

OOV Handling by Learning Subword using CNN based N-grams

Yonathan Purbo Santosa

Matriculation Number: 2993106

October 2019

A thesis submitted for the degree of Master of Science

Institute of Computer Science

Supervisors:

Prof. Dr. Jens Lehmann

Dr. Giulio Napolitano

INSTITUT FÜR INFORMATIK

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, I 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

1	Introduction	1
1.1	Motivation	1
1.2	Objectives	4
1.3	Contributions	4
1.4	Thesis Structure	5
2	Related Work	6
3	Preliminaries	10
3.1	Feedforward Neural Network	10
3.2	Long-short Term Memory	13
3.3	Mimick	16
3.4	Convolutional Neural Network	17
3.5	N-grams	19
4	Method	23
4.1	Out-of-Vocabulary Model	23
4.1.1	Sequence Feature Extraction	23
4.1.2	Embedding Generation	26
4.1.3	Error and Backpropagation	27
4.2	Measuring Performance on Downstream Tasks	28
4.2.1	Part-of-Speech Tagging	28
4.2.2	Word Similarity Tasks	29
5	Implementation	31
5.1	Preparation	31
5.1.1	Dataset Preparation	31

5.1.2	Programming Language and Tools	33
5.1.3	Hardware	33
5.2	Training	33
5.2.1	Training OOV model	33
5.3	Evaluating with Downstream Tasks	35
5.3.1	Part-of-Speech Tagging	35
5.3.2	Word Similarity	36
6	Results and Discussion	38
6.1	OOV handling model	38
6.2	POS-tagging results	40
6.2.1	Randomly Generated OOV Embedding as Baseline . . .	40
6.2.2	Non-trainable OOV Handling Model	40
6.2.3	Trainable OOV Handling Model	42
6.2.4	OOV Handling Model and Pre-Trained Embedding Joint Training	45
6.3	Word Similarity	46
6.4	Discussion	48
7	Conclusion and Future Works	50
7.1	OOV Handling Model against Randomly Initialized OOV Em- bedding	50
7.2	CNN against bi-LSTM for Extracting Text Features	51
7.3	CNN against bi-LSTM for Downstream Tasks	51
7.3.1	POS-tagging	51
7.3.2	Word Similarity Task	52
7.4	Future Works	52
	Bibliography	53

Abstract

Word embedding is numerical representation that is able to represent word in a dense vector instead of sparse vector. Word embedding normally consists of a set of vocabulary and a set of vector representation for each entry in vocabulary. The purpose of word embedding is to enable text data 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 was used to process sequence data. By doing so, intermediate sequence of a word has a higher chance being carried on to predict the embedding of OOV then tested on downstream tasks such as POS-tagging and word similarity tasks which should improve performance in downstream tasks consequently.

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

Chapter 1

Introduction

1.1 Motivation

Word embedding is a numerical form for word representation which is mainly used in natural language processing domain. Unlike images and sounds data, text data is represented differently in machine. Generally, image is represented as two-dimensional to four-dimensional matrix (given channels and alpha value) with finite number of cell elements containing numerical value to represent color intensity on each location (Tyagi, 2018). For example, RGB image is represented with 3-dimensional matrix. Each channel represents the intensity of red color, green color, and blue color respectively in the form of 2-dimensional matrix. On the other hand, sound is represented as one-dimensional signal or a stack of those signals (given several channels, it becomes two-dimensional) that represents 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 by using analog or digital signals but not with text data (Li and Yang, 2017). Hence word embedding was proposed to give the ability to represent text as a mathematical model namely a dense vector.

Text data consists of a sequence of characters that is represented by standardized codes, for example ASCII. In ASCII each character is represented by a number from 0 to 127 that later on extended until 255. Combinations of these number are translated by computer to represent a character. Originally, text data can only be modeled by using one-hot vector in natural language pro-

cessing. This vector is a one-dimensional vector that has d -dimension, given d words that are existed 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 is stacked with another one-hot vectors to represent the order of use in a sentence. Similar representation is also used to represent characters sequence. The problem with one-hot vector is the lack of any information that infers connection between one to another. It only encodes that a certain word is used in a certain sentence in a certain sequence. Instead of a sparse representation for each word, dense vector representation 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 usages 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 connection between each word can be inferred from the corpus (Kutuzov and Kunilovskaya, 2018). On top of that, multiple usage of a vocabulary might also be used as a training data as an example 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). Furthermore, there are cases of typographical error, especially on social media platform where anybody willingly write a text over these platforms (Liu, 2010) and some of these words may not be exist in the corpus hence not included in the vocabulary even though there is another word which has the exact similar meaning to those words thus it should have more or less a close distance to the standard or original word (Eisenstein et al., 2012). In addition, with the increasing number of smartphones which uses touch screen to type,

the number of typographical error is expected to be increased as well (Ghosh and Kristensson, 2017). All these unavailable words in the corpus because of the inability to collect such corpus or simply because of the emergence of the new slang or the typographical error which makes the embedding of such word unknown is called as *out-of-vocabulary* (OOV) words. One may use a simple approach by assigning a 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 expected to be generalized in training. Despite the fact that it can be used to represents OOV, a further improvement on downstream tasks should be able to be achieved by using machine learning method to infer OOV embeddings.

To infer the OOV embeddings, the proposed model would be built over quasi-generative perspective. Only by knowing the pre-trained vocabularies and its embedding, the model would generate embedding for OOV words. Previous *state-of-the-art* used bi-directional long-short term memory (bi-LSTM) to infer OOV embeddings which 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 was used to transform sequence of characters in a word into sequence of embedding. The sequence of the character embeddings was forwarded into bi-LSTM then to fully connected layer. In a language model, bi-LSTM generally works by separating sub-word by remembering and forgetting previous sequence from both direction.

In the implementation of MIMICK, the last hidden state of the bi-LSTM was used to predict its embedding. By the architecture of LSTM, the gates inside the hidden neuron might drop the previous information if the input triggers the cell gate to do so. Hence a problem might arise when there are more than two important sub-words or subsequences which are not in sequence, meaning that there exist at least one character that could trigger the cell gate to drop previous information between two sub-words that considered to be important by the model. Because of that, the information in 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 the word is considered to be 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 is able to generate embedding for the correspondence word has to be created. In MIMICK, the whole sequence was 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 by 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 give better results in downstream tasks since the information that is processed is complete and only left for the model to pick which n-grams features are more important. What is left is for the model is to pick which grams that should be used in the model which is impossible to do since there are many word combinations which makes the model needs to accept a huge number of inputs. Instead of handpicking the features of n-grams, convolutional neural network (CNN) can be used to learn the existing features and pick which features need to be considered important by using character embedding and treat the character sequence embedding as two-dimensional matrix. CNN has kernels that define which feature is needed by training the model. On top of that, the features that are closely similar to the needed feature will still be able to produce high response. The features that are picked will then be processed to predict the word embedding for the input word by using feedforward network.

1.3 Contributions

1. A new OOV handling model that used CNN for subword learning
2. Evaluation on OOV handling by using random embedding or unknown token “<UNK>” against machine learning method for OOV handling
3. Analysis of the baseline model and the proposed model regarding on the OOV and the in-vocabulary embeddings

4. 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 the research done in these 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 the solution proposal to the problem and the testing method for the results analysis are explained. In the Implementation chapter, the method of solution proposed to the problem are described in a technical way to show how it is implemented and tested. In the Results and Discussion chapter, the results are shown and discussed to the relation to the previous *state-of-the-art*. Lastly, Conclusion and Future Works chapter talks about the conclusion that are able to be pulled from this research.

Chapter 2

Related Work

Polyglot is one of word embeddings that is 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 with 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 a word that is not used in Wikipedia. This pre-trained embedding was also used in the baseline model MIMICK for generating OOV embedding in many languages.

Word2vec is another word embedding that was trained using skip-gram model (Mikolov et al., 2013a). The available language choice for this pre-trained 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 $w(t+2)$ are 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. The more examples and vocabularies a corpus has, the better the representation of the embeddings will be since more information will be able to be retrieved. Word2vec model has no OOV handling either, meaning either random vector or unknown “ $<UNK>$ ” embedding will be used for it.

