally, η is set to be near zero.

On the development of perceptron, it is clear that model consisting of single layer of input and single layer of output does not enough to solve XOR problem (Goodfellow et al., 2016), since what perceptron does is separating the space with a hyperplane, or in XOR problem a hyperline as shown on figure 3.2. Thus multilayer perceptron were introduced. This model consist of single layer of input, single layer of output, and one or more hidden layer. On top of that, more non-linear activation function were introduced. Some of those are

sigmoid, tanh, rectified linear unit (ReLU), and softmax described in equation 3.7 until 3.10.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.7}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.8}$$

$$ReLU(x) = x^{+} = max(0, x) \tag{3.9}$$

$$softmax(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \tag{3.10}$$

3.2 Long-short Term Memory

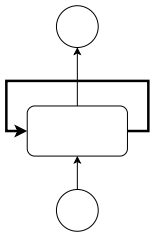

Figure 3.3: Recurrent Neural Network

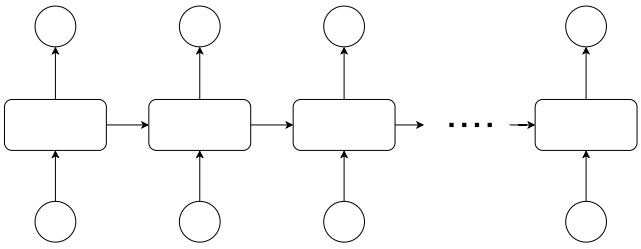

Figure 3.4: Unfolded Recurrent Neural Network

As aforementioned before, recurrent neural network (RNN) is a neural network model that has feedback to its own neuron depicted in figure 3.3. This model used for processing sequential data (Goodfellow et al., 2016). The idea is given sequence of input with length of t, $\mathbf{x}_i^{(t)} = x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(\tau)}$, the input is

processed with shared parameter to give t-lengths output as depicted in figure 3.4. Common usage of RNN is for processing sentence. For example, given two sentences "I went to Paris in 2004" and "In 2004 I went to Paris". If we query the model to extract information from both sentences and ask when did the narrator went to Paris, 2004 would be the relevant information regardless when it is appears in a sentence (Goodfellow et al., 2016). If such task is trained using a feedforward neural network that processes fixed size inputs, the parameters for each input feature will be separated for each sequence. In comparison with feedforward neural network, RNN will use the same parameters to process all the input sequence.

The problem with RNN is that when given a really long sequence, either the gradient calculation will vanish since if weight is near zero and is multiplied several times sequentially over time and the gradient exploded if the weight is larger than one and it will be multiplied several times (Goodfellow et al., 2016). Those problems making processing long sequence on RNN unfavorable. Hence another method called long-short term memory (LSTM) was introduced. LSTM was created with idea in mind that some paths exist to give gradient ability to flow for long sequence. This was achieved by introducing self-loops inside the hidden layer that has time-scale mechanism that can change dynamically based on the input (Goodfellow et al., 2016). New components called cell gate C_t, forget gate f_t, and input gate i_t are introduced to let the recurrent network split the graph of the hidden unit thus gradient can be flowed longer by depending only from the output o_t or from both output and hidden states o_t and h_t respectively. The cell gate interacts with input i_t and previous hidden state h_{t-1} to decide the current hidden state h_t. The

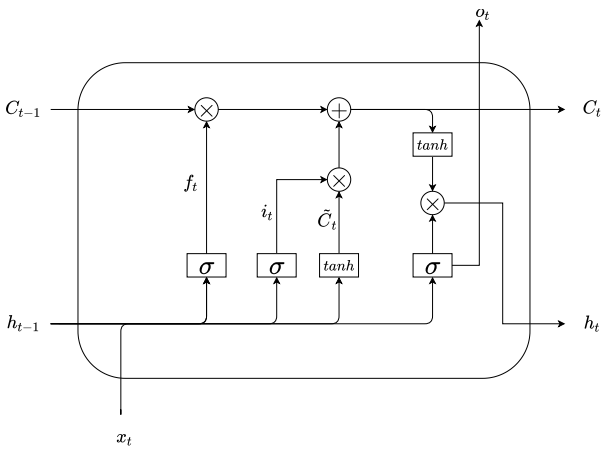

Figure 3.5: Gates inside LSTM hidden cell

calculation process of LSTM for each time step $t = 1$ to $t = \tau$ is as follows,

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.11}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.12}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.13}$$

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{3.14}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.15}$$

$$h_t = o_t \times tanh(C_t) \tag{3.16}$$

$$\hat{y}_t = softmax(o_t) \tag{3.17}$$

Cell gate C_t controls the hidden state as described in equation 3.16. If $C_t = 0$, the hidden state $h_t = 0$, meaning the previous information of a certain features is dropped for the gated hidden states. The LSTM hidden cell's architecture is depicted in figure 3.5.

On top of the normal sequence, the reverse sequence can also be calculated starting at time step $t = \tau$ to $t = 1$ then the output is joined with the forward sequence. This method is called bidirectional-LSTM (bi-LSTM) depicted in figure 3.6.

