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. Jens Lehmann

Dr. Giulio Napolitano

Institut für Informatik III

Rheinische Friedrich-Wilhelms-Universität Bonn

Declaration of Authorship

I, Yonathan Purbo Santosa, declare that this thesis, titled "OOV Handling by Learning Subword using CNN based N-grams", and the work presented in it is 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

I have received a myriad of support, advice, and assistance throughout this document writing. I would like to thank my supervisors Prof. Dr. Jens Lehmann and Dr. Giulio Napolitano for formulating this topic. I would also like to thank my Tutor Debanjan Chaudhuri for guiding with advice to finish this document.

I would like to thank my family and friends both here in Germany and back home in Indonesia for giving me ceaseless love, support, and advices throughout my study in Bonn University. You gave me great escape to rest my mind from my thesis.

In addition, would also like to thank Indonesia Endowment Fund for Education (LPDP) for giving me the opportunity to study abroad in Germany and covering my expenses and helping me to finish the Master program.

Contents

Abstract

Word embedding is word representation that is able to represent word in a dense vector instead of sparse vector. Word embedding normally consists of set of vocabulary and set of vector representation for each entry in vocabulary. The purpose of word embedding is to make text data able to be processed with mathematical function. In the training process of collecting vocabulary and its vector representation, some words might not exist in the training corpus. Thus, this word has no vector representation for downstream tasks. A model called Mimick was created to handle this problem by using bi-LSTM to predict embedding of words that are unavailable in the vocabulary (out-of-vocabulary or OOV). The problem with this implementation is that only the last hidden states were used for predicting the embedding. In the bi-LSTM architecture exists cell gate that controls the information flow. When this gate decided to forget previous sequence, then previous information will not be carried on to the latter sequence. Thus, another method is proposed to tackle this problem. CNN that mimics n-grams is used to process sequence data. By doing so, intermediate sequence of a word has higher chance of the information being carried on to predict the embedding of OOV then tested on downstream tasks such as POS-tagging and word similarity tasks that should improve performance in downstream tasks.

Keywords: CNN, CNNgrams, OOV, OOV handling, out-of-vocabulary, out-of-vocabulary handling

Chapter 1

Introduction

1.1 Motivation

Word embedding is a method for word representation mainly used in natural language processing domain. Unlike images and sounds data, text data represented differently in machine. Generally image is 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 channel 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 mathematical models either using analog or digital signals but not with text data (Li and Yang, 2017). Hence word embedding was introduced to give ability in representing text as a mathematical model namely a dense vector.

Text data consists of sequence of characters that is represented by codes that is standardized, for 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 depending 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 use in a sentence. Similar representation is also used to represent characters. The problem with one-hot vector is that it does not have any information that infers connection between one to another. 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 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 larger and more words connection can be inferred from the corpus (Kutuzov and Kunilovskaya, 2018). On top of that, multiple use 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 is another word that has the exact similar 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 increasing number of smartphones which uses touch screen, the number of typographical error is expected to increased as well (Ghosh and Kristensson, 2017). All these unavailable words in the corpus because of inability to collect such corpus or simply because of emergence of new slang or typographical error making the embedding of such word is unknown and it is 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 representing OOV, further improvement on downstream tasks should be able to be achieved by using machine learning method to infer OOV embeddings.

To infer OOV embeddings, the proposed model will be built over quasi-generative perspective. Only knowing the pre-trained vocabularies and its embedding, the embedding for OOV words will be generated by the model. Previous *state-of-the-art* used bi-directional long-short term memory (bi-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 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 direction. 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 or subsequences and they are not in sequence, meaning that there exist at least one character between two sub-words considered important by the model, hence the information on the last hidden state is incomplete since the previous information will be dampened or completely dropped by LSTM hidden neuron interiors if the next subsequence 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. A new OOV handling model
2. Analysis of the baseline model and the proposed model regarding on the OOV and the in-vocabulary embeddings
3. Evaluation on different settings for baseline model and the proposed model against downstream tasks

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 (Al-Rfou et al., 2013). This embedding was trained using 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). Similar with Polyglot, 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). Similar with previous word embedding, Dict2vec also has no OOV handling method and since the vocabularies were only inferred from Cambridge dictionary, there is no typographical error.

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 randomly generated embeddings each time an OOV emerges. This case fueled MIMICK, an OOV handling model to be created (Pinter et al., 2017). This model were able to tackle this problem successfully by using uses bidirectional long-short term memory (bi-LSTM) to process embedding of characters of an OOV word to produce the word 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). By doing so, the OOV embeddings are able to be predicted without the needs of knowing lexicon or model used for creating the word embedding. To prove that such OOV handling model can perform better than randomly generated embeddings or unique embedding for unknown token "$<UNK>$", several downstream tasks

were used to evaluate such as Part-of-Speech-Tagging (POS-tagging) and word similarity task that will be explained further below.

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.

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).

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).

Ioffe and Szegedy (2015) proposed a model to compensate slower training time caused by internal covariate shift. This phenomenon happened because of the distribution of values changes between layers in neural network, making training model that saturates to a nonlinearity activation function harder (Ioffe and Szegedy, 2015). By using a method called Batch Normalization, each layer's inputs were normalized to match distribution of certain mean and variance. As a result, larger learning rates can be used and smaller training epoch was needed to achieve a certain error values compared to one without Batch Normalization (Ioffe and Szegedy, 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 animal 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 neurons are responsible for the numerical representation of the input and the output. Given an input vector $\mathbf{x_i}$ with n-dimension from a dataset $D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_i}, y_i)\}$, 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 and so is different responses given different input stimulus with different weight values. The output o_i is calculated by using the dot product between $\mathbf{x_i}$ and $\mathbf{w}$ as shown on equation 3.1.

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

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

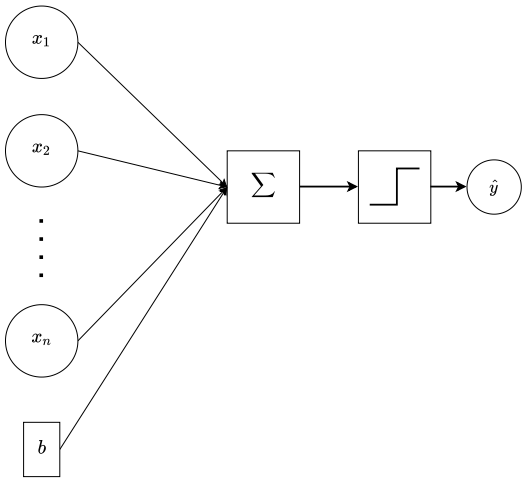

Figure 3.1: Perceptron

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 output as shown on equation 3.3.

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

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

The results then compared with the target 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 based on 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 $z = J(\mathbf{y})$,

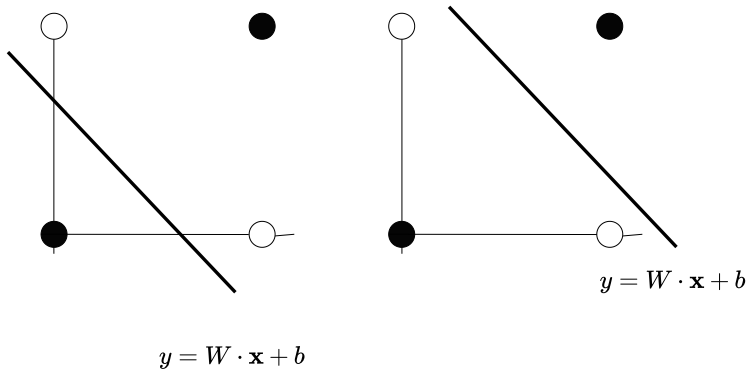

Figure 3.2: XOR Problem

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. This nonlinearity makes the model easier to train since those functions are differentiable.