Dict2vec is yet another embedding that was trained by looking up defini-

tions of words from Cambridge dictionary (Tissier et al., 2017b). This embedding was created because the previous method is trained with an unsupervised manner, which means that there is no supervision between pairs of words. There might exist a pair of words that are highly related but do not appear enough inside a corpus which makes it harder for the model to find a connection (Tissier et al., 2017b). Thus, this model was 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 was evaluated by 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 other than by assigning some unknown token “<UNK>” or by letting 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 was able to tackle this problem successfully by using bi-directional 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 was taken from quasi-generative perspective, which means that the original embedding was assumed to have some form that could generate the embeddings (Pinter et al., 2017). By doing so, the OOV embeddings were 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 could perform better than randomly generated embeddings or unique embedding for unknown token “<UNK>”, several downstream tasks were used to evaluate the model, 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 a given word in a certain sentence. In English, there are words that have ambiguous grammatical category, such as word “tag” can either be noun or verb depending on the usage of it (Cutting et al., 1992). To tackle this problem, many researchers proposed to use of 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 would be implemented to serve as the downstream task. This model was a bi-LSTM that took sequence of word embeddings to represent a sentence or parts of sentence then each word embedding were categorized for its tag based on the usage in that sentence.

In spite of the performance of MIMICK, there are evidence of CNN that is used for sequence modeling can outperform LSTM and RNN architecture (Bai et al., 2018). Moreover, CNN converges 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 by 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 is 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 original embedding or ASCII code. 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 with 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 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 the 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 (MLP) are a mathematical model that is inspired by how neuron works in biological body (Goodfellow et al., 2016). The model takes numerical input as a stimulus and produces numerical output as a 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 $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_i, y_i)\}$, 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 produce different responses given similar input stimulus and so are the different responses given different input stimulus with similar weight values. The output o_i is calculated by using the dot product between \mathbf{x}_i and \mathbf{w} as follows,

$$o_i = \mathbf{x}_i \cdot \mathbf{w} \quad (3.1)$$

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

After calculating the output o_i and adding another parameter bias b , the result is passed to activation function $f(o_i)$ to produce the model output \hat{y}_i as follows,

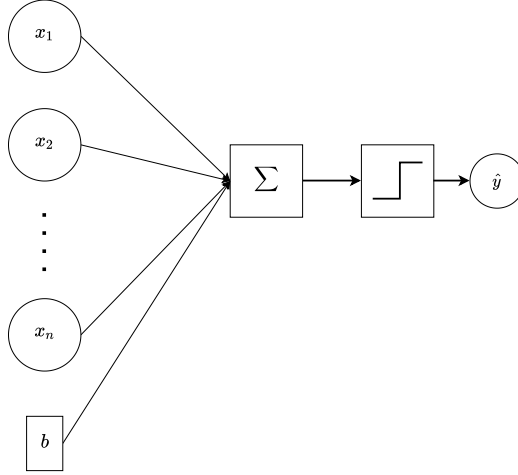


Figure 3.1: Perceptron

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

This model is called perceptron and it is depicted in figure 3.1. The input information is 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 are 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 are producing the minimum cost. If $\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}) = \mathbf{x} \cdot \mathbf{W}$ and the cost $z = J(\mathbf{y})$, the simple calculation of the gradient can be calculated as follows,

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \quad (3.4)$$

$$= \sum_j \frac{\partial z}{\partial y_j} \sum_k w_k \quad (3.5)$$

The correction on parameter \mathbf{W} is calculated with a learning parameter η as

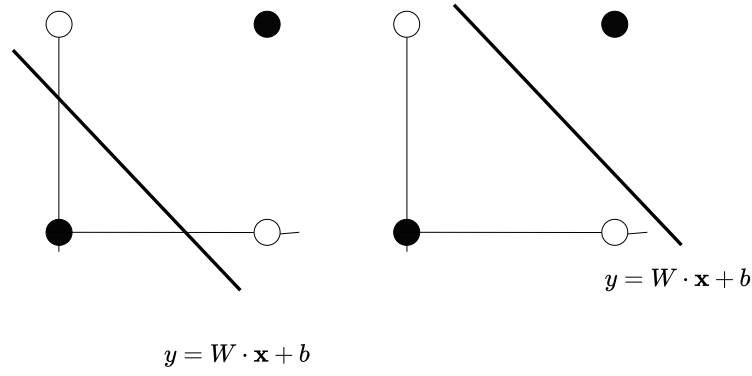


Figure 3.2: XOR Problem

shown in the following equation,

$$\hat{w}_i = w_i - \frac{\partial z}{\partial x_i} \times \eta \quad (3.6)$$

Notice that the gradient in equation 3.6 is multiplied by -1 because the gradient points uphill and in order to find the minimum cost the parameters need 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 a model which consists of a single layer of input and a single layer of output is not enough to solve XOR problem (Goodfellow et al., 2016), because the perceptron only separates the space with a hyperplane, or in XOR problem a hyperline as shown on figure 3.2. Thus multilayer perceptron was introduced. This model consists of a single layer of input, a 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 which are described in equation 3.7 until 3.10. This nonlinearity makes the model easier to train since those functions are differentiable.

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

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

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

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

3.2 Long-short Term Memory

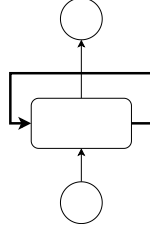


Figure 3.3: Recurrent Neural Network

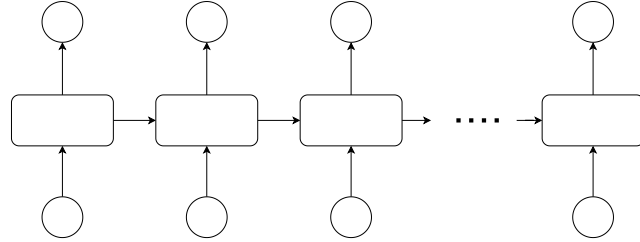


Figure 3.4: Unfolded Recurrent Neural Network

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 is used for processing sequential data (Goodfellow et al., 2016). For the given sequence of input with length of t , $\mathbf{x}_i^{(t)} = x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(\tau)}$, 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 the model is tasked to extract information from both sentences and asked when did the

narrator went to Paris, 2004 would be the relevant information regardless of when it appears in a sentence (Goodfellow et al., 2016). If such task is trained by 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 the weight is near zero and it is multiplied several times sequentially over time or 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 a long sequence (Goodfellow et al., 2016). This was achieved by introducing self-loops inside the hidden unit 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 the 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,

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

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

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

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

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

$$h_t = o_t \times \tanh(C_t) \quad (3.16)$$

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

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 the 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 that the previous information

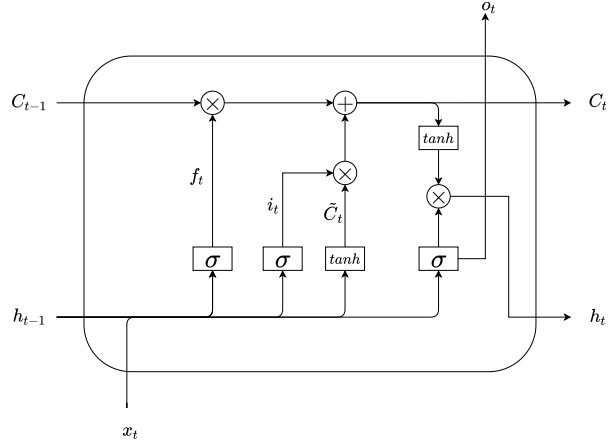


Figure 3.5: Gates inside LSTM hidden cell

of a certain feature is dropped for the gated hidden states. The LSTM hidden cell's architecture is depicted in figure 3.5.