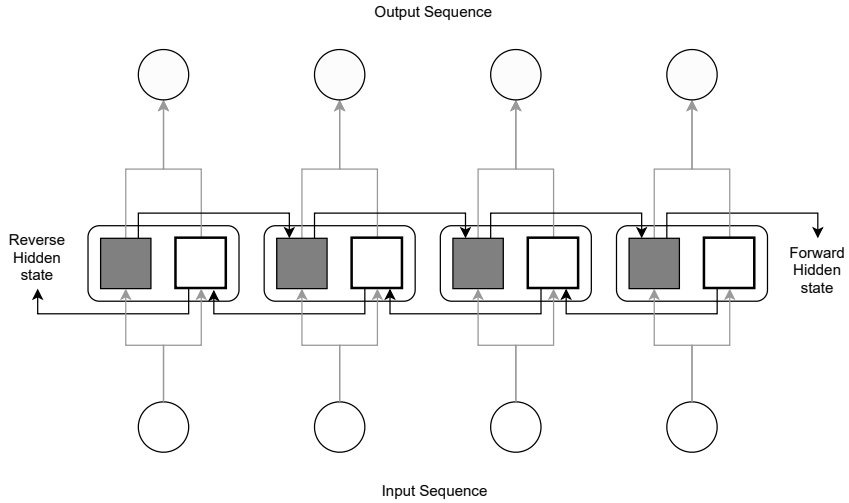

Figure 3.6: bi-LSTM with 4 sequence

3.3 Mimick

In this research, MIMICK is used as baseline model. Firstly, pre-trained embedding which contains word w_i and its embedding e_i is used as input and target respectively. Afterward, the character embedding $g_i \in \mathbf{G}$ for each character $c_i \in \mathbf{C}$ was defined. Each word w_i as input first broken down into sequence of characters, $w_i = [c_1, c_2, \ldots, c_n]$, then each character was transformed into its embedding, producing sequence of character embeddings $[g_1, g_2, \ldots, g_n]$. Those character embeddings then fed into bi-LSTM as a sequence to extract the features of the word input. The last hidden states of both forward $\mathbf{h}_f$ and backward $\mathbf{h}_b$ then concatenated and fed into a fully connected layer with parameters $\mathbf{T}_h$, $\mathbf{b}_H$, $\mathbf{b}_T$ and $\mathbf{O}_T$ and nonlinear function g to predict the embedding of the input word as described by equation 3.18.

$$f(w) = \mathbf{O}_T \cdot g(\mathbf{T}_h[\mathbf{h}_f; \mathbf{h}_b] + \mathbf{b}_h) + b_T \tag{3.18}$$

The objective of the training is to get the predicted embedding $f(w_i)$ as close as the pre-trained word embeddings e_i. This was done by minimizing the

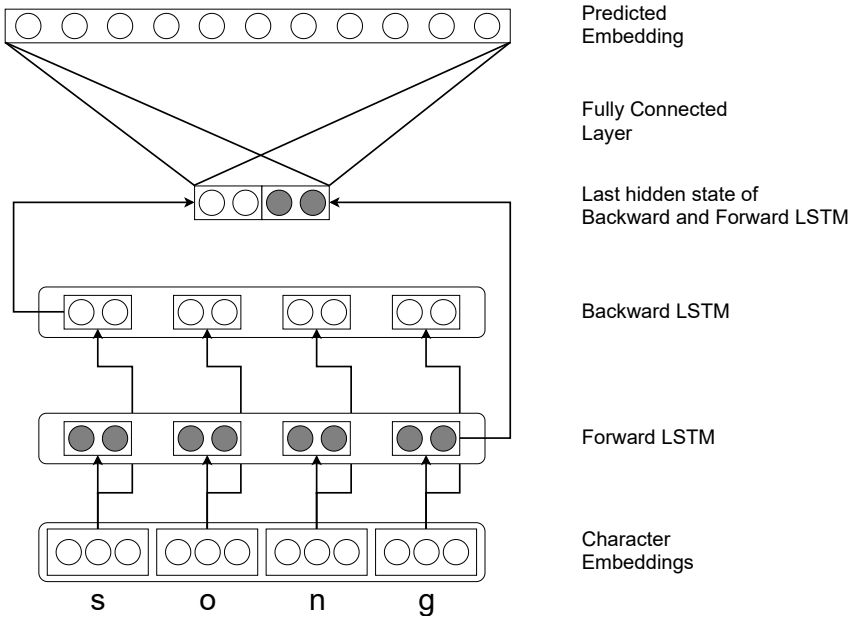

Figure 3.7: Mimick architecture

squared Euclidean error,

$$\mathcal{L} = \|f(w_i) - e_i\|_2^2 \tag{3.19}$$

The full process of predicting the embedding $f(w_i)$ is depicted in figure 3.7.

As aforementioned in the previous section, cell gate C_t controls which hidden neuron in hidden states at time t will pass through to the next sequence. Mimick uses only the last hidden state of the bi-LSTM thus increases the chance of the early important sequence to be dropped when cell gate C_t decided to drop the information at certain point when there exist some important sequence to trigger the cell gate C_t to drop previous information. Although bi-LSTM could serve the purpose to include the early sequence, the intermediate important sequence that is appears in the middle of two important sequences might be dropped because of the cell gate C_t decided to drop previous sequence. This problem fueled another approach to be introduced, namely using convolutional neural network as the feature extractor for the character sequence.

3.4 Convolutional Neural Network

Convolutional neural network (CNN) is a model that is used to process data that has grid topology (Goodfellow et al., 2016). For a time series data that has a regular time interval, CNN can process this as a one-dimensional grid data and for an image data CNN can process this as a two-dimensional grid or 3-dimensional grid given the number of channel presents on the image data. This model concept was first introduced for handwriting recognition (LeCun, 1989).

In general, convolution is operation of two function, the data and the kernel, that is taking values from both function and elementwise multiplied was done and then summed to get the total overlaps between both function at time t. In general, the kernel has only limited size that has non zero values while the rest is zero. Thus the convolution operation is done locally by processing parts of the data several times. To obtain the entirety processed data, the kernel needed to be shifted in all direction depending on the data dimension. This process is easier to be explained with mathematical expression. In equation 3.21, one-dimensional data or function $f(t)$ is being convoluted with a kernel $g(t)$.

$$h(t) = (f * g)(t) \tag{3.20}$$

$$h(t) = \int_{-\infty}^{\infty} f(u)g(t-u)du \tag{3.21}$$

In discrete type signal, the calculation processes becomes as shown in equation 3.23.

$$h(t) = (f * g)(t) \tag{3.22}$$

$$h(t) = \sum_{u=-\infty}^{\infty} f(u)g(t-u) \tag{3.23}$$

For two-dimensional data, the discrete convolution function becomes as shown

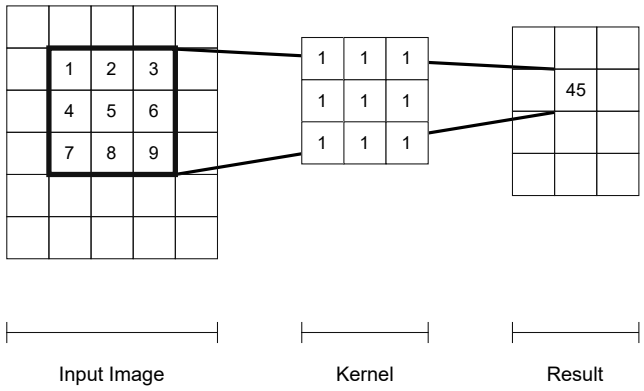

Figure 3.8: Convolution process of input image with kernel size 3×3

in equation 3.24.

$$h[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[m,n] \cdot g[i-m, j-n] \tag{3.24}$$

In CNN, one of the function is the input data and the other is the weight. Typically, the weight's, also known as kernel, size is smaller than the input data although it is possible to have kernel size that is bigger than the input but there is no reason to have larger kernel if the objective is to learn local features of the data. To process an image data, the image input is convoluted with some kernels to produce different spatial features. This features then will be processed with a feedforward neural network to produce some prediction of classification or regression. The convolution process of one patch of an input image is depicted in figure 3.8.