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

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

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

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

3.2 Long-short Term Memory

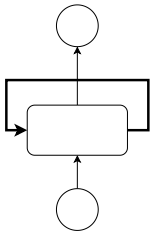

Figure 3.3: Recurrent Neural Network

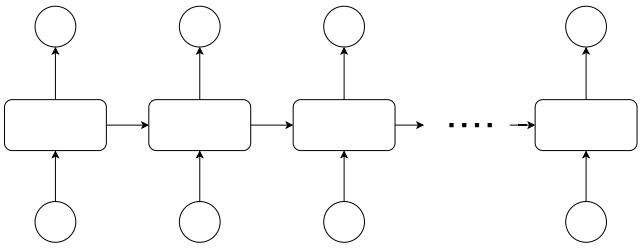

Figure 3.4: Unfolded Recurrent Neural Network

As previously mentioned, 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 of 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 it is multiplied several times sequentially over time and the gradient exploded if the weight is larger than one and it is 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 (Goodfellow et al., 2016). This was achieved by introducing self-loops inside the hidden layer that has time-scale mechanism that can change dynamically based on the input and previous state (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 calculation

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

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

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

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

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

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

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

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

The forget gate f_t and the input gate i_t interacts with input x_t, previous hidden state h_{t-1}, and previous cell gate C_{t-1} to control the current cell gate C_t. This cell gate C_t will be used to control the information flow from previous hidden state h_{t-1} to the current hidden state h_t as shown in equation 3.16. If $C_t = 0$, the hidden state h_t will be 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.

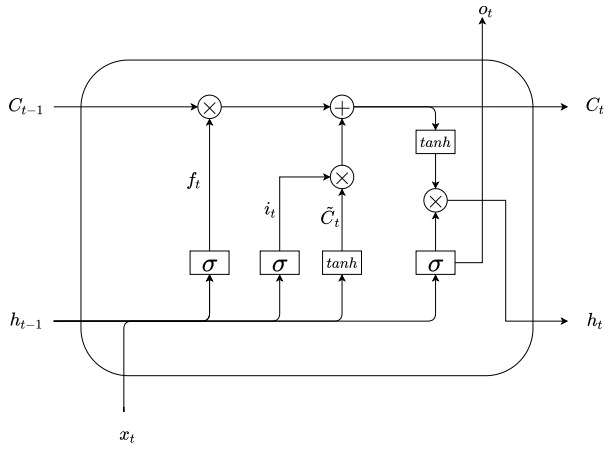

Figure 3.5: Gates inside LSTM hidden cell

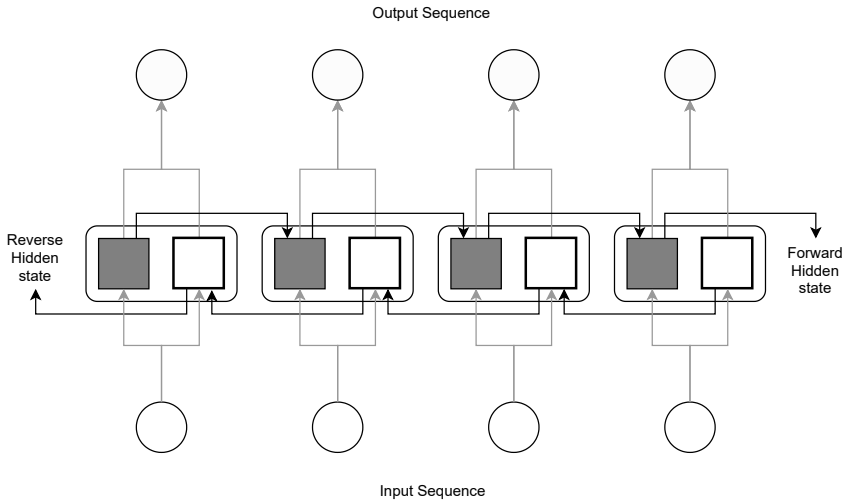

Figure 3.6: bi-LSTM with 4 sequence

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 $[g_1, g_2, \ldots, g_n]$.

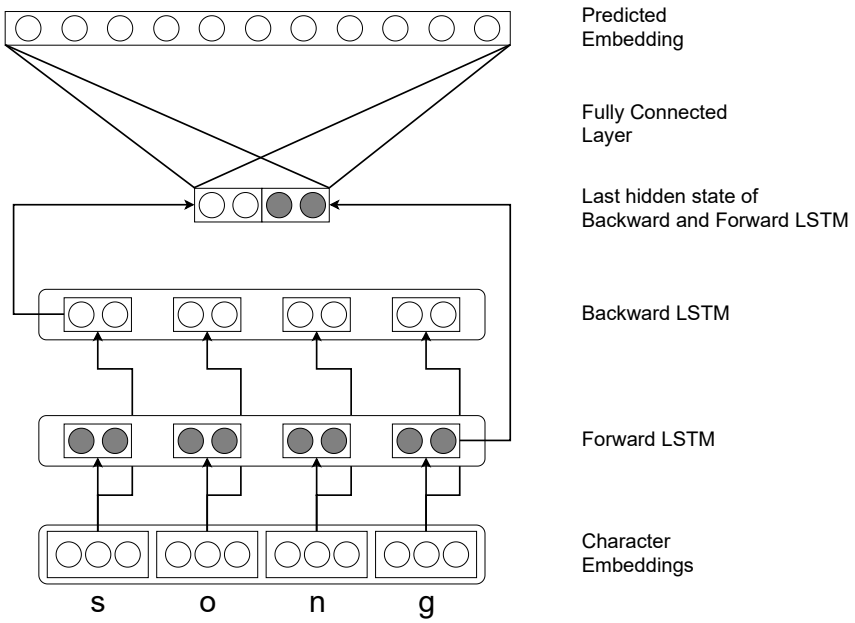

Figure 3.7: Mimick architecture

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 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 already indicated in the previous subchapter, 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 input vector $\hat{x}_t$ or hidden state $\hat{h}_{t-1}$ that could trigger the cell gate C_t to drop previous information entirely. Although bi-LSTM could serve the purpose to include the early sequence, there might exist intermediate sequence that appears in the middle of two important sequences that could be dropped because of the cell gate C_t decided to drop previous sequence for both forward and reverse LSTM. The solution to this is to increase the number of hidden states size to reduce such chance.

This problem fueled another approach to be introduced, namely using convolutional neural network (CNN) 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 element wise 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 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

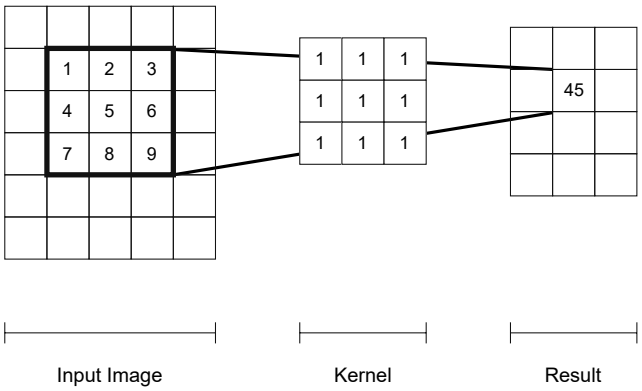

Figure 3.8: Convolution process of input image with kernel size 3×3

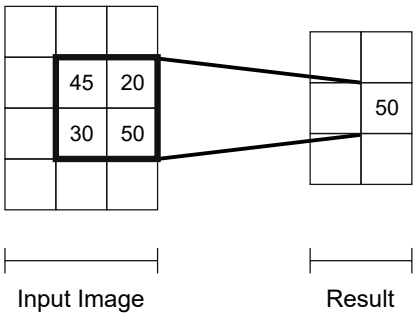

Figure 3.9: Maxpooling process of input image with window size 2×2

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 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,