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

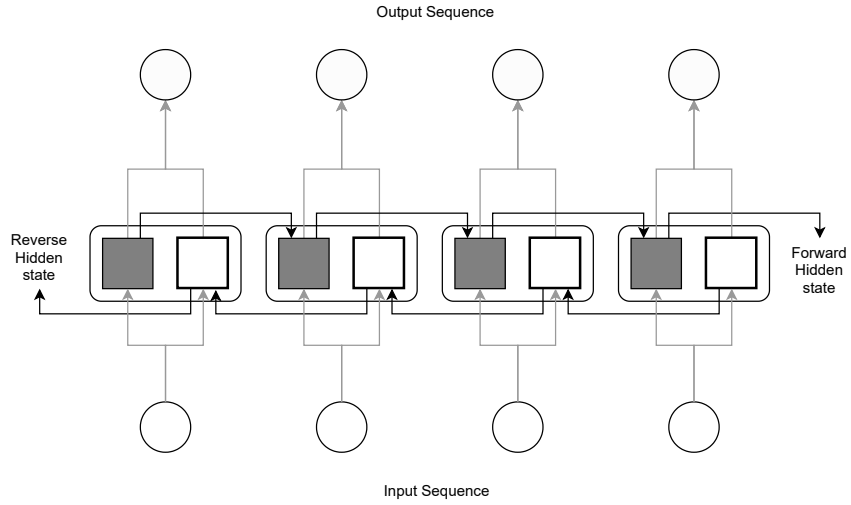


Figure 3.6: bi-LSTM with 4 sequence

3.3 Mimick

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

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

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

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

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 previous 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

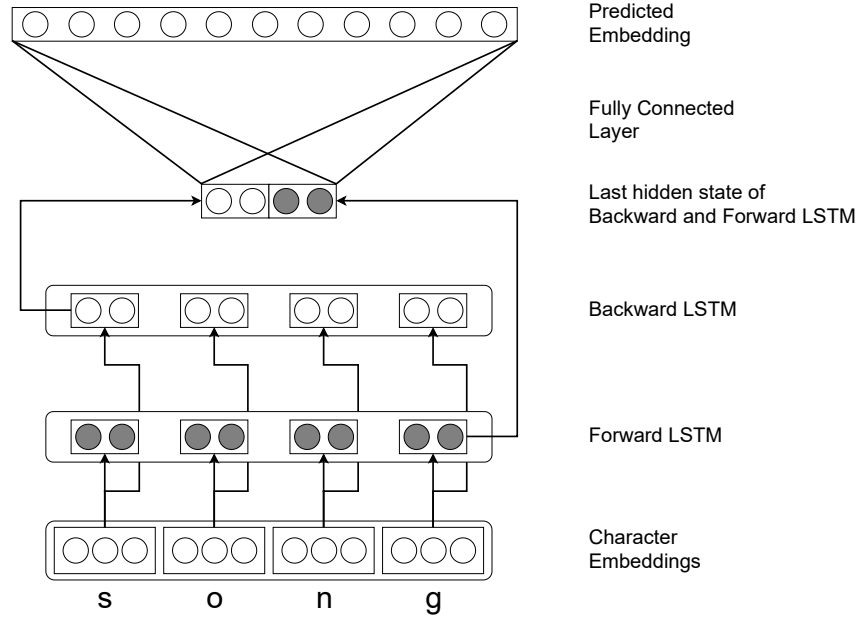


Figure 3.7: Mimick architecture

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 exists on the image data. This model concept was first introduced for handwriting recognition (LeCun, 1989).

In general, convolution is operation of two functions, the data and the kernel, that is by doing element wise multiplication from both function and then summed to get the total overlaps between both function at time t . In general, the kernel only has limited size that is non-zero 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 $g(t)$.

$$h(t) = (f * g)(t) \quad (3.20)$$

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

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

$$h(t) = (f * g)(t) \quad (3.22)$$

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

For two-dimensional data, the discrete convolution function becomes as shown in equation 3.24.

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

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 is 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 from a given kernel. Then the process of convolution and maxpooling is repeated until desired output size is produced. The process of maxpooling one patch of an input image is depicted in figure 3.9. Maxpooling process act as a gate for the highest response to receive backward connection for correcting the kernel. This is so that only parts of the image that has highest response will also corresponds to the correction of the kernel while the others treated as less useful for a certain feature.

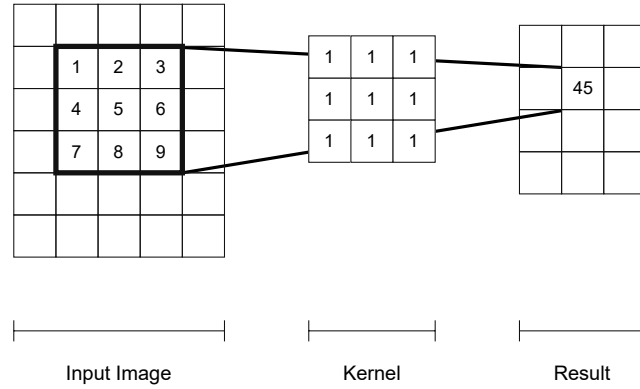


Figure 3.8: Convolution process of input image with kernel size 3×3

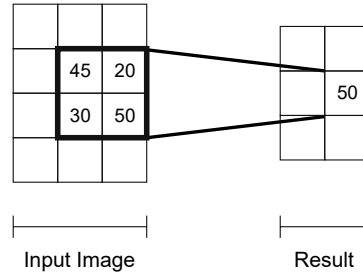


Figure 3.9: Maxpooling process of input image with window size 2×2

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

3.5 N-grams

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

Please do not sit ...

word *on* or *at* is more likely to follow instead of *run* or *bacteria*. In short, the previous task can be written as $P(w|h)$, probability of the next word w given some history part of sentence h . In previous case, the history h is "Please do not sit" and the probability in question is the following word w will be

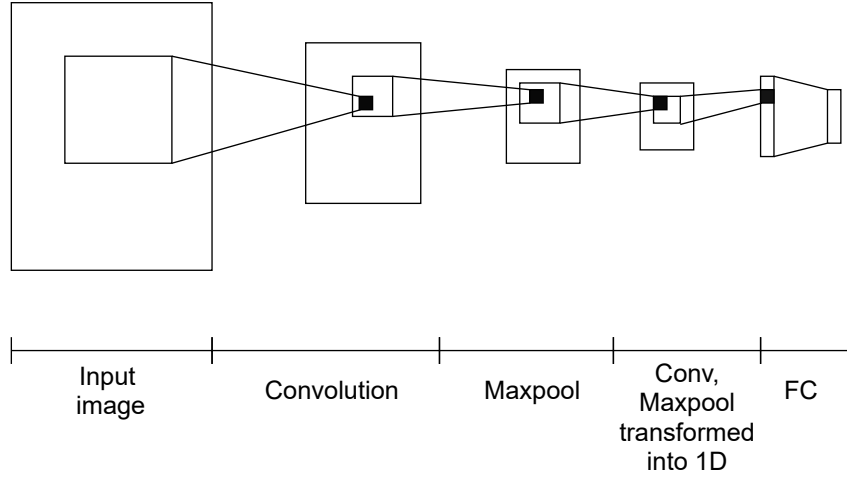


Figure 3.10: Example of CNN architecture

"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,

$$P(\text{on}|\text{Please do not sit}) = \frac{C(\text{Please do not sit on})}{C(\text{Please do not sit})} \quad (3.25)$$

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, \dots, w_n) , the joint probability of these sequence can be calculated by using chain rule as follows,

$$P(w_1, w_2, \dots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \dots P(w_n|w_1, w_2, \dots, w_{n-1}) \quad (3.26)$$

$$= \prod_{i=1}^n P(w_i|w_1, \dots, w_{i-1}) \quad (3.27)$$

As shown on equation 3.26, each occurrence of preceding sequence that followed by the desired word is estimated by counting the occurrence as shown in equation 3.25 for the whole history.