Another intermediate layer that can also be used in CNN called maxpooling layer. Maxpooling is a process of finding a maximum value inside a given window from a given grid. In CNN, maxpooling is used for finding within the input data that gives the highest response with a given kernel. Then the process of convolution and maxpooling is repeated until desired architecture is produced. The process of one patch of an input image is depicted in figure 3.9. Maxpooling process act as a gate for the highest response to receive backward connection for correcting the kernel. This is so that only parts of the image that has highest response will also corresponds to the correction of the kernel

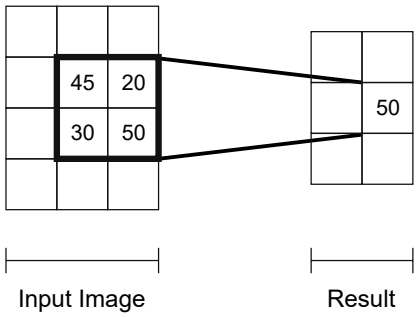

Figure 3.9: Maxpooling process of input image with window size 2×2

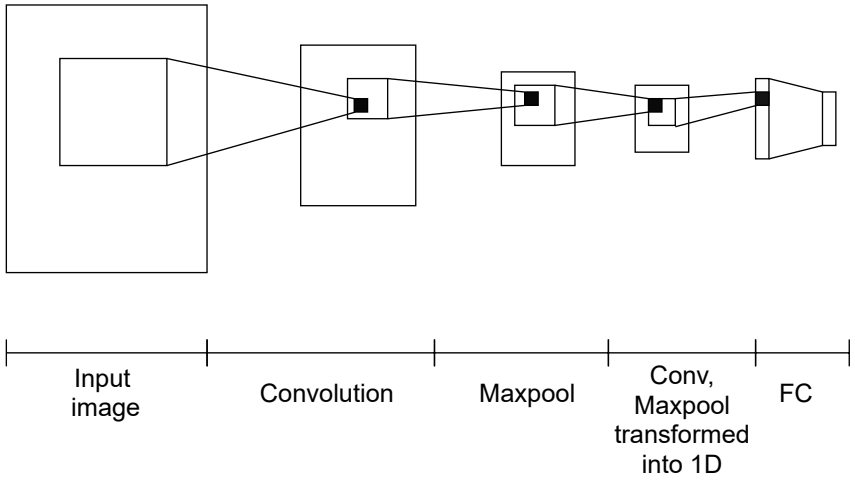

Figure 3.10: Example of CNN architecture

while the others treated as less useful.

Convolution and maxpool then can be combined together to produce features map that act as input to the fully connected network. As an example, CNN with two convolution and maxpool architecture is depicted in figure 3.10. In most cases, after doing convolution, non-linear activation function are applied.

3.5 N-grams

N-grams is a method that is mostly used for word prediction (Jurafsky and Martin, 2009). Given a sentence,

`Please do not sit ...`

word *on* or *at* is more likely to follow instead of *run* or *bacteria*. In short, the previous task can be written as $P(w|h)$, probability of the next word w given some history part of sentence h. In previous case, the history h is "*Please do not sit*" and the probability in question is the following word w will be "*on*". To solve this task, counting the appearance of history h followed by word w can be used to retrieve the probability (Jurafsky and Martin, 2009). Mathematically it can be written as follow,

$$P(\text{on}|\text{Please do not sit}) = \frac{C(\text{Please do not sit on})}{C(\text{Please do not sit})}$$

Previous method can give good estimation, but because language is creative and new sentences generated every time, everything that exist on the internet is not enough to produce good estimate (Jurafsky and Martin, 2009). On top of that, if the joint probability of the sequence would be calculated, there will be many estimations where each estimation is not exact because there is no way for the probability to be calculated given long sequence of preceding words because as stated above, language is creative (Jurafsky and Martin, 2009). Given sequence of words $(w_1, w_2, \ldots, w_n)$, the joint probability of these sequence can be calculated by using chain rule as follows,

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_n|w_1, w_2, \ldots, w_{n-1}) \tag{3.25}$$

$$= \prod_{i=1}^{n} P(w_i|w_1, \ldots, w_{i-1}) \tag{3.26}$$

As shown on equation 3.25, each occurrence of preceding sequence that followed by the desired word is estimated by counting the occurrence as shown in equation 3.5 for the whole history.

Instead of previous calculation, better way to calculate the word W given history h is needed because as stated above, language is creative making calculating exact probability impossible and there will be too many estimation if

there is long sequence that precedes the target word. Hence n-grams has been introduced to approximate the probability of word w from last few sequence of the history h instead of a whole (Jurafsky and Martin, 2009). For instance, only two preceding sequences will be taken into calculation. In other words, instead of following probability calculation,

$$P(\text{on}|\text{Please do not sit}) \tag{3.27}$$

the approximation of the probability will be as follows,

$$P(\text{on}|\text{not sit}) \tag{3.28}$$

In other words, the conditional probability then approximated by following equation,

$$P(w_n|w_1, w_2, \ldots, w_{n-1}) \approx P(w_n|w_{n-2}, w_{n-1}) \tag{3.29}$$

This approximation method then can be used for joint probability approximation as follows,

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i|w_{i-2}, w_{i-1}) \tag{3.30}$$

for $P(w_j) = 1$ if $j < 1$.

There are multiple n-grams model that are differentiated by the number of sequence used to estimate probability of the next word w. For example, preceding word in the history h is used for estimating the next word w in bigram model and two preceding words in the history h is used in trigram model. In general, the window used for n-grams are configurable to the needs of the expected results.

Another application of n-grams is that the sequence of character is used instead of sequence of words to estimate the probability. Differently from word n-grams, character n-grams are able to infer the morphological features of a written sentence or words (Kulmizev et al., 2017). Instead of using history of words, character n-grams used history of character to predict the next character. All the equation is similar to the word n-grams. On top of that, character

n-grams are really good for detecting patterns in case of typographical error and represented less sparsely compared to word n-grams since there are only so much character compared to the words made up from existing characters (Kulmizev et al., 2017).