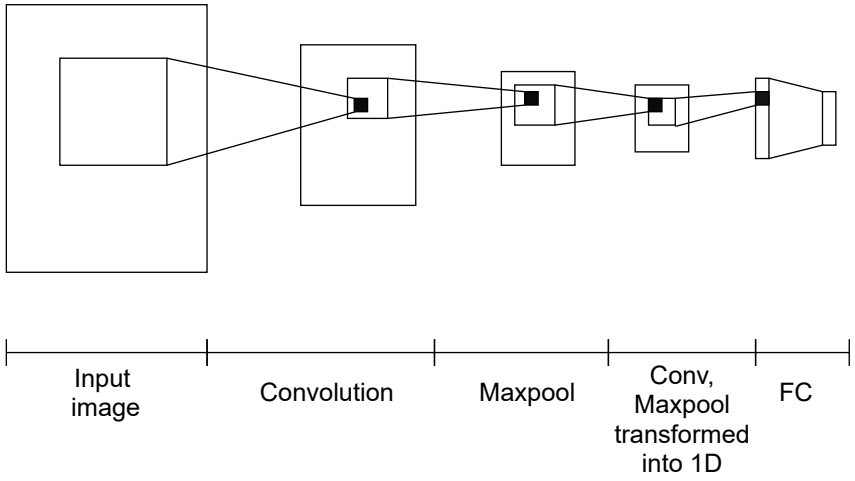

Figure 3.10: Example of CNN architecture

```
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 exists 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 is 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 was 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, the original vocabulary and its embedding were used for training the model to generate the embedding. In chapter Introduction, the 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 are 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

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, a method that accounts for the whole sequence yet still able to divides the whole sequence into subsequences is needed. Consequently, n-grams was chosen because this method splits word into sequence of characters depending 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

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

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

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

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

Figure 4.2: 4-grams examples

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 was 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 was 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 was applied to this vector by using ReLU activation,

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

Several kernel was used to learn several features for producing embeddings. Each of these kernel was responsible to find grams that were affecting the results, thus the vector $\hat{l_i}$ that is results of convolution $\{g\}^m * k_i$ will be maxpooled 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 were 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 fully connected layer consists of one hidden layer and one output layer with batch normalization before each layer to further sped up the training convergence according to Ioffe and Szegedy (2015). This normalization layer basically maintains that the distribution of each minibatch remains similar so that covariance shift is minimized in the

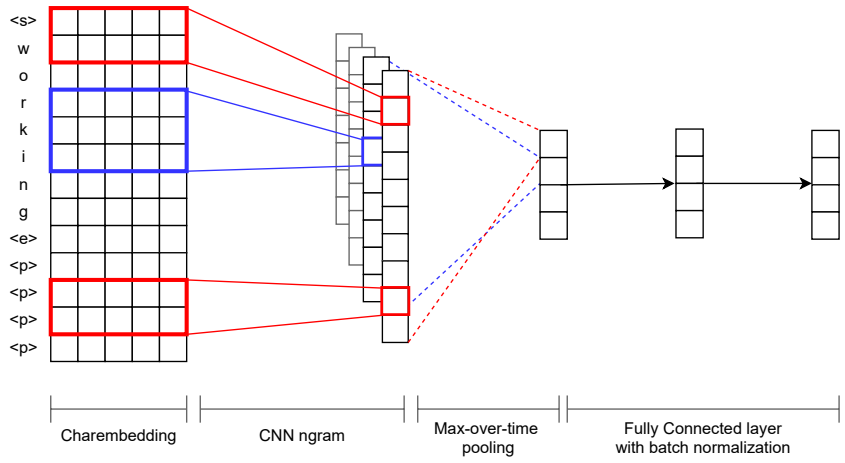

Figure 4.3: OOV Inferencing Model

training process (Ioffe and Szegedy, 2015). This process was done by applying transformation on each minibatch to a certain mean and variance (Ioffe and Szegedy, 2015). This mean and variance are learnable parameters. Let minibatch $\mathcal{B} = x_1, x_2, \ldots, x_m$, following operations were applied,

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{4.4}$$

$$\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} \left(x_i - \mu^2_{\mathcal{B}}\right) \tag{4.5}$$

$$\tilde{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \tag{4.6}$$

$$BN(x_i) = \gamma\tilde{x}_i + \beta \tag{4.7}$$

Equation 4.4 and equation 4.5 tries to find the mean and variance to normalize the minibatch by equation 4.6. After all of the data in the minibatch $\mathcal{B}$ were normalized, the data then scaled by parameter γ and shifted by parameter β as shown in equation 4.7. By making the parameters γ and β learnable, the model will adjust this process in the training process.

After all the process above was done, the generated embedding vector $\tilde{e}$ was produced. 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 were added at the beginning and the end of the word respectively.

Furthermore, padding token was added if the input word was 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 thus the result of maxpooling it will be 0. The reason for this is so that various 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 adjust all of the parameters for the neural network using mean squared error function,

$$Error = \frac{1}{2}\|e - \tilde{e}\|_2^2 \tag{4.8}$$

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

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 (Horsmann, 2018), 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 $w_i \in S$ was 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 $\tilde{e}_i$ was predicted using the OOV handling model, else the original embedding was 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,

$$embedding = mask \odot e_i + (1 - mask) \odot \tilde{e}_i \tag{4.9}$$

$$mask = \begin{cases} \vec{1} & \text{if } \ w_i \in \mathcal{V} \\ \vec{0} & \text{otherwise} \end{cases} \tag{4.10}$$

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 was passed through LogSoftmax activation function,

$$LogSoftmax(x_i) = log\left(\frac{exp(x_i)}{\sum_j exp(x_j)}\right) \tag{4.11}$$

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 were separated thus there were many chances that only frequent classes needed to be calculated before trying to calculate the infrequent classes. After calculating the loss, stochastic gradient descent was used to optimize the parameters of the model. The complete process of POS-tagging process is shown in figure 4.4. After training was done,

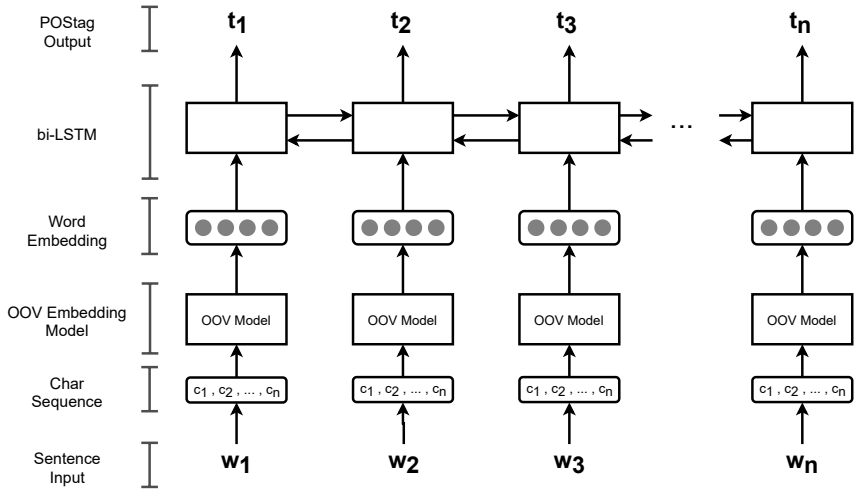

Figure 4.4: Pos-tagging Process

the accuracy of the POS-tagger based on different OOV handling model were compared.

4.2.2 Word Similarity Tasks

Word similarity task is basically a 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,

$$CosineSimilarity(e_1, e_2) = \frac{e_1 \cdot e_2}{\|e_1\|\|e_2\|} \tag{4.12}$$

After all of the cosine distance for all pairs were calculated, Spearman rank's correlation from the dataset and the generated embedding are calcu-

lated by using equation 4.13 and by using equation 4.14 when no tied ranks exists. The results of both MIMICK and the proposed model from several word similarity datasets then averaged and compared.

$$\rho = \frac{n\sum_{i=1}^{n} u_i v_i - \left(\sum_{i=1}^{n} u_i\right)\left(\sum_{i=1}^{n} v_i\right)}{\sqrt{\left[n\sum_{i=1}^{n} u_i^2 - \left(\sum_{i=1}^{n} u_i\right)^2\right]\left[n\sum_{i=1}^{n} v_i^2 - \left(\sum_{i=1}^{n} v_i\right)^2\right]}} \tag{4.13}$$

$$= 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)} \text{ where } d_i = u_i - v_i \tag{4.14}$$

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 randomly generated 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.

Pre-trained Word Embedding