Instead of previous calculation, better way to calculate the probability of 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,

$$P(\text{on}|\text{Please do not sit}) \quad (3.28)$$

the approximation of the probability will be as follows,

$$P(\text{on}|\text{not sit}) \quad (3.29)$$

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

$$P(w_n|w_1, w_2, \dots, w_{n-1}) \approx P(w_n|w_{n-2}, w_{n-1}) \quad (3.30)$$

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

$$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^n P(w_i|w_{i-2}, w_{i-1}) \quad (3.31)$$

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, one 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 that is chosen 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. As mentioned in chapter Introduction and chapter Preliminaries, 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 passed 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 last 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 $C_t = 0$ in LSTM 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 early and later sequences information is held by the last hidden state for each reverse LSTM and forward LSTM respectively. Another problem might

un|recogniz|able
inter|national|ities
oto|rhino|laryngolog|ical
hepatico|chol|angio|gastro|stomy

Figure 4.1: Word examples with three or more subsequences

unrecognizable :unre|nrec|reco|ecog|cogn|ogni|gniz|niza|izab|
zabl|able
internationalities :inte|nter|tern|erna|rnat|nati|atio|tion|iona|
onal|nali|alit|liti|itie|ties
otorhinolaryngological :otor|torh|orhi|rhin|hino|inol|nola|olar|lary|
aryn|ryng|yngo|ngol|golo|olog|logi|ogic|gica|
ical
hepaticocholangiogastratomy :hepa|epat|pati|atic|tico|icoc|coch|cho|chol
hola|olan|lang|angi|ngio|giog|ioga|ogas|gast|
astr|stro|tros|rost|osto|stom|tomy

Figure 4.2: 4-grams examples

arise when we need to divide the sequence into more than three subsequence as shown in figure 4.1. Hence another approach is needed since intermediate subsequence might get deleted even by using bi-LSTM. Another method that might be able to solve this problem for MIMICK is by increasing the hidden size and hoping 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 divide 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). Afterwards, those sequences of character embeddings were 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 was given. In other words, given a set of vocabulary \mathcal{V} with size $|\mathcal{V}|$ and pretrained embeddings $\mathcal{W}^{|\mathcal{V}| \times d}$ for each word $w_i \in \mathcal{V}$ that is represented as a vector e_i with d -dimension, the model is trained to map function $f : \mathcal{V} \rightarrow \mathbb{R}^d$ that minimizes the following loss function,

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

This approach is similar to MIMICK Pinter et al. (2017). The text input was represented as a sequence of character $[c_1, c_2, \dots, c_m]$ 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 $[g_1, g_2, \dots, g_m]$ with length m will be called $\{g\}^m$. $\{g\}^m$ becomes 2-dimensional matrix that has size of $m \times b$. In summary, the word w was transformed using embedding generation function h into $\{g\}^m$ as follows,

$$h : w \rightarrow \{g\}_1^m \quad (4.2)$$

To process $\{g\}^m$ like an n-grams, CNN n-grams implementation by Kim (2014) was used. 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 function,

$$ReLU(x) = \max(0, x) \quad (4.3)$$

Several kernels were used to learn several features for producing embedding. 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 the given sequence of

character embedding $\{g\}^m$, only gram that produces the highest value when convoluted by using kernel k_i would be processed. Note that as mentioned earlier, each kernel k_i can also return a high response for j number of similar features from the input. Since, there are $|K|$ number of filter, $|K| + j$ number of grams would be considered to be important for calculating the result. 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 has a kernel with size $b \times 2$, trigram 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 was concatenated and fed into fully connected layer with output size matching the pre-trained embedding \mathcal{W} with d -dimension by using 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 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 $\mathcal{B} = x_1, x_2, \dots, x_m$, following operations were applied,

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

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad (4.5)$$

$$\tilde{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad (4.6)$$

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

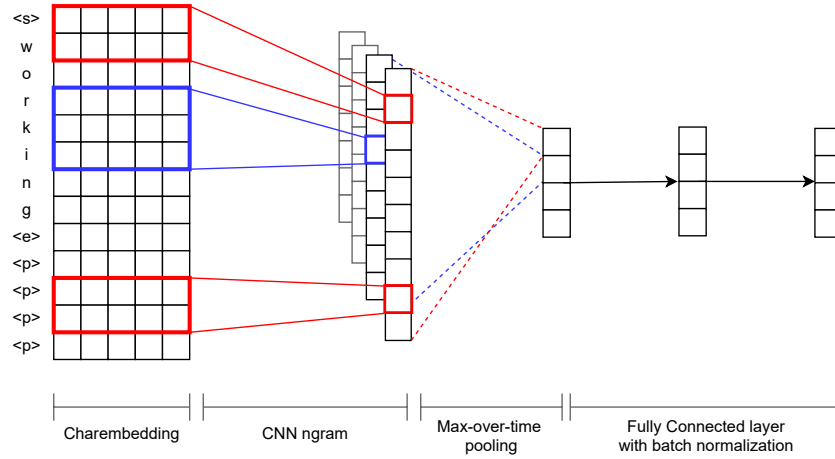


Figure 4.3: OOV Inferencing Model

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 these parameters 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 embedding prediction is shown on figure 4.3. On figure 4.3, starting and ending token were added at the beginning and at 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 did not go through maxpool layer since only grams that has the highest value could go through the next layer and the result of $ReLU(0) = 0$, thus the result of maxpooling would not come from the padding token. The reason for this is so that various length of words were able to be processed by the model.

4.1.3 Error and Backpropagation

After the embedding \tilde{e} was predicted, it was compared to the original embedding e to adjust all of the parameters for the neural network by using mean

squared error function,

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

By minimizing *Error*, it is similar to minimizing \mathcal{L} shown in equation 4.1. The gradient of the *Error* was 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. Then the results from the downstream tasks were 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 word usage in a sentence or a corpus based on the grammatical usage of the word (Horsmann, 2018), for example: verb, noun, adverb, etc. The POS-tagger model is similar to the one that was proposed by Ling et al. (2015). Given sentence $S = \{w \in \mathcal{V} | [(w_1, t_1), (w_2, t_2), \dots, (w_n, t_n)]\}$ 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 by using the pre-trained embedding. For the OOV, every sequence of the characters building a word transformed into sequence of character embedding $\{g\}_i^m$ using the function represented in equation 4.2 then the embedding \tilde{e}_i was predicted by using the OOV handling model. To separate embedding generation process for the in-vocabulary and the OOV, the output of the OOV handling model and the original embedding were masked in the following way,

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

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

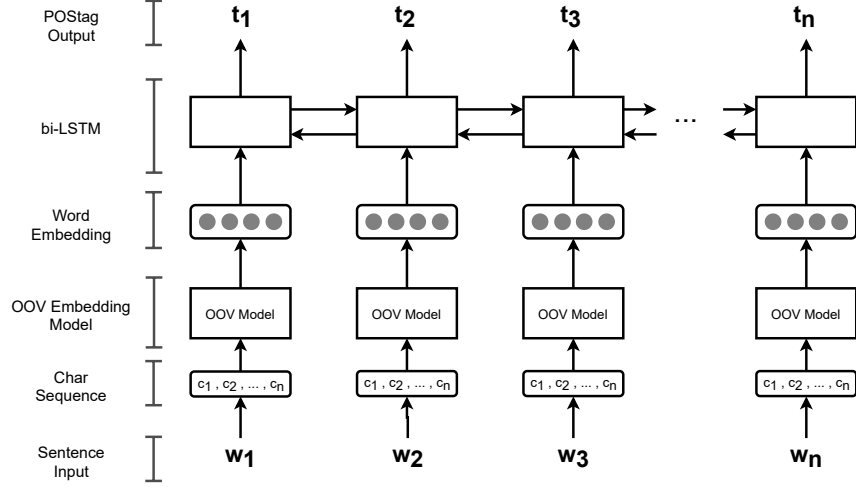


Figure 4.4: Pos-tagging Process

By doing so, the gradient would not flow into the OOV model if the word exist in \mathcal{V} and would flow if the word is unavailable 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,

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

to classify the POS-tag t . To ease up computation time, adaptive LogSoftmax was 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 the 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 the training was done, the accuracy of the POS-tagger based on different OOV handling model were compared.

4.2.2 Word Similarity Tasks

Word similarity 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 have stronger connection to a certain word than the other. 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 were calculated respectively using the following equation,

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

The OOV handling model was used if the word entry was an OOV and the original embedding was used if the word entry was in-vocabulary.

After all of the cosine distance for all pairs were calculated, Spearman rank's correlation from the dataset and the generated embedding were calculated by using equation 4.13 and by using equation 4.14 when no tied rank exists. The results of both MIMICK and the proposed model from several word similarity datasets then averaged and compared.

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

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

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 would tries to predict embedding from a known vocabulary $v_i \in \mathcal{V}$ with its known embedding $e_i \in \mathcal{W}$. 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. Word2vec¹ 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

¹Pretrained embedding available at <https://code.google.com/archive/p/word2vec/>

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, these kind of phrases were 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" were used in this research.

Another pre-trained embedding is polyglot² 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 was used as baseline model (Pinter et al., 2017).

Dict2vec³ is yet another word embedding that was 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 by using word similarity tasks with removal for OOV words.