Chapter 4

Method

4.1 Out-of-Vocabulary Model

4.1.1 Sequence Feature Extraction

OOV problem is handled from quasi-generative perspective as aforementioned in chapter Introduction by using neural language model under assumption that there is a form that could generate embedding for the original embedding. Hence that, the original vocabulary and its embedding is used for training the model to generate the embedding. In chapter Introduction, reasons why MIMICK could perform worse is because the OOV embedding is generated from the last hidden states of the bi-LSTM and the hidden states is controlled by cell gates C_t making the information that is carried on is the most recent information. If at certain time step t the cell gates decided to forget past information, then the early information might not be coded into the hidden state. On top of that, there are evidences that recurrent architecture could perform worse than CNN for sequence modelling (Bai et al., 2018).

If explained formally, when $C_t = 0$ from equation 3.14, hidden state from equation 3.16 will also be 0, resetting to its starting state, rendering hidden states prior to time t gone. This problem can be solved by using bi-LSTM, since bi-LSTM processes sequence in forward and reverse order making both

$un|recogniz|able$

$inter|national|ities$

$oto|rhino|laryngolog|ical$

$hepatico|chol|angio|gastro|stomy$

Figure 4.1: 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 4.2: 4-grams examples

early and later sequences held by the last hidden state for each reverse LSTM and forward LSTM respectively. Another problem might arise when we need to divide sequence into more than three subsequence as shown on figure 4.1. Hence another approach is needed since intermediate subsequence might get deleted or carried along with the later sequences even with bi-LSTM. Another method that might be able to solve this problem for Mimick is by increasing hidden size and hope that it will be able to compensate the sequence that is dropped by the cell gate in the other hidden cell.

For all subsequence to be processed, we need a method that accounts for the

whole 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 4.2. Before processing the n-grams, the word first split into sequence of characters and then each character is transformed into embedding and processed with CNN inspired from CNN word n-grams (Kim, 2014). Those sequences of character embeddings then fed into learning algorithm. This idea is similar to how human tries to recognize an unseen word by reading subword that is understandable beforehand when no explanation or context were given. 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 the loss function,

$$\mathcal{L} = \|f(w_i) - e_i\|_2^2 \tag{4.1}$$

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 $\{g\}^m$. $\{g\}^m$ becomes 2-dimensional matrix that has size of $m \times b$. In summary, given word w will be transformed using embedding generation function h into $\{g\}^m$ as shown on equation 4.2.

$$h : w \rightarrow \{g\}_1^m \tag{4.2}$$

To process $\{g\}^m$ like an n-grams, CNN is used inspired by CNN n-grams implementation by Kim (2014). CNN n-grams is basically a method to do convolution on matrix by using a kernel $k_i^{b\times n} \in K$ for n is the window size of the grams and b is the dimension size of the character embedding. This operation is represented with $*$ symbol as stated in equation 3.23. This operation produced another vector $\hat{l}$ that represents the value of each grams, then non-linearity is

applied to this vector by using ReLU activation,

$$ReLU(x) = max(0, x) \tag{4.3}$$

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 $\{g\}^m * k_i$ will be max-pooled to produce one number. In details, from given sequence of character embedding $\{g\}^m$, 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. Furthermore, by using several window sizes for n-grams (bigram, trigram, etc.) by changing the size of the kernel more features will be able to be learned. For instance bigram will has a kernel with size $b \times 2$, trigram will has a kernel with size $b \times 3$, and so on and so forth, making the different sizes of n-grams can be trained together only then concatenated later.

4.1.2 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 $hardtanh$ matching the maximum and minimum bound of the pretrained embedding $\mathcal{W}$ resulting a new embedding vector $\tilde{e}$. The generated embedding vector $\tilde{e}$ then passed into a highway network to decide whether some information should be carried or should be forgotten to obtain a new sets of embedding. Given the output of max-over-time pooling m, The highway network is calculated by the following equation,

$$t = ReLU(f(\mathbf{W}m + b)) \tag{4.4}$$

$$\mathbf{Z} = t \odot g(\mathbf{W_H}m + b_\mathbf{H}) + (1 - t) \odot m \tag{4.5}$$

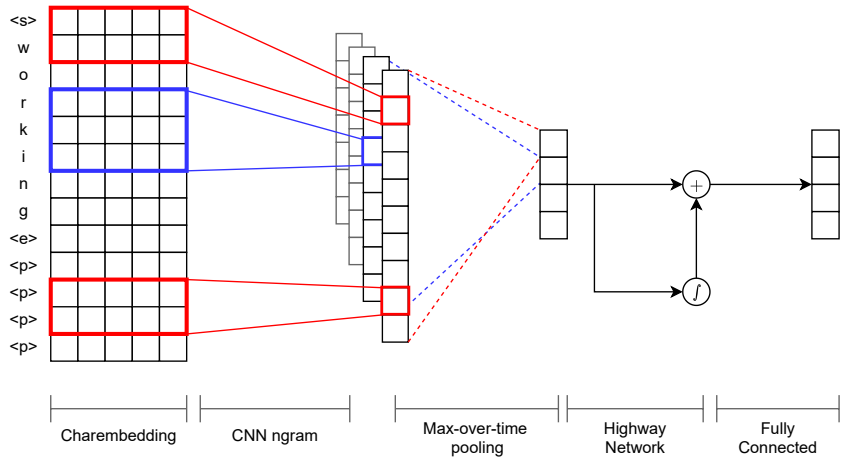

Figure 4.3: OOV Inferencing Model

The complete process from input word, feature extraction, until predicting embedding is shown on figure 4.3.

On figure 4.3, starting and ending token is added at the beginning and the end of the word respectively. Furthermore, padding token is added if the input word is shorter than the longest input size in the minibatch. 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. The reason for this is so that different length of words are able to be processed by the model.

4.1.3 Error and Backpropagation

The predicted embedding $\tilde{e}$ from the model then compared with the original embedding e to learn new parameters for the neural network using mean squared error function,

$$Error = \frac{1}{2}\|e - \tilde{e}\|_2^2 \tag{4.6}$$

By minimizing $Error$ it is similar by minimizing $\mathcal{L}$ shown in equation 4.1. The error then backpropagated to fine-tune the neural network parameters, character embedding $\mathcal{G}$, and the kernel $k \in K$.

4.2 Measuring Performance on Downstream Tasks

In natural language modeling (NLP), there are several tasks that make use 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).