In order to train the model, pre-trained embedding was needed since the model will tries to predict embedding from known vocabulary $v \in \mathcal{V}$ with its known embedding $e \in \mathcal{W}$. 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 words that frequently appears as a phrase, 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 was 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 pre-trained 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 pre-trained embedding was 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 datasets were used in order to increase the pair examples since for the datasets that are 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

[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`

et al., 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 shuffled and split up into train-val set with 80% and 20% quota 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

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

Table 5.1: OOV Handling Model Parameters

Hyperparameter	**Mimick**	**CNN**
Train-Val split	80%; 20%	
Batch size	64	
Epoch	100	
Momentum	0.5	
Learning Rate (η)	[0.01; 0.1]	0.1
Dropout	0	0.5
Num features	[50; 100; 200; 300]	[20; 50; 100; 150]

characters and starting token $<s>$ and ending token $<e>$ were added at the 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 $lr = 0.01$ for polyglot and $lr = 0.1$ for the rest with no dropout as dropout does not work well just as in the original implementation of 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 sets of n-grams sizes, for instance $n = [2, 3, 4]$ or $n = [5, 6, 7]$ can be used instead 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 two layers of fully connected network with batch normalization layer for each layer to produce the predicted word embedding. Batch normalization helps with the training process by making all of the minibatch data to have similar distribution for each layer (Ioffe and Szegedy, 2015). The error then calculated using Mean Squared Error as mentioned on equation 4.8 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). To find the optimal parameters such as learning rate, number of features/hidden neurons, and number of epoch, several number of tests with different number of parameters needed to be done as preliminaries. The summary of the hyperparameters are after the preliminary tests are 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 as well as the previous *state-of-the-art*. In this task, two kinds of evaluation methods were done. Before going into the method, there are several steps to prepare the datasets for training the POS-tagger. Firstly, the POS-tag dataset words were collected as sets $\mathcal{S}$. This was done in order to collect the vocabulary uniquely since in Python set elements are unique. Secondly, each elements in the set $\mathcal{S}$ was scanned whether it is in-vocabulary or out-of-vocabulary. If the element turns out to be OOV, this element which is a word, will be added into the vocabulary $\mathcal{V}$ and its embedding, either a zero vector, a randomly initiated value, or a generated embedding from the OOV model depending by the method used, will be added

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

to the original embedding $\mathcal{W}$. The dataset were split up into 80% for train set and 20% for test set.

The first method was to train the POS-tagger by using a pre-trained embedding. The original pre-trained embedding entries $\mathcal{W}$ were concatenated with the OOV model predicted embeddings $f(\text{OOV})$ so that each word $v \in \mathcal{V}$ has vector representation. In short, the new embedding $\mathcal{W}_{new} = \mathcal{W} \cup f(\text{OOV})$. The new pre-trained embedding $\mathcal{W}_{new}$ will not be trained in this method by setting the parameters to be fixed values.

The second method was to train the OOV model together with the POS-tagger. For this method, for each OOV entries added into the vocabulary $\mathcal{V}$, a zero vector representing the embedding of the OOV will be added to the embedding $\mathcal{W}_{new}$ that later on replaced with the generated embedding $f(\text{OOV})$ if the input word is OOV as explained in the equation 4.9 by masking the entries. Thus the OOV model will still be trained if the input turns out to be OOV while it is non-trainable if it is in-vocabulary input. Both of the models were used here then the accuracies were compared.

Lastly, it is similar to the second method but the pre-trained embedding was set to be a trainable parameters. Using similar method of masking, the gradient will flow separately depending on the embedding used, whether it is from the pre-trained embedding or from the embedding generated by the OOV model.

On top of different method used in this task, the sentence length was limited to 5 words. If the sentence length is less than 5 words then padding tokens were added at the end of the sentence to make the length becomes 5 words and if the sentence's length is more than 5 words, the slice of the sentence were randomly selected. Note that this padding token is different than the padding token of the character embedding. The padding token $<p>$ is a zero vector with dimension similar with the pre-trained embedding $\mathcal{W}$. The POS-tagger were trained with 100 epoch with 6 different seeds for the random number generation. The learning rate used was 0.1 with dropout for CNN model 0.5

Table 5.2: POS-Tagger Model Parameters

Hyperparameter	**Mimick**	**CNN**
Learning Rate (η)	0.1	
Batch size	64	
Epoch	100	
Momentum	0.5	
Training seed	[64; 20; 128; 0; 5; 10]	
Dropout	0	0.5

and no dropout for Mimick since it makes the model performs worse based on preliminary tests. The momentum used for the training was 0.5. In summary the parameters used are shown in table 5.2.

5.3.2 Word Similarity

Word similarity task is quite straight forward. Given pairs of words, if the word is an OOV in the pre-trained embedding, the word embeddings were predicted from OOV models then the cosine similarity of the word embeddings were calculated and compared between two models, else the original embedding from the pre-trained embedding was 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 from five tries. On top of that, other two pre-trained embedding used in this research were also used for comparing results. The results were then compared between two models.

Chapter 6

Results and Discussion

6.1 OOV handling model