Word Similarity Dataset

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

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

³Pretrained embedding available at <https://github.com/tca19/dict2vec>

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, 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 Colaboratory⁴ which gives randomized hardware specification based on the availability, thus the exact hardware configuration that was 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 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 $\eta = 0.01$ for polyglot and $\eta = 0.1$ for the rest with no dropout as dropout did not work well just as in the original implementation of MIMICK (Pinter et al., 2017). On the other hand, CNN

⁴Google 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]

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, but it has worse results in the pre-testing phase. Thus those settings were not used. The results on the downstream tasks for the different settings for the CNN-grams and MIMICK model model would be compared.

After max-over-time pooled was done to the convolution results, the vector representation was passed into two layers of fully connected network with batch normalization layer for each layer to predict the 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). After the embedding was predicted, the error was calculated by using Mean Squared Error as mentioned on equation 4.8 and the derivative was calculated then back-propagated to update the model parameters.

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 that were picked 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 NLTK⁵ were used to test the performance of the model as well as the previous *state-of-the-art*. In this task, three kinds of evaluation methods were done and will be explained below. Before going into the method, there were 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 for testing, would be added to the original embedding \mathcal{W} . The dataset were split up into 80% for train set and 20% for test set.

Non-trainable OOV Handling Model

The first method was to train the POS-tagger by using a pre-trained embedding as the baseline result. 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} would not be trained in this method by setting the parameters to be fixed values.

Trainable OOV Handling Model

The second method was to train the OOV model together with the POS-tagger. For this method, each OOV entries were added into the vocabulary \mathcal{V} and a zero vector representing the embedding of the OOV would be added to the

⁵Available at <https://www.nltk.org/>

embedding \mathcal{W}_{new} that later on replaced with the generated embedding $f(\text{OOV})$ from the OOV handling model if the input word is an OOV as explained in the equation 4.9 by masking the entries. Thus the OOV model would still be able to be trained if the input turns out to be OOV while it was non-trainable if it was in-vocabulary input. This method was used by both models then the results were compared.

OOV Handling Model and Pre-Trained Embedding Joint Training

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

On top of different methods used in this task, the sentence length was limited to 5 words. If the sentence length was 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 was more than 5 words, the slice of the sentence was 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 was 0.1 with dropout for CNN model 0.5 and no dropout for MIMICK since it makes the model performs worse based on preliminary tests. The momentum for the training was 0.5. In summary the used parameters 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 was 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 the two models, else the original embedding from the pre-trained embedding was used.

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

The baseline embeddings used in 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 for each dataset.

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 it is shown on the table below. Some of the input that is used were similar to the one used in testing MIMICK on the original paper for Polyglot embedding for English. 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.

Table 6.1: Nearest Neighbors Mimick (Polyglot)

OOV 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	$\exp \frac{n}{\frac{\#}{\sqrt{x}}}$

Table 6.2: Nearest Neighbors CNN (Polyglot)

OOV 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

The first word was "hurtling", typographical error from the word hurting. For this word, both model seemed to be able to locate that the nearest neighbor was another verb with suffix *-ing*. Interestingly for CNN, it could relate this word with action that might induce pain thus hurting a subject. The second word was "expectedly". Both model were able to predict that the nearest neighbor was another word with suffix *-ly*. The third input word was "Actively". For this input, MIMICK predicted that this word was a name which the first character in the word is capitalize, thus the nearest neighbor was a name. On the other hand, CNN predicted that this was some word with suffix *-ly* and the first character was capitalized. The next word was "corduroy", which is a pattern in textile. CNN model were able to predict that one of the nearest neighbor was "garment". The next input was "question-and-answer". This input was 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 was a number "1657". MIMICK predicted that the nearest neighbor was 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 were 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 for both model.

6.2 POS-tagging results

6.2.1 Randomly Generated OOV Embedding as Baseline

Before the OOV handling model was used, 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. Afterwards, This collection of embedding was used for training the POS-tagger model. The new was set to be trainable as mentioned in the previous chapter. The results for three different pre-trained embedding that was used in the POS-tagger are shown in figure 6.1.

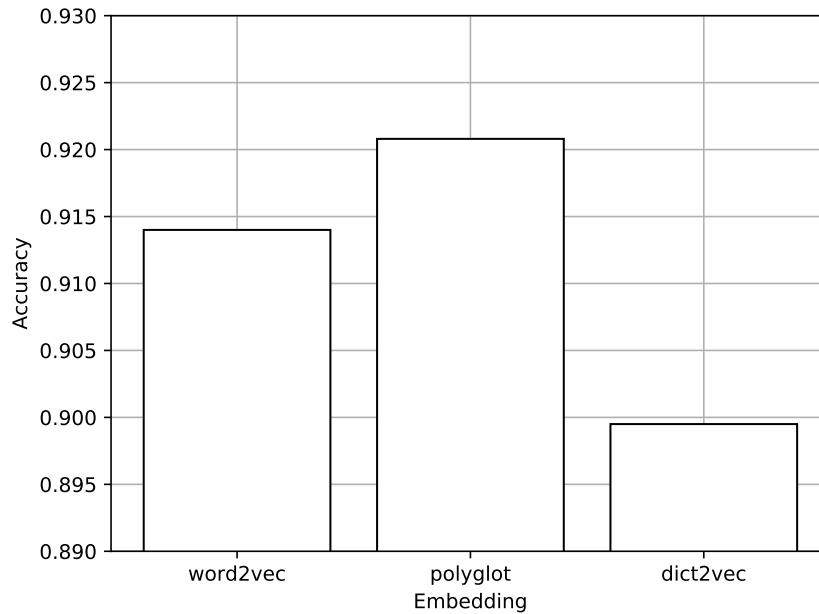


Figure 6.1: POS-tagging results with trainable random OOV embedding and pre-trained embedding

6.2.2 Non-trainable OOV Handling Model

In order to test the best number of features for each OOV handling model, the model and the embedding were set to be in evaluation mode or frozen, meaning that the parameters would not change when training the POS-tagger model. The OOV embedding from both model on top of the pre-trained embedding was used as input for POS-tagging task.

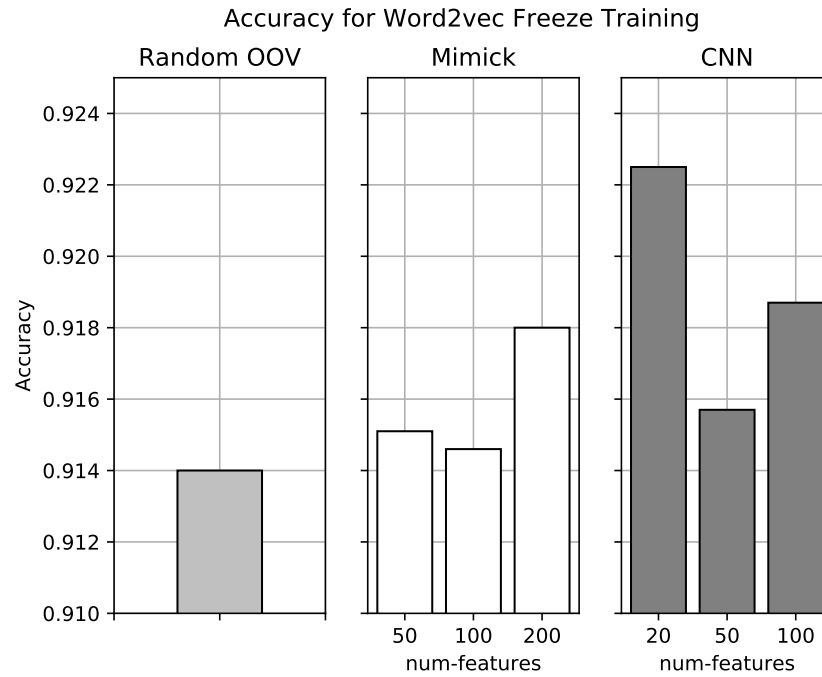


Figure 6.2: POS-tagging results with frozen embeddings and OOV handling model (Word2vec)