4.2.1 Part-of-Speech Tagging

Part-of-speech tagging or POS-tagging is a task of classifying usage of 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 that exist in the vocabulary $w_i \in \mathcal{V}$ and $w_i \in S$ is transformed into embedding e_i. For the OOV, every sequence of the characters building a word transformed into sequence of character embedding $\{g\}_i^m$ using model represented in equation 4.2 then the embedding $\tilde{e}_i$ is predicted using the OOV handling model, else the original embedding is used if the word exist inside the vocabulary $\mathcal{V}$. This was done by masking the output of the OOV handling model and the original embedding in the following way,

$$embedding = mask \odot e_i + (1 - mask) \odot \tilde{e}_i \tag{4.7}$$

$$mask = \begin{cases} \vec{1} & \text{if } \; w_i \in \mathcal{V} \\ \vec{0} & \text{otherwise} \end{cases} \tag{4.8}$$

This way the gradient would not flow into the OOV model if the word exist in $\mathcal{V}$ and will flow if the word does not exist in $\mathcal{V}$ and the embedding was generated by the OOV model.

The sequence of embeddings $\tilde{e}_i$ or e_i then fed into bi-LSTM and the output

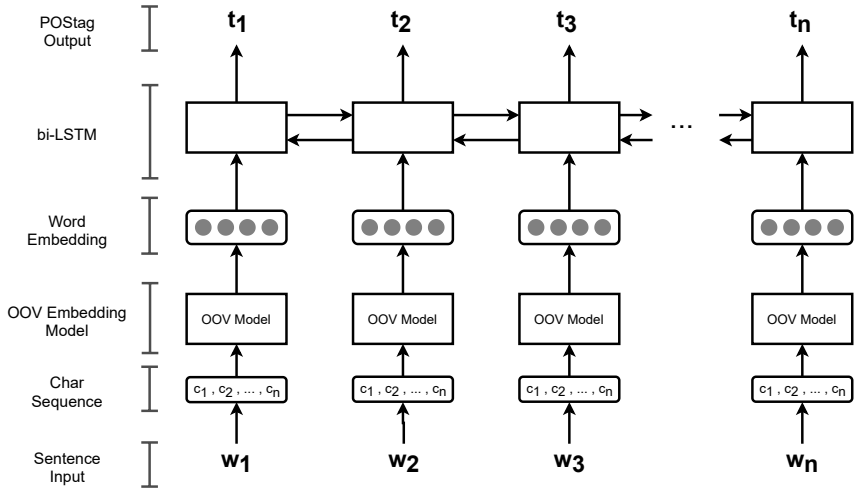

Figure 4.4: Pos-tagging Process

was passed through LogSoftmax activation function,

$$LogSoftmax(x_i) = log\left(\frac{exp(x_i)}{\sum_j exp(x_j)}\right) \tag{4.9}$$

to classify the POS-tag t. To ease up computation time, adaptive LogSoftmax is used (Grave et al., 2016). Instead of calculating the whole tag classification, the frequent and infrequent classes are separated thus there are many chances that only frequent classes needs to be calculated before trying to calculate the infrequent classes. The complete process of POS-tagging process is shown in figure 4.4. After training was done, the accuracy of the POS-tagger based on different OOV handling model were compared.

4.2.2 Word Similarity Tasks

Word similarity tasks is basically task to evaluate the similarities between two words based on human given scores. In practice, several human subjects 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 $\tilde{e}_1$ and $\tilde{e}_2$ based on the generated embedding from OOV model for w_1 and w_2 calculated respectively using the following equation,

$$CosineSimilarity(e_1, e_2) = \frac{e_1 \cdot e_2}{\|e_1\|\|e_2\|} \tag{4.10}$$

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 4.11 and by using equation 4.12 when no tied ranks exists. The results of both MIMICK and the proposed model from several word similarity datasets then averaged and compared.

$$\rho = \frac{n\sum_{i=1}^{n} u_i v_i - \left(\sum_{i=1}^{n} u_i\right)\left(\sum_{i=1}^{n} v_i\right)}{\sqrt{\left[n\sum_{i=1}^{n} u_i^2 - \left(\sum_{i=1}^{n} u_i\right)^2\right]\left[n\sum_{i=1}^{n} v_i^2 - \left(\sum_{i=1}^{n} v_i\right)^2\right]}} \tag{4.11}$$

$$= 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)} \text{ where } d_i = u_i - v_i \tag{4.12}$$

Chapter 5

Implementation

5.1 Preparation

5.1.1 Dataset Preparation

Character Embedding

The character dictionary consists of first 128 ASCII characters with deletion for non character types (the first 32 ASCII symbols). Then the character embedding was initialized by randomizing from normal distribution with $\mu = 0$ and $\sigma = 1$. Another entries such as unknown token $<UNK>$, starting token $<s>$, ending token $<e>$ and padding token $<p>$ were added along with the random character embedding initialization.

Pretrained Word Embedding

In order to train the model, pretrained embedding is needed since the model will tries to predict embedding 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 (name, hyperlink, connected words, etc.) was used (Mikolov et al., 2013b). The

embedding contains 300-dimensional vectors. Word2vec [1] contains phrases that frequently appears, for instance the word New and Jersey appears frequently side by side because both of this words form name of a state in United States of America. In the original vocabulary, this phrase might be written as "New_Jersey", thus for the purpose of simplifying the input and the downstream tasks those phrases did not included. On top of phrases, the original vocabulary also includes hyperlinks which usually contains "http". Such entries will also be removed. Only first 40 thousands words with removal if the word contains "_" (underscore) or "http" is used in this research.

Another pretrained embedding is polyglot[2] which contains multilingual embeddings (Al-Rfou et al., 2013). For this research, only English embedding that contains around 100.000 words with 60-dimensional vector representations will be used. This pretrained embedding is also used in OOV handling model MIMICK which used as baseline model (Pinter et al., 2017).

Dict2vec[3] is yet another word embedding that trained based on the definition of a word in dictionary (Tissier et al., 2017a). Each words location in Euclidean space was determined by the appearance of another words in the definition that is defined by the Cambridge dictionary. Originally, this embedding was tested using word similarity tasks with removal of OOV words.