After training the model was done, some OOV words were fed into the model and 5 nearest in-vocabulary words were calculated for sanity check and shown on the table below. The input used were similar to the one used in testing Mimick on its original paper for Polyglot embedding for English and few examples from the brown corpus that turned out to be OOV, but only inputs that produced interesting results are shown. Both model CNN and Mimick results for nearest neighbor that were trained with Polyglot are shown in Table 6.2 and Table 6.1 respectively. The number of features for Mimick and CNN was 100.

The first word was "hurtling", typographical error from the word hurting. For this word, both model seems to be able to locate that the nearest neighbor is another verb with suffix *-ing*. Interestingly for CNN, it relates this word with action than might induce pain thus hurting a subject. The second word is "expectedly". Both model were able to predict that the nearest neighbor is another word with suffix *-ly*. The third input word is "Actively". For this input, Mimick predicted that this word is a name which the first word is capitalize, thus the nearest neighbor is name. On the other hand, CNN predicted

that this is some word with suffix *-ly* and capital letter is used at the beginning. The next word is "corduroy", which is a pattern in textile. CNN model were able to predict that one of the nearest neighbor is "garment". The next input is "question-and-answer". This input constructed from 3 words connected with dash (-) symbol. CNN model predicted that the nearest neighbor were activities that the subjects do, for instance "hostess" and "recruiter". The last input is a number "1657". MIMICK predicted that the nearest neighbor is mathematical function in latex for example "\exp", while CNN predicted the nearest neighbor to be an abbreviation.

From those results, both models were able to predict the nearest neighbor that might be related to each other for some cases, but this results are highly dependent on the pre-trained embedding that was used to train the OOV handling model. If both model were trained using Word2vec, then for input word "hurtling" and "corssing" produced nearest neighbor "turning", "putting", "squeezing", "pushing", "talking", and "pulling" in different order sorted from the closest to the furthest.

Table 6.1: Nearest Neighbors Mimick (Polyglot)

Word	**Nearest Neighbors**
hurtling	inflating compromising concealing grasping channeling
expectedly	materially substantively sensibly energetically ethically
Actively	Harrold Wasson Wheatcroft Dever Covey
corduroy	heartbeat kink delicacy diaper damper
question-and-answer	barometer bottleneck ventilator slowdown spurt
1657	\exp \frac{n \frac{# sqrt xˆ{

Table 6.2: Nearest Neighbors CNN (Polyglot)

Word	**Nearest Neighbors**
hurtling	pulling whipping catching shaking burning
expectedly	frequently legitimately painfully inappropriately purposefully
Actively	Easily Potentially Entirely Merely Displaying
corduroy	basket garment wedge medium nutrient
question-and-answer	hostess dentist recruiter carer pastime
1657	GER BO INT SEP AGP

6.2 POS-tagging results

Before using the OOV handling model, random embedding for OOV entries were used to lay the baseline of the OOV handling method for POS-tagging. The OOV entries were added into the vocabulary list and their embedding were randomly initialized. This collection of embedding then trained in conjunction with the POS-tagger model. The results are shown in figure 6.1.

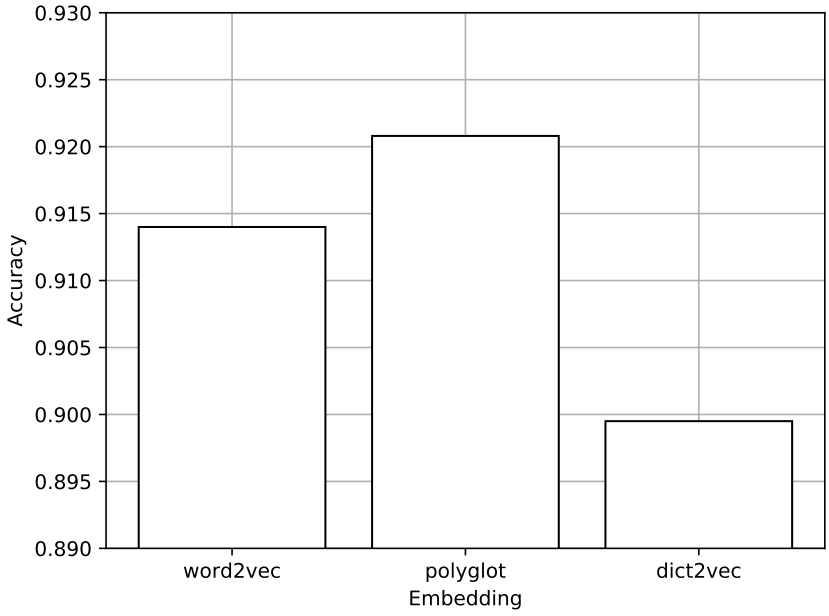

Figure 6.1: POS-tagging results with random OOV embedding

In order to test the best number of features parameter for each OOV handling model, the model and the embedding were set to be in evaluation mode or frozen, meaning that the parameters will not change when training the POS-tagger model. The OOV embedding from both model on top of the pre-trained embedding were used as input for POS-tagging task. Firstly, the embedding for OOV entries were predicted through the OOV handling model and added to the pre-trained word embedding. Then the collection of the pre-trained embedding and the predicted OOV embedding weights were fixed so it would not be changed in the training process. The results of different number of features on different pre-trained embeddings are shown in figure 6.2, figure 6.3, and figure 6.4. With this settings, only CNN with 20 number of features can outperforms the Mimick model in Word2vec word embedding while Mimick performs better in the other word embedding across different number of features used. Interestingly, this setting for bot models perform worse than randomly initialized OOV embeddings for Dict2vec pre-trained embedding.

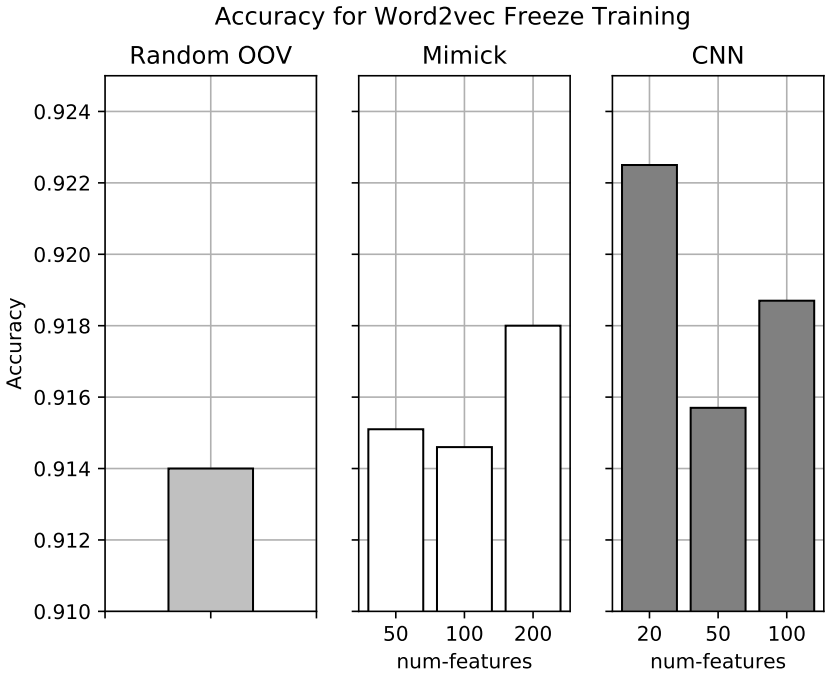

Figure 6.2: POS-tagging results frozen embeddings (Word2vec)

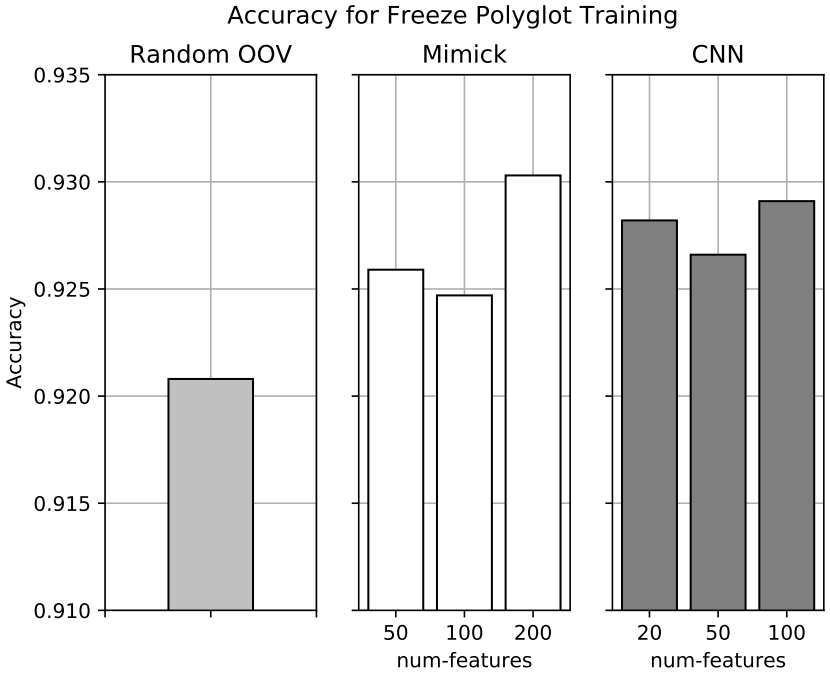

Figure 6.3: POS-tagging results frozen embeddings (Polyglot)

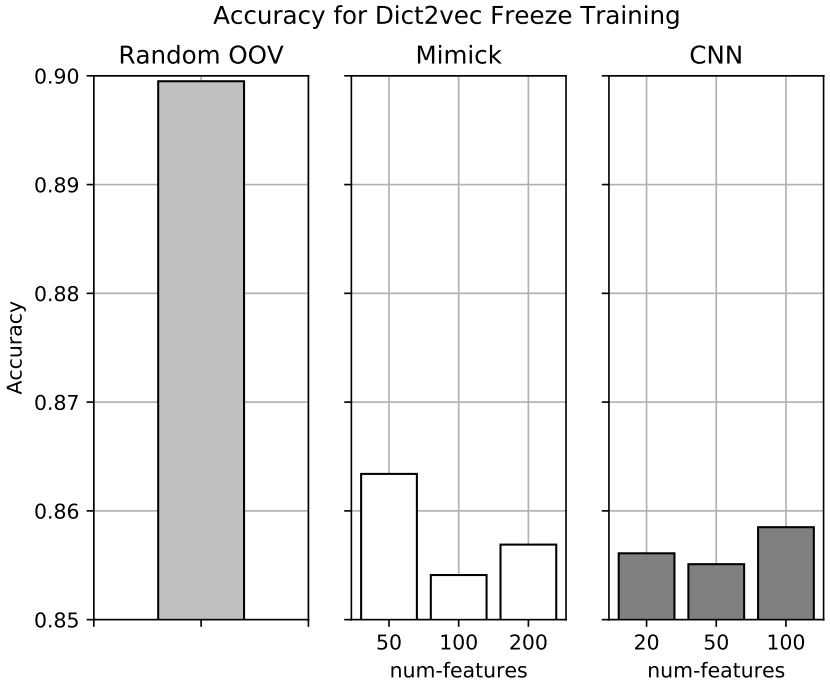

Figure 6.4: POS-tagging results frozen embeddings (Dict2vec)

Secondly, the OOV handling model were also trained in conjunction with

the POS-tagger to improve accuracies of the POS-tagger as well as to see whether further improvements in accuracies will be achieved by training both the OOV handling model and the POS-tagger. Note that the word embedding parameters were still untrainable in this setting. The results of these experiments were shown in figure 6.5, figure 6.6, and figure 6.7. The accuracy was increased for both OOV handling model as expected surpassing the randomly initialized OOV embeddings and the previous setting across all pre-trained embeddings. Surprisingly, by allowing the OOV handling model to be trained the accuracies of the POS-tagger model using CNN as the OOV handling model are surpassing the accuracies from POS-tagger that were using Mimick as the OOV handling model in all of the pre-trained embeddings used in this research.

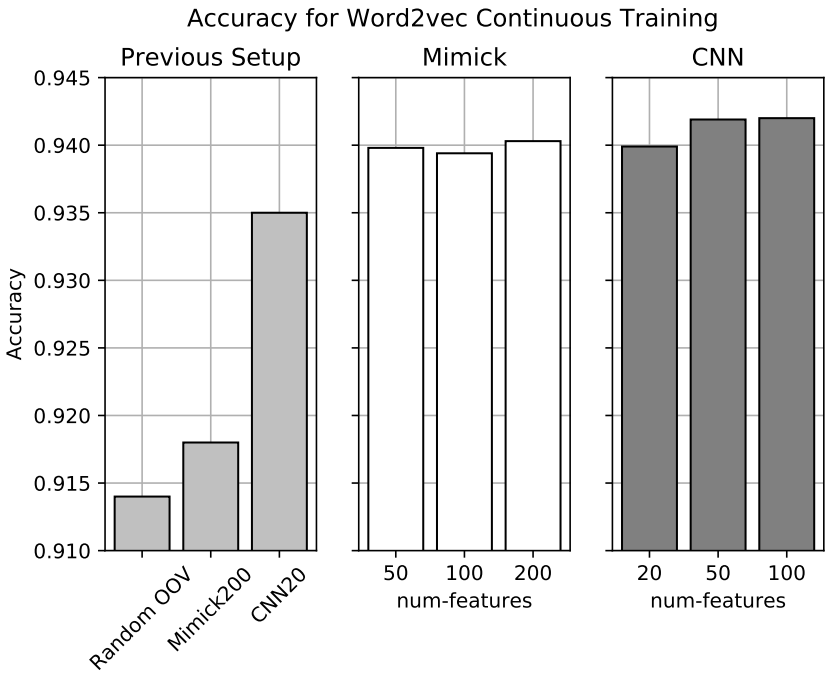

Figure 6.5: POS-tagging results continuous embeddings (Word2vec)

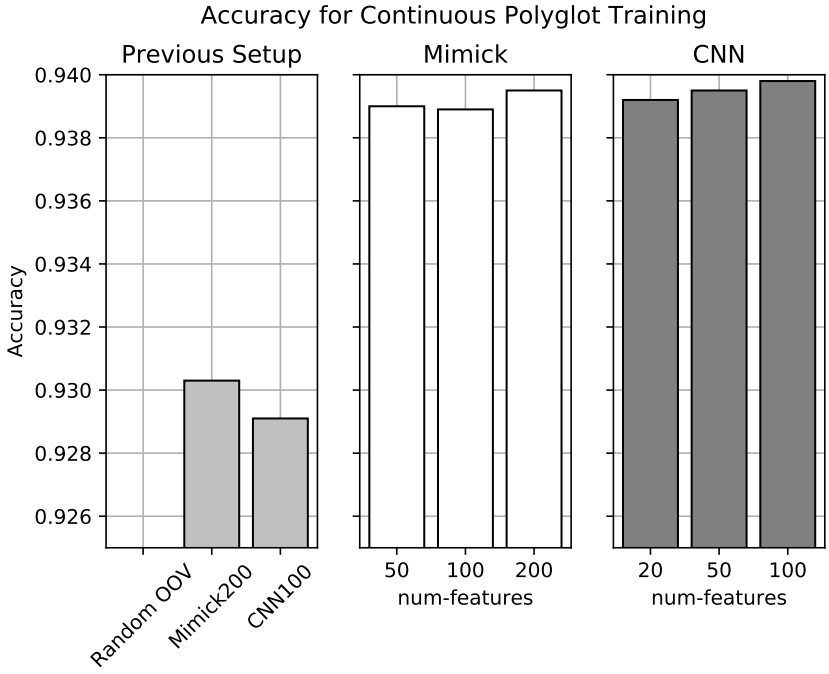

Figure 6.6: POS-tagging results continuous embeddings (Polyglot)

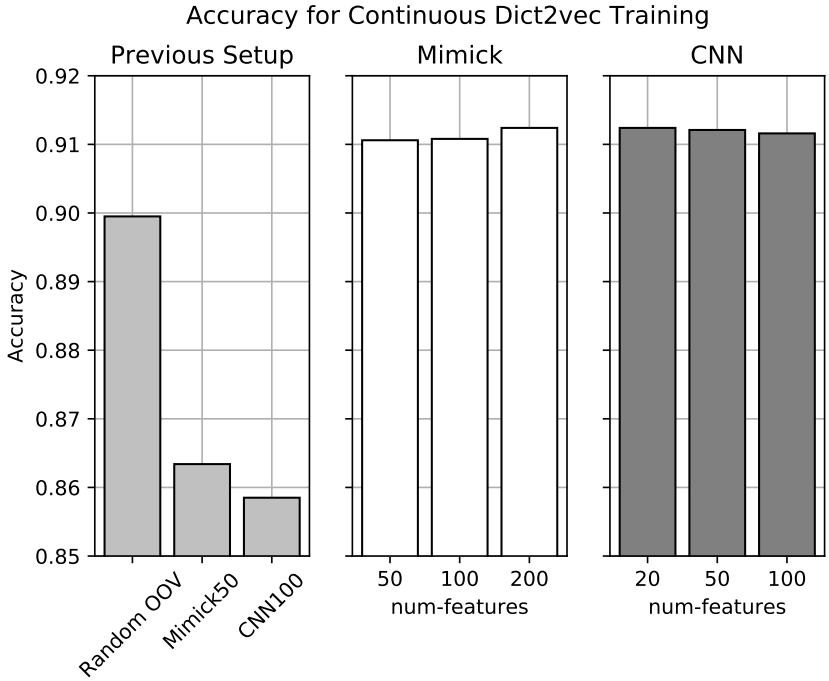

Figure 6.7: POS-tagging results continuous embeddings (Dict2vec)

Thirdly, to simulate how such model will be used in the downstream tasks