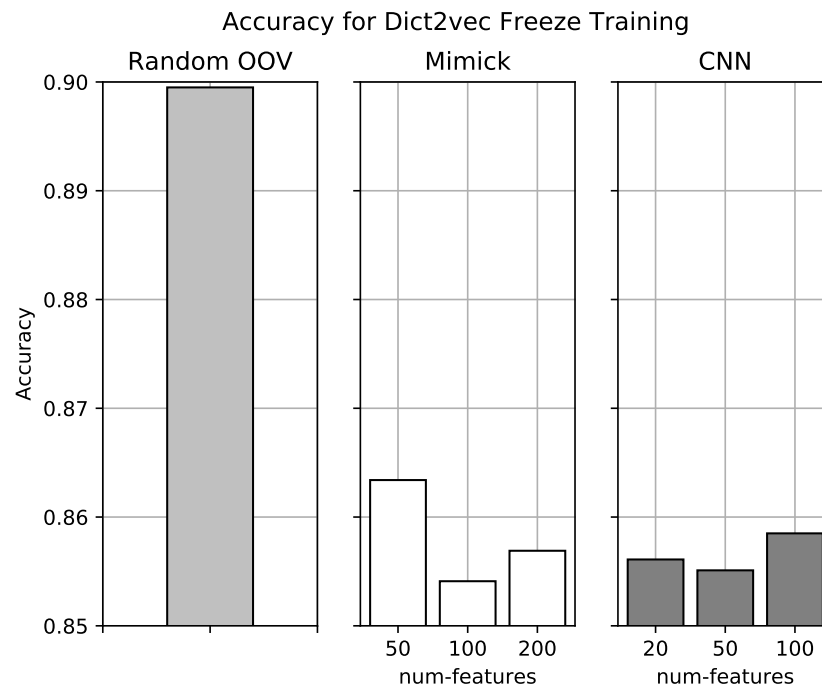


Figure 6.4: POS-tagging results with frozen embeddings and OOV handling model (Dict2vec)

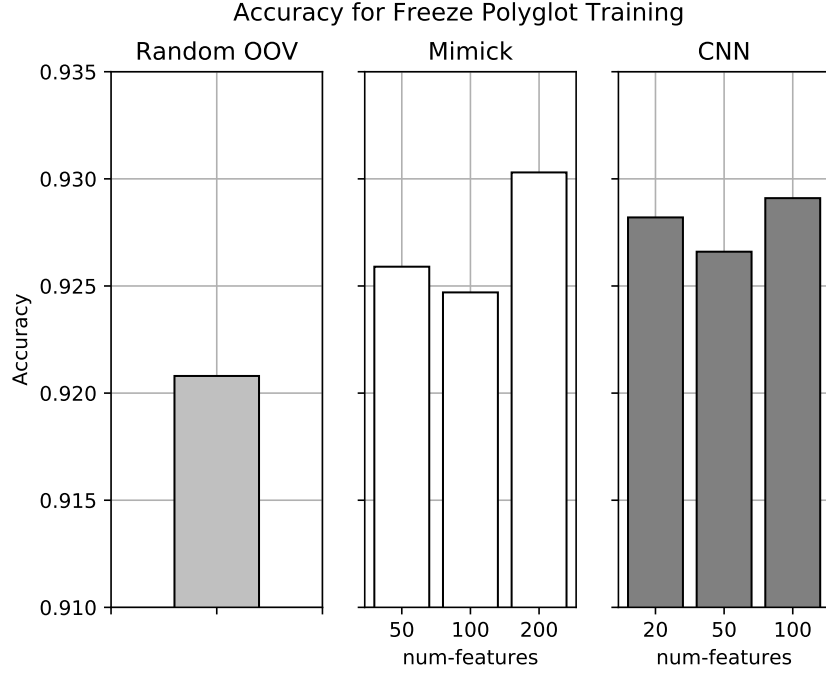


Figure 6.3: POS-tagging results with frozen embeddings and OOV handling model (Polyglot)

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 that was used. Interestingly, this setting for both models perform worse than randomly initialized OOV embeddings for Dict2vec pre-trained embedding.

6.2.3 Trainable OOV Handling Model

Secondly, the OOV handling model was also trained in conjunction with the POS-tagger to improve accuracies of the POS-tagger as well as to see whether further improvement in accuracy can 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 are 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 that used CNN as the OOV handling model were surpassing the accuracies from POS-tagger that used MIMICK as the OOV handling model in all of the pre-trained embeddings used in this research.

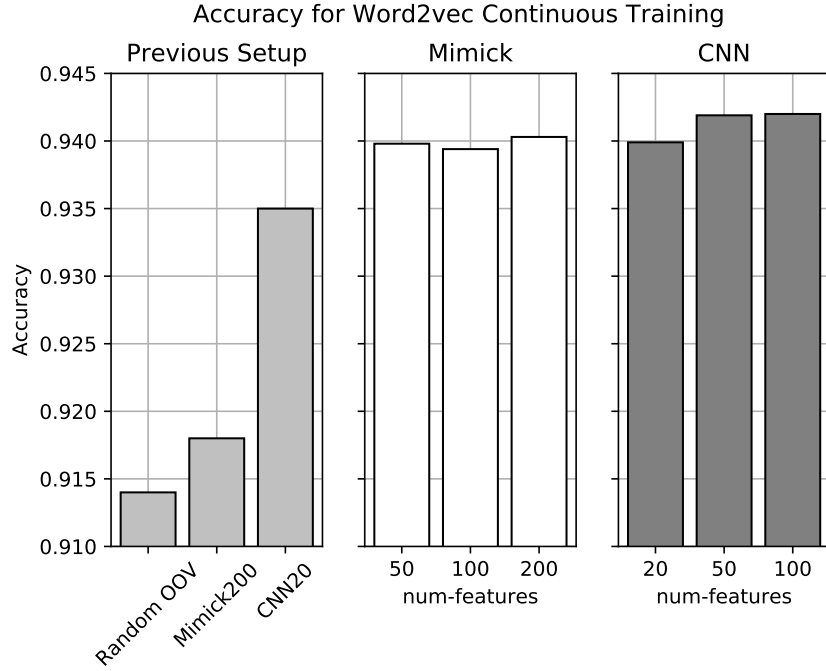


Figure 6.5: POS-tagging results with frozen embedding and trainable OOV handling model (Word2vec)

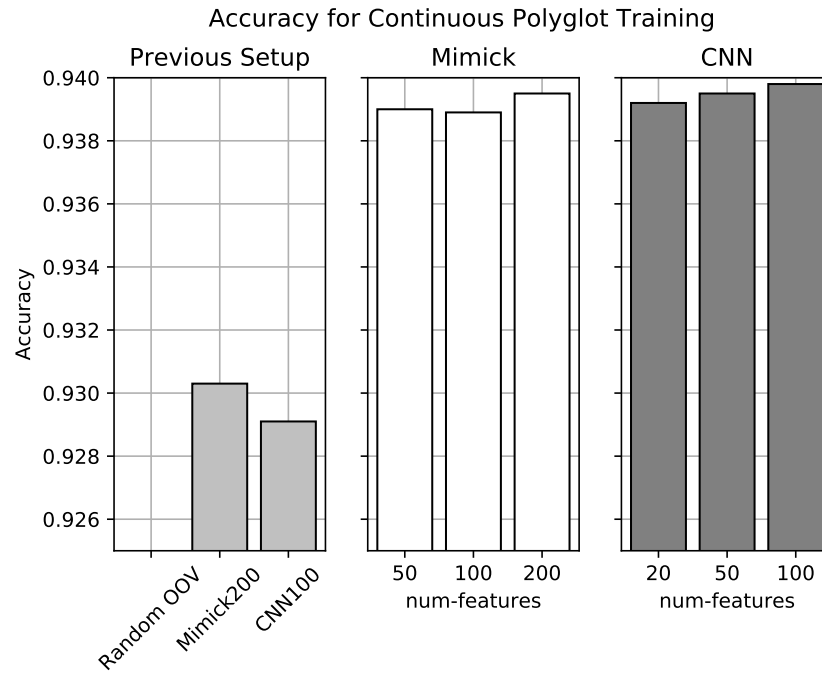


Figure 6.6: POS-tagging results with frozen embedding and trainable OOV handling model (Polyglot)