Word Similarity Dataset

Several word similarity dataset were used in order to increase the pair examples since for the datasets that is collected, the highest number of pairs is just above 3000 pairs. Those datasets used for word similarity tasks are Card-660 (Pilehvar et al., 2018), MC-30 (Miller and Charles, 1991), MEN-TR-3k (Bruni et al., 2012), MTurk-287 (Radinsky et al., 2011), Mturk-771 (Halawi et al.,

[1]Pretrained embedding available at https://code.google.com/archive/p/word2vec/

[2]Pretrained embedding available at https://polyglot.readthedocs.io/en/latest/index.html

[3]Pretrained embedding available at https://github.com/tca19/dict2vec

2012), RG-65 (Rubenstein and Goodenough, 1965), RW-STANFORD (Luong et al., 2013), SimLex-999 (Hill et al., 2014), YP130 (Yang and Powers, 2006), VERB143 (Baker et al., 2014), and Wordsim353 (Finkelstein et al., 2001).

5.1.2 Programming Language and Tools

The model was trained using PyTorch 1.1.0 machine learning library on top of Python 3.6 (Paszke et al., 2017). Most of the basic functions, for instance 2d convolution layer, 2d maxpool layer, fully connected layer, bi-LSTM, and many activation functions and loss functions and other mathematical functions were already implemented as a library in PyTorch, thus will serve enough for the purpose of this research.

5.1.3 Hardware

The model was trained on the freely available Google Colaboratory[4] which gives randomized hardware specification based on the availability, thus the exact hardware configuration used cannot be determined. The GPU engine was used to train the model. Nevertheless, this only affects the time needed to train the model and not the results.

5.2 Training

5.2.1 Training OOV model

Firstly, the pretrained embeddings acted as the datasets were split up into train-val set with 80% and 20% randomized split respectively with minibatch size of 64. The word $w_i \in \mathcal{V}$ becomes the input of the model and the word embedding $e_i \in \mathcal{W}$ becomes the target. The input word split into sequence of characters and starting token $<s>$ and ending token $<e>$ were added at the

[4]Google Colaboratory available at `https://colab.research.google.com/`

Table 5.1: OOV Handling Model Parameters

Hyperparameter	**Mimick**	**CNN**
Learning Rate (η)	[0.01; 0.1]	0.1
Batch size	64	64
Epoch (word2vec)	100	100
Epoch (polyglot-en)	100	100
Epoch (dict2vec)	100	100
Momentum	0.5	0.5
Dropout	0	0.5
Num features	[50; 100; 200]	[20; 50; 100]

beginning and at the end of the word respectively. For every minibatch, the longest word will be used as the maximum length. Every word that was shorter than the longest word will be padded with a padding token $<p>$. The sequence of characters then transformed into character embeddings then processed by the model producing the predicted word embedding. The MIMICK model was trained with learning rate $lr = 0.01$ with no dropout as dropout does not work well just as in MIMICK (Pinter et al., 2017). On the other hand, CNN model was performing better with dropout when the model was pre-tested using different parameters. In summary, the hyperparameters setting is shown in table 5.1.

For the proposed model, the character embeddings were processed with CNN n-grams following Kim (2014) architecture for word n-grams. N-grams with window size $n = [2, 3, 4, 5, 6, 7]$ were used with respective kernel size $n \times b$ to simulate n-grams. Different n-grams sizes, for instance $n = [2, 3, 4]$ and $n = [5, 6, 7]$ can be used on top of the whole n-grams sizes. The results on the downstream tasks for the different settings then compared with the whole model and with the MIMICK model.

After max-over-time pooled, the vector representation then passed into

fully connected network with highway network to produce the predicted word embedding. The error then calculated using Mean Squared Error as mentioned on equation 4.6 then back-propagated to update the weights.

Similar datasets were used to train MIMICK with parameters taken from the original paper (Pinter et al., 2017). After compared several parameters then adjusted to see whether there is improvement or not. The summary of the hyperparameters are as shown in table 5.1.

5.3 Evaluating with Downstream Tasks

5.3.1 Part-of-Speech Tagging

For POS-tagging task, the readily brown corpus and its tagset from NLTK[5] were used to test the performance of the model. In this tasks, two kind of evaluation methods were done. Firstly, all the word embeddings only inferred from the trained OOV model and the OOV model trained together with the POS-tagger. Secondly, only OOV from the pretrained embedding are inferred by using the OOV model and then those embeddings are used as input for the POS-tagger. For the latter method, embeddings of the words are frozen to get a notion whether the OOV model improves the results compared to replacing OOV with unknown token or a random embedding. The OOV model then changed with MIMICK (Pinter et al., 2017) to compare the results with the previous state-of-the-art.

5.3.2 Word Similarity

Word similarity task is quite straight forward. Given pairs of words, if the word is an OOV in the pretrained embedding, the word embeddings were predicted from OOV models then the cosine similarity of the word embeddings

[5]Available at https://www.nltk.org/

are calculated and compared between two models, else the original embedding from the pretrained embedding is used.

The baseline embeddings used for this task, dict2vec (Tissier et al., 2017a), was also tested using random OOV embedding by randomly giving OOV word random embedding and choosing the maximum results of several tries. On top of that, other two pre-trained embedding used in this research were also used for comparing results. The results then compared with if the OOV were predicted using the OOV models trained.

Chapter 6

Results and Discussion

6.1 OOV model

After training the model was done, some words were fed into the model and the nearest words appears in the vocabulary were calculated and shown on the table below. The input used are similar to the one used in testing MIMICK on its original paper, though it might be hard to see the correlation between the input and the nearest neighbors since it is highly dependant on the vocabulary or pretrained embedding and the vocabulary size used for training, thus reliance on dictionary and search engine were needed although it is not scientific.

Both model CNN and MIMICK results that were trained with word2vec are shown on Table 6.2 and Table 6.1 respectively. The first test word were an abbreviation "MCT" with length of three characters and all-capitalize. Both model were able to predict that the nearest neighbors are also another abbreviation. For name as input, "McNeally" and "Vercelloti", both model were able to predict the nearest neighbor to be name as well, as shown on Table 6.2 and Table 6.1. For "McNeally" the nearest neighbor were English name for male for both models. Interestingly for "Vercellotti", MIMICK predicted that the nearest neighbor was former American baseball player, while CNN predicted that the nearest neighbor was Ukrainian politician. For an adjective

input "Secretive", both model were predicted that the nearest neighbors are group of verbs, nouns, adjectives, and adverbs. Both models were also able to handle typography error like "corssing" and "developiong", only that CNN model nearest word were "developing" while MIMICK were a verb that has suffix *-ing*. The nearest neighbors for corssing were also verb with suffix *-ing*. Interestingly, for both model, nearest neighbor for flatfish has no correlation at all.

Similar nearest neighbors were also produced by the model when trained with polyglot (Al-Rfou et al., 2013) only with different set of words for each input.