application, both the pre-trained embedding and the OOV handling model will be trained and the results will be compared to previously stated testing method. The number of features with the highest results from each OOV handling model for each word embedding were used. The results are shown in figure 6.8. For this setting, CNN model were able to outperform Mimick for Polyglot and Dict2vec embeddings while it perform worse for Word2vec embedding.

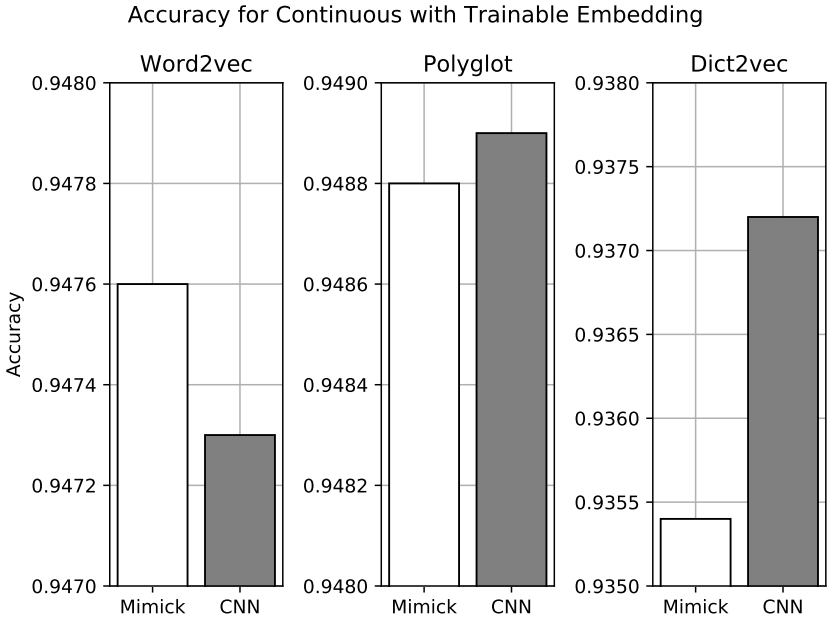

Figure 6.8: POS-tagging results for OOV handling model and pre-trained embeddings training

Given the results from various settings, in general CNN were able to achieve higher accuracies for Brown tag-set dataset. Even though Mimick is better at predicting OOV embeddings for downstream tasks right *out-of-the-box*, CNN still outperform Mimick when the OOV handling model was also included in the training. This proof that CNN are able to learn OOV handling better for POS-tagging compared to bi-LSTM that were used in Mimick.

6.3 Word Similarity

For word similarity task, the cosine distance between each given pairs from the dataset were calculated. Afterward, Spearman's rank correlation coefficient between the predicted embedding cosine distance and the score from the dataset were calculated.

Table 6.3: Word Similarity Task Results (word2vec)

Dataset	**OOV**	**Invocab**	**OOV Ratio**	**Mimick***	**CNN***
card	989	317	75.73%	**52.27**	27.25
mc30	4	35	10.26%	748.11	**758.79**
men	83	668	11.05%	**672.54**	664.64
mturk287	97	402	19.44%	518.15	**519.76**
mturk771	18	1095	1.62%	**657.93**	657.04
rg65	2	46	4.17%	709.77	**732.61**
rwstanford	1494	1457	50.63%	**233.83**	217.47
simlex	20	1008	1.95%	422.77	**425.90**
simverb	90	737	10.88%	**302.37**	292.91
verb143	4	113	3.42%	467.60	**478.25**
wordsim	15	422	3.43%	**658.92**	648.25
yp130	6	141	4.08%	**460.32**	443.22
			average	**492.05**	488.84

* multiplied by 1000

On Table 6.3, the OOV handling model trained with Word2vec (Mikolov et al., 2013b). The predicted embedding from the OOV handling model with highest accuracy from POS-tagging tasks were used for calculating the Spearman's rank correlation ρ. For easier reading, ρ values were multiplied by 1000. From 12 word similarity datasets, CNN model has higher Spearman's

rank correlation coefficient on 5 datasets from 12 datasets with averaged Spearman's rank correlation coefficient of 488.84 compared to MIMICK that achieved 492.05. The higher ρ values for each model and each dataset were marked with bold type setting in the table.

Table 6.4: Word Similarity Task Results (polyglot)

Dataset	**OOV**	**Invocab**	**OOV Ratio**	**Mimick***	**CNN***
card	864	442	66.16%	**128.93**	114.11
mc30	1	38	2.56%	605.25	605.25
men	14	737	1.86%	490.57	**492.17**
mturk287	76	423	15.23%	443.31	**458.81**
mturk771	3	1110	0.27%	**432.48**	432.20
rg65	1	47	2.08%	531.59	**524.77**
rwstanford	999	1952	33.85%	272.33	**290.78**
simlex	4	1024	0.39%	232.20	**234.16**
simverb	53	774	6.41%	**137.01**	134.42
verb143	0	117	0.00%	335.81	335.81
wordsim	0	437	0.00%	412.83	412.83
yp130	5	142	3.40%	**44.76**	44.62
			average	338.92	**339.99**

* multiplied by 1000

The same procedure was used for both models trained with polyglot (Al-Rfou et al., 2013). From 12 word similarity datasets, MIMICK has 4 datasets that has higher Spearman's rank correlation coefficient than CNN model and 5 datasets that is lower compared to CNN model while the other 2 has similar ρ value because of no OOV entries. In contrast with the model trained with word2vec, the model trained with polyglot (Al-Rfou et al., 2013) only achieved Spearman's rank correlation coefficient as high as 338.92 and 339.99 for MIMICK and CNN respectively, giving results that CNN has higher ρ value

compared to Mimick as shown on Table 6.4.

Table 6.5: Word Similarity Task Results (Dict2vec)

Dataset	**OOV**	**Invocab**	**OOV Ratio**	**Random***	**Mimick***	**CNN***
card	828	478	63.40%	48.07	80.89	**95.32**
mc30	0	39	0.00%	847.57	847.57	847.57
men	1	750	0.13%	713.16	723.63	**723.89**
mturk287	2	497	0.40%	652.27	**655.32**	653.13
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	**403.79**	400.27
simlex	3	1025	0.29%	454.80	**460.66**	459.87
simverb	24	803	2.90%	375.15	390.09	**393.39**
verb143	0	117	0.00%	187.82	187.82	187.82
wordsim	18	419	4.12%	642.71	718.72	**723.72**
yp130	2	145	1.36%	577.76	621.38	**621.75**
			average	519.22	550.55	**551.96**

* multiplied by 1000

For baseline embedding Dict2vec (Tissier et al., 2017a). The original embedding added with randomly generated embedding for OOV handling were tried five times and the one with highest results for each dataset was selected. The results then compared with CNN and Mimick models. Dict2vec only able to achieve Spearman's rank correlation coefficient of 519.22. In contrast to Mimick model and CNN model were able to achieve 550.55 and 551.96 on average respectively as shown on Table 6.5. This shows that both OOV handling models can handle OOV better than initializing embedding randomly for OOV. On top of that, for Dict2vec word embedding, CNN performs better than Mimick.

In summary, for word similarity tasks over 12 datasets and 3 pre-trained

word embeddings CNN performs better on Polyglot and Dict2vec while MIMICK performs better on word2vec in word similarities tasks. This further proved that CNN are better for handling OOV than MIMICK.

Chapter 7

Conclusion

Given some training examples of pretrained word embedding, machine learning could be used to generate embedding for *out-of-vocabulary* embedding which unavailable in the pretrained word embedding vocabulary collection. This method however, needs time and resources to be trained to generate a good approximation of the *out-of-vocabulary* embedding.

7.1 OOV Handling Model against Randomly Initialized OOV Embedding

For *out-of-vocabulary* embedding to be used in the downstream tasks, one can assign randomly generated embedding for each *out-of-vocabulary* or by assigning single token with one vector representation replacing all *out-of-vocabulary* entries. This approach has its advantages such as cheaper computation to generate embedding and has relatively high accuracies at around 90% for POS-tagging tasks. However, the training on the downstream tasks using randomly generated embedding might need more time or more sophisticated architecture to be able to keep up with the one that uses OOV handling model such as Mimick or the proposed model from this research. As evidence, with similar setup as the OOV handling model the randomly generated embedding for