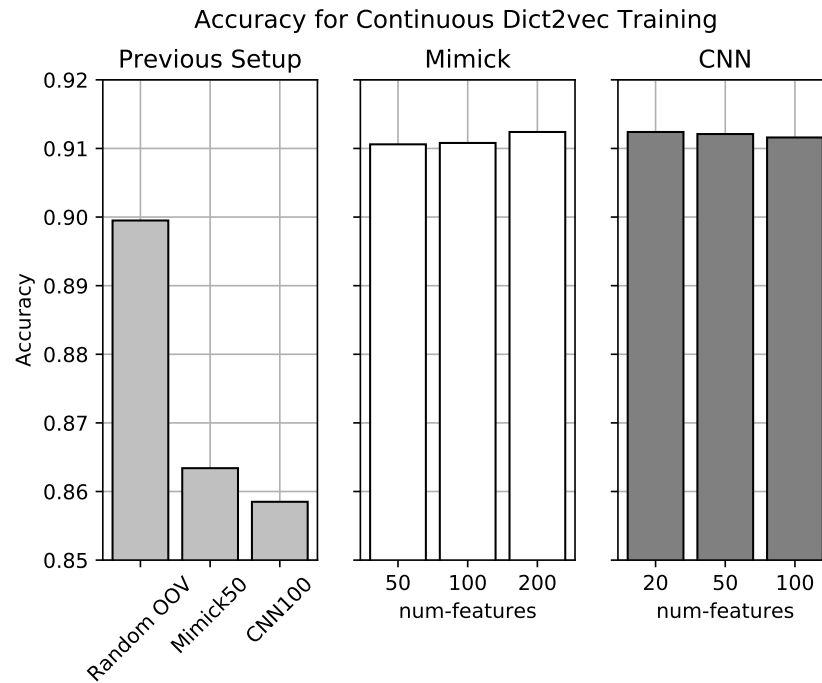


Figure 6.7: POS-tagging results with frozen embedding and trainable OOV handling model (Dict2vec)

6.2.4 OOV Handling Model and Pre-Trained Embedding Joint Training

Thirdly, to simulate how such model will be used in the downstream tasks application, both the pre-trained embedding and the OOV handling model would be trained and the results were compared to previously stated testing method. The number of features with the highest results from each OOV handling model for each word embedding from the second method were used for the current method. 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 performed worse for Word2vec embedding.

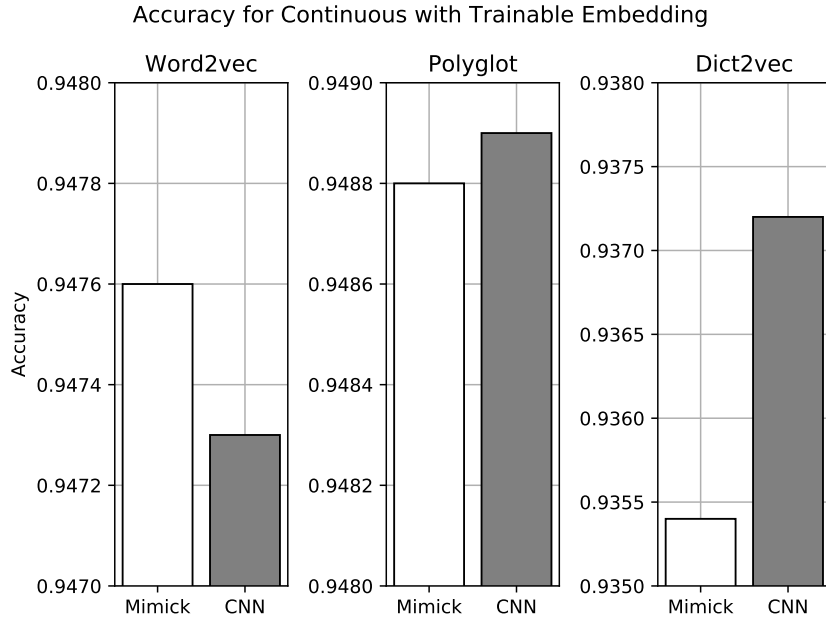


Figure 6.8: POS-tagging results with trainable pre-trained embeddings and OOV handling model

Given the results from various settings, in general CNN was 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 proves that CNN are able to learn OOV handling better for POS-tagging tasks compared to bi-LSTM that was used in MIMICK.

6.3 Word Similarity

For word similarity task, the cosine distance between each given pairs from the dataset were calculated. Afterwards, 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 the table 6.3, the OOV handling model was 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 are multiplied by 1000. From 12 word similarity datasets, CNN model has higher Spearman’s rank correlation coefficient on 5 datasets from 12 datasets with the average Spearman’s rank correlation coefficient 488.84 compared to MIMICK that achieved 492.05. The highest ρ value for each model and each dataset is marked with bold type setting in the table.

The same procedure was used for both models was that trained with poly-

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

glot (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 while CNN achieved 339.99, showing that CNN has higher ρ value compared to MIMICK as shown on Table 6.4.

For the baseline embedding Dict2vec (Tissier et al., 2017a), the original embedding was added with randomly generated embedding for OOV. The one with highest results from five random seed for each dataset was selected. Afterwards, the results was compared with CNN and MIMICK models. Dict2vec with random OOV embedding was only able to achieve Spearman’s rank correlation coefficient of 519.22 while 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

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

initializing embedding randomly for OOV for word similarity tasks, especially when there is OOV entries. On top of that, for Dict2vec word embedding CNN performs better than MIMICK in many datasets.

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.

6.4 Discussion

Based on the results shown above, both OOV handling model were able to predict the nearest neighbor in the vocabulary for some OOV. This shows that OOV handling model can be trained to predict embedding of an OOV. Sadly, it did not always working for some cases. Even though it was not producing good results for the nearest neighbor, this model was able to improve downstream tasks such as POS-tagging and word similarity compared to randomly

initialized OOV embedding for OOV handling.

As shown in the POS-tagger results, the model with OOV handling could improve the accuracy over the model with randomly initialized OOV handling even though the pre-trained embedding and the randomly initialized embedding were trained in regards to the POS-tagger loss. Further improvements could be achieved when both the OOV handling model and the pre-trained embedding were also trained together with the POS-tagger.

Subsequently, both OOV handling model achieved different results for POS-tagging task. MIMICK was performing well when both the pre-trained embedding and the OOV handling model were set to be non-trainable. This case is true for Polyglot and Dict2vec embedding while it is not for Word2Vec. On the other hand, CNN N-grams was performing well when both the pre-trained embedding and the OOV handling model were trained with the POS-tagger model, especially for the one that was trained with Polyglot and Dict2vec. Those results shows that while MIMICK can be used if the pre-trained embedding is not meant to be trained while CNN can be used if the pre-trained embedding is meant to be included in the training graph. If the pre-trained embedding is not trained, then MIMICK has the advantageous of higher accuracy for POS-tagging task and cheaper computation for predicting the OOV embedding. In contrast, when the pre-trained is suppose to be trained in conjunction with the OOV handling model, CNN could achieve higher accuracy compared to MIMICK.

Despite the fact that MIMICK can outperform CNN when the pre-trained embedding and the OOV handling model is not included in the training process, this case does not hold for word similarity task where there is also no trainable parts of the calculation. Both OOV handling model were only used for predicting the OOV embedding to calculate the cosine distance between word pairs. Although compared to randomized OOV embedding, both model still performed better for Dict2vec pre-trained embedding.

Chapter 7

Conclusion and Future Works

Given some training examples of pretrained word embedding, machine learning can 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 unknown token “<UNK>” with one vector representation replacing all *out-of-vocabulary* entries. This approach has its advantages such as cheaper computation time to generate embedding and has relatively high accuracies at around 91% for POS-tagging task. However, the training on the downstream tasks using randomly generated embedding might need more epoch or more sophisticated architecture to be able to achieve similar accuracy 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 in general. The randomly generated embedding for *out-of-vocabulary* entries method could produce POS-tagging results for three different pretrained embedding averaged around 91% and by using OOV hand-

ling model the accuracy increased to be around 93%.

7.2 CNN against bi-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. There can exist a case where the cell gate decided to drop the previous sequence rendering information from early sequence to be missing. Even though by the nature of neural network that it is a black box, the solution for such problem in this case such as increasing the size of the hidden neuron size and by using bidirectional LSTM to reduce the chance of such cases to be minimized, MIMICK still perform worse than CNN N-grams. It can also be that the parameters used in this research is not high enough, so further analysis for this case needed to be done. On the other hand, 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.