Table 6.1: Nearest Neighbors Mimick (word2vec)

Word	**Nearest Neighbors**
MCT	EC ATI BT SMC Nvidia
McNeally	Miller Smith McKee Grimes Thompson
Vercellotti	Smoltz Pettitte Pujols Buehrle Peavy
Secretive	Seeks Approves Expands Implementation Preview
corssing	turning talking putting squeezing shifting
flatfish	just sort anyway things silly
compartmentalize	subjective appropriate constrained decentralized regard
pesky	just anyway maybe guess something
lawnmower	it liberalism something anyway kind
developiong	undermining implementing reshaping diminishing destroying
hurtling	squeezing ripping turning pulling chasing
expectedly	undermined unaware understood exposed utilized

Table 6.2: Nearest Neighbors CNN (word2vec)

Word	**Nearest Neighbors**
MCT	DPP PCT PMC TSMC RTA
McNeally	Murphy McCullough McIntyre Gallagher Delaney
Vercellotti	Yanukovych Tymoshenko Yushchenko Saakashvili Ancelotti
Secretive	Important Acquisition Process Benefits Transactions
corssing	putting turning squeezing pulling sneaking
flatfish	Wasps Premiership footballing Saracens flavorful
compartmentalize	efficiencies retrofit development commercialize integrate
pesky	weird cranky goofy scary joked
lawnmower	driveway sidewalk porch mother creek
developiong	developing development develop reshaping investment
hurtling	knocking chasing tearing ripping slamming
expectedly	predictably certainly unbelievably amazingly quite

6.2 POStagging results

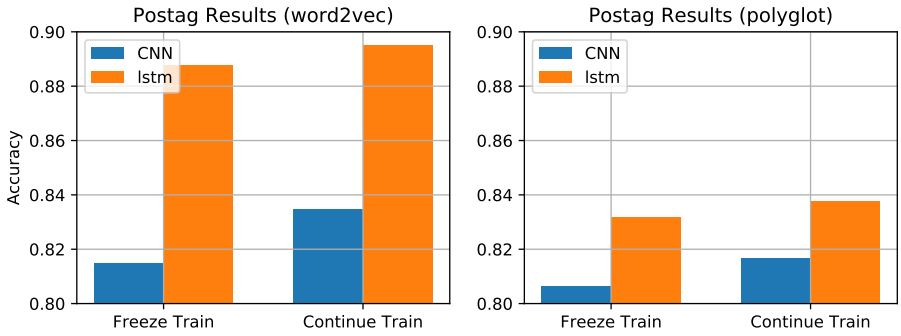

Figure 6.1: POStagging results

The OOV embedding from both model on top of the pretrained embedding were used as input for POStagging task. Firstly, the OOV model were frozen,

Table 6.3: Nearest Neighbors Mimick (polyglot)

Word	**Nearest Neighbors**
MCT	NAL MIB AWS SIA SMP
McNeally	McCready Hiatt Tolan McAdams Coxon
Vercellotti	Aurich Cavour Gubbio Barcelos Camoes
Secretive	Rhetorical Predictive Contextual Affective Perceptual
corssing	inflating straining concealing compromising channeling
flatfish	whirlpool cocoon diaper crevice gutter
compartmentalize	reproducible quantifiable repeatable synergistic biologic
pesky	waxy lozenge phosphor thermoplastic flake
lawnmower	dishwasher caddy welder motorist rowboat
developiong	compromising inflating loosening halting venting
hurtling	splashing blasting shredding combing pounding
expectedly	realistically energetically conspicuously materially imperfectly

meaning that the OOV model output were used as input as is by setting the weight to be frozen. Secondly, the OOV model were also trained in conjunction with the POStagger to improve accuracies of the POStagger. The improvement on both models and both pretrained embedding by allowing the OOV model to be trained during training the POStagger can be seen on Figure 6.1.

Both frozen training and continued training for CNN model has higher accuracies compared to Mimick. It shows that CNN model are better to handle POStagging than Mimick.

Table 6.4: Nearest Neighbors CNN (polyglot)

Word	**Nearest Neighbors**
MCT	NDS TEN GTC CPO UNI
McNeally	briefly quietly Akerman enthusiastically Coons
Vercellotti	Hassel Lemaire Sarno Perrot Necker
Secretive	Rhetorical Subjective Legitimate Contextual Constructive
corssing	confining straining inflating impacting shrinking
flatfish	narcotic transient hangover lameness stench
compartmentalize	commercialisation numeracy alertness institutionalization practicality
pesky	eyeballs jerky wrinkles fuss bruise
lawnmower	lavatory washroom toilet restroom mattress
developiong	distancing compromising orienting manoeuvring harmonizing
hurtling	compromising confining inflating lightening channeling
expectedly	substantively realistically sensibly procedurally irrevocably

6.3 Word Similarity

On Table 6.5, the model trained with word2vec (Mikolov et al., 2013b). The predicted embedding then used for calculating the spearman rank correlation. For easier reading, all the results of word similarity task shown in the table and the texts were multiplied by 1000. From 12 word similarity datasets, CNN model has higher Spearman correlation coefficient on 6 datasets with averaged Spearman correlation coefficient of 504.23 compared to MIMICK that achieved 501.97.

The same procedure for both models trained with polyglot (Al-Rfou et al.,

Table 6.5: Word Similarity Task Results (word2vec)

Dataset	**OOV**	**Invocab**	**OOV Ratio**	**CNN***	**Mimick***
card	989	317	75.73%	71.39	85.65
mc30	4	35	10.26%	714.29	731.64
men	82	669	10.92%	681.98	668.05
mturk287	95	404	19.04%	605.97	542.96
mturk771	17	1096	1.53%	656.73	654.57
rg65	2	46	4.17%	663.70	714.45
rwstanford	1488	1463	50.42%	301.63	291.60
simlex	17	1011	1.65%	429.11	429.57
simverb	90	737	10.88%	308.92	310.88
verb143	4	113	3.42%	485.09	483.57
wordsim	13	424	2.97%	647.94	648.74
yp130	5	142	3.40%	483.99	461.94
			average	504.23	501.97

* multiplied by 1000

2013). From 12 word similarity datasets, Mimick only has 5 datasets that has higher Spearman correlation coefficient than CNN model. In contrast with the model trained with word2vec (Mikolov et al., 2013b), the model trained with polyglot (Al-Rfou et al., 2013) only achived Spearman correlation coefficient as high as 339.55 and 331.30 for CNN and Mimick respectively as shown on Table 6.6.