out-of-vocabulary has lower accuracy for POS-tagging tasks. The randomly generated embedding for *out-of-vocabulary* entries method could produce POS-tagging results for three different pretrained embedding averaged around 90% and by using OOV handling model the accuracy increased to be around 93%.

7.2 CNN against LSTM for Extracting Text Features

As shown by the results of the downstream tasks, CNN model generally outperforms bi-LSTM implemented in Mimick. As mentioned before, Mimick uses the last state of the hidden state from the bi-LSTM architecture. This renders the intermediate sequence that could be important for predicting the embedding lost when the cell gate decided to drop previous information. By using CNN to extract text features, the intermediate sequence that appears somewhere in the middle of a word can still be inferred. With some generalization, the CNN features also able to use one kernel for several sequence that has certain similarities in terms of sequence of embeddings hence the higher accuracy and Spearman's rank correlation values.

7.2.1 POS-tagging

In POS-tagging tasks, CNN model accuracy is around 1% higher than Mimick. On top of the CNN architecture, some batch normalization were added to make the training converge faster. The only disadvantages of the CNN model is that it has more parameters to learn, meaning higher complexity and slower training time. On top of that, Mimick seems to perform better *out-of-the-box* compared to CNN. The advantages of the CNN model is that it perform better than Mimick when the OOV handling model is included in the training.

7.2.2 Word Similarity Task

Word similarity task was used to evaluate the distance between two words according to human subject. In this task, the word generated embedding expected to have similar degree of similarities between two embeddings from two words. In this task, the CNN model were able to achieve higher ρ value than Mimick. Even though this task is highly dependent on the embedding used for predicting the embedding of given word since every embedding is trained on different corpus that is trained with different purposes, CNN still able to performs better.

7.3 Future Works

Both models Mimick and CNN were able to produce embeddings for *out-of-vocabulary* words, but both model were not really able to define a function that could generate word embedding mimicking existing pretrained embedding since updating pretrained embedding with new dataset will requires retraining the embedding. Instead of retraining the embedding, finding function or model that are able to mimick the pretrained embedding will save computation time at least until the language itself is changing into something completely different making the word embedding becomes obsolete. On top of that, if the function generating the in-vocabulary embedding, this could save space since pretrained embedding storage size could be large.

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. Cited on 2, 6, 34, 49

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. Cited on 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. Cited on 35

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. Cited on 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. Cited on 34

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. Cited on 8

Eisenstein, J., O'Connor, B., A. Smith, N., and P. Xing, E. (2012). Mapping the geographical diffusion of new words. *PLOS ONE*, 9. Cited on 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. Cited on 35

Forrester, J. C. (2008). A brief overview of english as a language in change. *CCSE*, abs/1508.06615(4). Cited on 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. Cited on 3

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

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. Cited on 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. Cited on 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. Cited on 34

Hill, F., Reichart, R., and Korhonen, A. (2014). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *CoRR*, abs/1408.3456. Cited on 35

Horsmann, T. (2018). *Robust Part-of-Speech Tagging of Social Media Text*. PhD thesis, Universität Duisburg-Essen. Cited on 30

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167. Cited on 9, 27, 28, 37

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. Cited on 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. Cited on 25, 26, 36

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2015). Character-aware neural language models. *CoRR*, abs/1508.06615. Cited on 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. Cited on 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. Cited on 2

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

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

Li, Y. and Yang, T. (2017). *Word Embedding for Understanding Natural Language: A Survey*, volume 26. Cited on 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. Cited on 2, 8

Liu, B. (2010). Sentiment analysis and subjectivity. In *Handbook of Natural Language Processing*. Cited on 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. Cited on 35

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*. Cited on 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. Cited on 2, 33, 48

Miller, G. A. and Charles, W. G. (1991). Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28. Cited on 34

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*. Cited on 35

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. Cited on 34

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*. Cited on 3, 7, 26, 29, 34, 36, 37

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. Cited on 34

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

Rubenstein, H. and Goodenough, J. B. (1965). Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633. Cited on 35

S. Harris, Z. (1954). Distributional structure. *Word*, 10:146–162. Cited on 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. Cited on 2, 34, 39, 50

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. Cited on 7

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

Yang, D. and Powers, D. (2006). Word similarity on the taxonomy of wordnet. Cited on 35

Appendix

Code

The implementation can be accessed in `https://github.com/yonathansantosa/oov_handling`

Training Screenshot

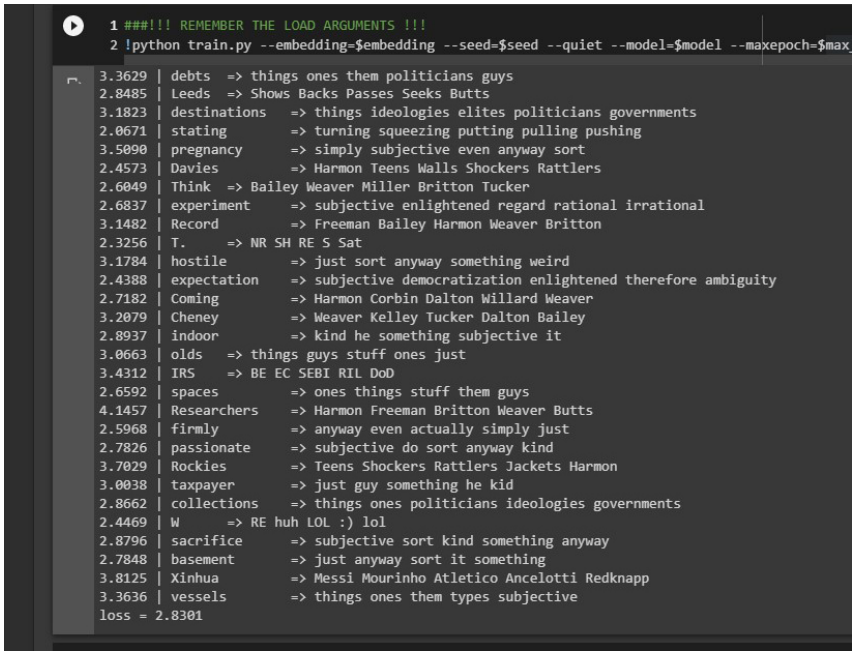

Figure: OOV Handling Model Training