7.3 CNN against bi-LSTM for Downstream Tasks

7.3.1 POS-tagging

In POS-tagging tasks, CNN model accuracy was averaged around 0.2% higher than MIMICK across all settings. 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.

7.3.2 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 and trained with different purposes. Despite of that, CNN still able to performs better than MIMICK.

7.4 Future Works

Both models MIMICK and CNN were able to produce embeddings for *out-of-vocabulary* words, but both model are not really able to fully 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 that is completely different making the word embedding becomes obsolete. On top of that, if the function can generate the in-vocabulary embedding, it can save storage capacity since in general pretrained embedding storage size is 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.
- 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.
- 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.
- 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.
- 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.
- 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.

- Eisenstein, J., O’Connor, B., A. Smith, N., and P. Xing, E. (2012). Mapping the geographical diffusion of new words. *PLOS ONE*, 9.
- 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.
- Forrester, J. C. (2008). A brief overview of english as a language in change. *CCSE*, abs/1508.06615(4).
- Garneau, N., Leboeuf, J., and Lamontagne, L. (2019). Predicting and interpreting embeddings for out of vocabulary words in downstream tasks. *CoRR*, abs/1903.00724.
- Ghosh, S. and Kristensson, P. O. (2017). Neural networks for text correction and completion in keyboard decoding. *CoRR*, abs/1709.06429.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- Grave, E., Joulin, A., Cissé, M., Grangier, D., and Jégou, H. (2016). Efficient softmax approximation for gpus. *CoRR*, abs/1609.04309.
- 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.
- Hill, F., Reichart, R., and Korhonen, A. (2014). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *CoRR*, abs/1408.3456.
- Horsmann, T. (2018). *Robust Part-of-Speech Tagging of Social Media Text*. PhD thesis, Universität Duisburg-Essen.
- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.
- Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

- 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.
- Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2015). Character-aware neural language models. *CoRR*, abs/1508.06615.
- 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.
- 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.
- Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR*, abs/1603.01360.
- LeCun, Y. (1989). *Generalization and network design strategies*. Elsevier.
- Li, Y. and Yang, T. (2017). *Word Embedding for Understanding Natural Language: A Survey*, volume 26.
- Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fernandez, 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.
- Liu, B. (2010). Sentiment analysis and subjectivity. In *Handbook of Natural Language Processing*.

- 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.
- 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*.
- 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.
- Miller, G. A. and Charles, W. G. (1991). Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28.
- 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*.
- 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.
- 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*.
- 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.
- Rocchesso, D. (1995). Sound processing. *Computer Music Journal*, 19.
- Rubenstein, H. and Goodenough, J. B. (1965). Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633.

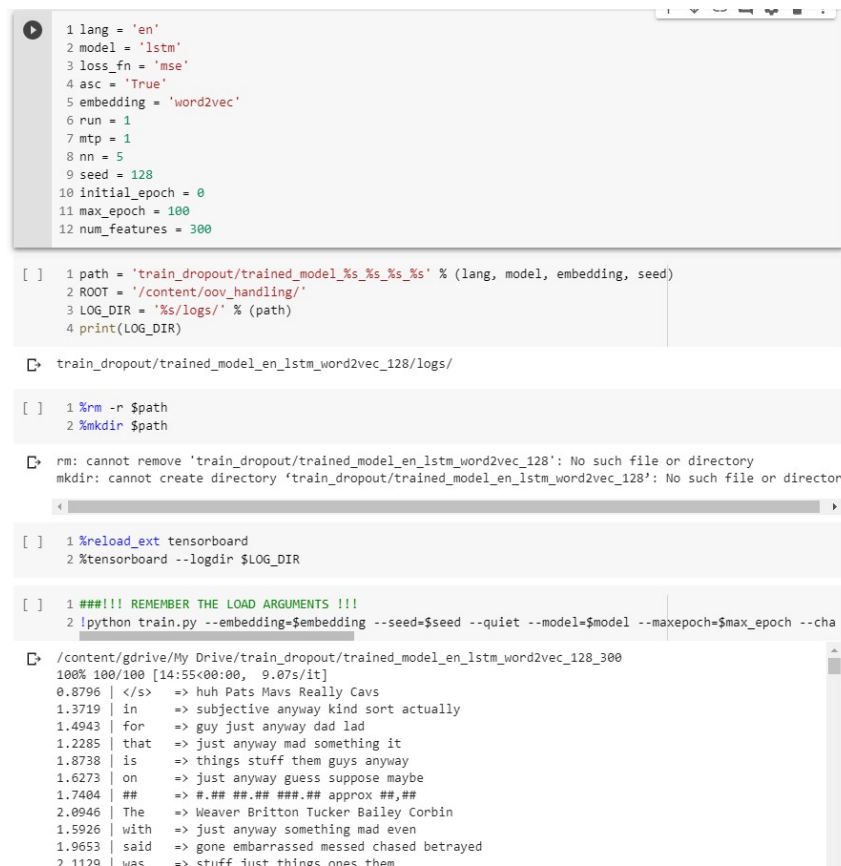
- S. Harris, Z. (1954). Distributional structure. *Word*, 10:146–162.
- Tissier, J., Gravier, C., and Habrard, A. (2017a). Dict2vec : Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263, Copenhagen, Denmark. Association for Computational Linguistics.
- 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.
- Tyagi, V. (2018). *Understanding Digital Image Processing*. CRC Press.
- Yang, D. and Powers, D. (2006). Word similarity on the taxonomy of wordnet.

Appendix

Code

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

Training Screenshot



```
1 lang = 'en'
2 model = 'lstm'
3 loss_fn = 'mse'
4 asc = 'True'
5 embedding = 'word2vec'
6 run = 1
7 mtp = 1
8 nn = 5
9 seed = 128
10 initial_epoch = 0
11 max_epoch = 100
12 num_features = 300

[ ] 1 path = 'train_dropout/trained_model_%s_%s_%s_%s' % (lang, model, embedding, seed)
2 ROOT = '/content/oov_handling/'
3 LOG_DIR = '%s/logs/' % (path)
4 print(LOG_DIR)

[ ] train_dropout/trained_model_en_lstm_word2vec_128/logs/

[ ] 1 %rm -r $path
2 %mkdir $path

[ ] rm: cannot remove 'train_dropout/trained_model_en_lstm_word2vec_128': No such file or directory
mkdir: cannot create directory 'train_dropout/trained_model_en_lstm_word2vec_128': No such file or director

[ ] 1 %reload_ext tensorboard
2 %tensorboard --logdir $LOG_DIR

[ ] 1 ###!!! REMEMBER THE LOAD ARGUMENTS !!!
2 !python train.py --embedding=$embedding --seed=$seed --quiet --model=$model --maxepoch=$max_epoch --cha

[ ] /content/gdrive/My Drive/train_dropout/trained_model_en_lstm_word2vec_128_300
100% 100/100 [14:55<00:00, 9.07s/it]
0.8796 | </s> => huh Pats Mavs Really Cavs
1.3719 | in => subjective anyway kind sort actually
1.4943 | for => guy just anyway dad lad
1.2285 | that => just anyway mad something it
1.8738 | is => things stuff them guys anyway
1.6273 | on => just anyway guess suppose maybe
1.7404 | ## => ### ##.## approx ##,##
2.0946 | The => Weaver Britton Tucker Bailey Corbin
1.5926 | with => just anyway something mad even
1.9653 | said => gone embarrassed messed chased betrayed
2.1129 | was => stiff just things nnae them
```

Figure: OOV Handling Model Training

```

1 for seed in [64,20,128,0,5,10]:
2     print(seed)
3
4     path = 'trained_model_%s_%s_%s_postag' % (lang, model, embedding)
5
6     %rm -r $path
7     %mkdir $path
8
9     # set paths
10    ROOT = %pwd
11    LOG_DIR = './%s/logs/' % (path)
12
13    print(LOG_DIR)
14
15    !python train_postag.py --train_embed --oov_random --seed=$seed --trained_seed=$trained_
... 64
rm: cannot remove 'trained_model_en_cnn_word2vec_postag': No such file or directory
./trained_model_en_cnn_word2vec_postag/logs/
98% 98/100 [45:12<00:55, 27.61s/it]

```

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 --charlen=20 --num_feature=$num_featur
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