For baseline embedding dict2vec (Tissier et al., 2017a). the original embedding added with random embedding for OOV handling were also compared with CNN and Mimick models. Dict2vec only able to achieve Spearman correlation coefficient of 519.22. In contrast CNN model an Mimick model were able to achieve 554.05 and 551.54 on average respectively as shown on Table 6.7. This shows that both OOV models can handle OOV better than initial-

Table 6.6: Word Similarity Task Results (polyglot)

Dataset	**OOV**	**Invocab**	**OOV Ratio**	**CNN***	**Mimick***
card	864	442	66.16%	125.61	78.16
mc30	1	38	2.56%	595.91	580.77
men	14	737	1.86%	486.48	487.51
mturk287	76	423	15.23%	457.89	443.08
mturk771	3	1110	0.27%	432.90	432.99
rg65	1	47	2.08%	530.47	525.82
rwstanford	999	1952	33.85%	298.92	260.79
simlex	4	1024	0.39%	234.81	234.86
simverb	53	774	6.41%	136.38	135.24
verb143	0	117	0.00%	335.81	335.81
wordsim	0	437	0.00%	410.28	408.60
yp130	5	142	3.40%	29.54	52.71
			average	339.58	331.36

* multiplied by 1000

izing embedding randomly for OOV. On top of that, from three pretrained embeddings, CNN model performs better than Mimick for word similarity task.

Table 6.7: Word Similarity Task Results (dict2vec)

Dataset	**OOV**	**Invocab**	**OOV Ratio**	**dict2vec***	**CNN***	**Mimick***
card	828	478	63.40%	48.07	112.75	74.61
mc30	0	39	0.00%	847.57	847.57	847.57
men	1	750	0.13%	713.16	723.76	719.79
mturk287	2	497	0.40%	652.27	651.04	652.05
mturk771	0	1113	0.00%	683.91	683.91	683.91
rg65	0	48	0.00%	832.86	832.86	832.86
rwstanford	619	2332	20.98%	214.60	424.97	426.77
simlex	3	1025	0.29%	454.80	457.11	458.45
simverb	24	803	2.90%	375.15	394.41	392.98
verb143	0	117	0.00%	187.82	187.82	187.82
wordsim	18	419	4.12%	642.71	713.81	710.63
yp130	2	145	1.36%	577.76	618.56	630.99
			average	519.22	554.05	551.54

* multiplied by 1000

Chapter 7

Conclusion

I was right all along.

7.1 What was I right about?

I was right about the following things.

7.1.1 Previous theories were wrong

People thought they understood, but they didn't.

7.1.2 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. 2, 6, 33, 39, 42, 43

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271. 8, 24

Baker, S., Reichart, R., and Korhonen, A. (2014). An unsupervised model for instance level subcategorization acquisition. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 278–289, Doha, Qatar. Association for Computational Linguistics. 34

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. 8

Bruni, E., Boleda, G., Baroni, M., and Tran, N.-K. (2012). Distributional semantics in technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 136–145, Jeju Island, Korea. Association for Computational Linguistics. 33

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. 8

Eisenstein, J., O'Connor, B., A. Smith, N., and P. Xing, E. (2012). Mapping the geographical diffusion of new words. *PLOS ONE*, 9. 3

Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., and Ruppin, E. (2001). Placing search in context: The concept revisited. pages 406–414. 34

Forrester, J. C. (2008). A brief overview of english as a language in change. *CCSE*, abs/1508.06615(4). 2

Garneau, N., Leboeuf, J., and Lamontagne, L. (2019). Predicting and interpreting embeddings for out of vocabulary words in downstream tasks. *CoRR*, abs/1903.00724. 3

Ghosh, S. and Kristensson, P. O. (2017). Neural networks for text correction and completion in keyboard decoding. *CoRR*, abs/1709.06429. 3

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. 10, 11, 12, 13, 14, 18

Grave, E., Joulin, A., Cissé, M., Grangier, D., and Jégou, H. (2016). Efficient softmax approximation for gpus. *CoRR*, abs/1609.04309. 30

Halawi, G., Dror, G., Gabrilovich, E., and Koren, Y. (2012). Large-scale learning of word relatedness with constraints. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 1406–1414, New York, NY, USA. ACM. 33

Hill, F., Reichart, R., and Korhonen, A. (2014). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *CoRR*, abs/1408.3456. 34

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 2, 20, 21, 22

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics. 26, 35

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2015). Character-aware neural language models. *CoRR*, abs/1508.06615. 7, 8, 9

Kulmizev, A., Blankers, B., Bjerva, J., Nissim, M., van Noord, G., Plank, B., and Wieling, M. (2017). The power of character n-grams in native language identification. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 382–389, Copenhagen, Denmark. Association for Computational Linguistics. 22, 23

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. 2

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR*, abs/1603.01360. 2

LeCun, Y. (1989). *Generalization and network design strategies*. Elsevier. 18

Li, Y. and Yang, T. (2017). *Word Embedding for Understanding Natural Language: A Survey*, volume 26. 1, 2

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. 2, 8

Liu, B. (2010). Sentiment analysis and subjectivity. In *Handbook of Natural Language Processing*. 3

Luong, T., Socher, R., and Manning, C. (2013). Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, Sofia, Bulgaria. Association for Computational Linguistics. 34

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. 6

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546. 2, 32, 42, 43

Miller, G. A. and Charles, W. G. (1991). Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28. 33

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*. 34

Pilehvar, M. T., Kartsaklis, D., Prokhorov, V., and Collier, N. (2018). Card-660: Cambridge Rare Word Dataset – a reliable benchmark for infrequent

word representation models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium. 33

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*. 3, 7, 8, 26, 29, 33, 35, 36

Radinsky, K., Agichtein, E., Gabrilovich, E., and Markovitch, S. (2011). A word at a time: Computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 337–346, New York, NY, USA. ACM. 33

Rocchesso, D. (1995). Sound processing. *Computer Music Journal*, 19. 1

Rubenstein, H. and Goodenough, J. B. (1965). Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633. 34

S. Harris, Z. (1954). Distributional structure. *Word*, 10:146–162. 2

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. 2, 33, 37, 43

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. 7

Tyagi, V. (2018). *Understanding Digital Image Processing*. CRC Press. 1

Yang, D. and Powers, D. (2006). Word similarity on the taxonomy of wordnet. 34

Appendix A

Code

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