```
[ ]  1 lang = 'en'
     2 model = 'lstm'
     3 loss_fn = 'mse'
     4 asc = 'True'
     5 embedding = 'word2vec'
     6 run = 1
     7 mtp = 1
     8 nn = 10
     9 training_seed = 128
    10 initial_epoch = 0
    11 max_epoch = 100
    12 num_features = 300
    13 # run = 2
    14 # epoch = 2000
```

```
[ ]  1 for seed in [64,20,128,0,5,10]:
     2     print(seed)
     3     path = 'trained_model_%s_%s_%s_postag' % (lang, model, embedding)
     4
     5     %rm -r $path
     6     %mkdir $path
     7
     8     # set paths
     9     ROOT = %pwd
    10     LOG_DIR = './%s/logs/' % (path)
    11
    12     print(LOG_DIR)
    13
    14     !python train_postag.py --train_embed --trained_seed=128 --seed=seed — —embeddir

⇨ 64
```

Figure: POS-tagger Training

```
[ ]  1 lang = 'en'
     2 model = 'lstm'
     3 loss_fn = 'mse'
     4 asc = 'True'
     5 embedding = 'word2vec'
     6 run = 1
     7 mtp = 1
     8 nn = 10
     9 training_seed = 128
    10 initial_epoch = 0
    11 max_epoch = 100
    12 num_features = 200
    13 # run = 2
    14 # epoch = 2000
```

```
[ ]  1 !python word_similarity.py --embedding=$embedding --model=$model -
```

```
⇨ dataset,oov,invocab,rho,p
card,989,317,0.05227463,0.17981301
mc30,4,35,0.74810861,0.00000201
men,83,668,0.67254152,0.00000000
mturk287,97,402,0.51815263,0.00000000
mturk771,18,1095,0.65792941,0.00000000
rg65,2,46,0.70977260,0.00000000
rwstanford,1494,1457,0.23383194,0.00000000
simlex,20,1008,0.42277191,0.00000000
simverb,90,737,0.30237219,0.00000000
verb143,4,113,0.46760400,0.00000000
wordsim,15,422,0.65891927,0.00000000
yp130,6,141,0.46032089,0.00000004
```

Figure: Calculating Spearman's Rank Correlation Coefficient for Word Similarity Task