OOV Handling by Learning Subword using CNN based N-grams

Yonathan Purbo Santosa

Matriculation Number: 2993106

XXXX XX, XX

Master thesis

Computer Science

Supervisors:

Prof. Jehn Lehmann

Dr. Giulio Napolitano

Institut für Informatik III

Rheinische Friedrich-Wilhelms-Universität Bonn

Declaration of Authorship

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

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other quali

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

Signed: _____

Date:

Acknowledgements

blah blah blah

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Word embeddings is a method of word representation mainly used for natural language processing. Text data appears differently in computer from images and sounds. Generally image represented as two-dimensional matrix with finite number of points cell elements containing numerical value (Tyagi, 2018). On the other hand, sound is represented as one-dimensional signal representing air pressure in the ear canal (Rocchesso, 1995). Both images and sounds can be easily represented as mathematic models either using analog or digital signals but not with text data (Li and Yang, 2017). Text data consists of strings that can only be modeled using one-hot vector (from given d-dimension for d words that are known, only one dimension is one and the rest are zeroes). This one-hot vector does not have any information that infers connection between one to the other. Hence that, vector representations that maps semantic and syntactic information given one-hot vectors in a euclidean space is introduced (Li and Yang, 2017). This positional information then can be used to infer interconnection between words, whether its similarities or usage of the word in a sentence (S. Harris, 1954). To obtain word embeddings, a model is created to extract features of a word from a corpus and map its loc-

ation in euclidean space based on the features found. These word embeddings then can be used to do many downstream tasks, such as postagging, named entity recognition (NER), and sentiment analysis (Ling et al., 2015; Lample et al., 2016). In general, large corpus with many words and examples of word usage is prefered because the size of the vocabularies will increases and more words connection can be inferred from the corpus (Kutuzov and Kunilovskaya, 2018). However, it is not posible to have such corpus since the language itself is changing overtime (Forrester, 2008). Furthermore, there are cases of typographical error, especially after internet and social media are booming where anybody willingly write text over these platforms and these words may not be present in the corpus hence not included in the vocabulary thus making many typographical error while in its correct form it actually is. In addition with the increased number of smartphone which uses touch screen, some typographical error is expected (Ghosh and Kristensson, 2017). All this words that are non-existent in the corpus and its embedding cannot be inferred are called as *out-of-vocabulary* (OOV) words. One may use simple approach by asigning unique random embedding or by replacing OOV with an unknown $<UNK>$ embedding. While in some cases using these simple approaches and continue on the training on the downstream tasks can produce acceptable results {CITE}, further improvement on downstream tasks can be achieved by using machine learning method to infer OOV embeddings.

To infer OOV embeddings, the model is built over quasi-generative perspective. Only knowing the vocabularies and its embedding, the model tried to generate embedding for OOV words. Previous *state-of-the-art* used lstm to infer OOV embeddings (Pinter et al., 2017). Character embedding is used to transform sequence of characters then forwarded into bi-lstm then to fully connected layer. In language model, bi-lstm generally works by separating sub-word by remembering and forgeting previous sequence from both end. Those sub-word then will be used to infer its embedding. The problem might arise when there are more than two important sub-words and they are not

in sequence, meaning that there exist at least one character between two sub-words considered important, hence the information is incomplete since lstm will dampen the previous sequence if the next sequence sub-words are considered more important. Because of the aformentioned problem, a new method will be proposed.

1.2 Objectives

Instead of taking the whole words as a sequence and considering its importance based on time using LSTM, by using n-grams and picking which grams that are considered to be important in theory should gives better results since the information is complete and only left for the model to pick which n-grams features are more important. The only problem is that for the model to pick which grams that should be included in the model is impossible to do since there are many word combinations making the model needs to accept huge number of inputs. Instead of handpicking the features of n-grams, convolutional neural network (CNN) can be used to pick which features needs to be considered by using character embedding and treat the character sequence embedding as an image. The features picked then will be processed to predict the word embedding for the input word.

1.3 Contributions

1. A new OOV handling model
2. Improved OOV handling model

1.4 Thesis Structure

The remainder of this document is structured as follows. In the Related Work chapter, the previous works that are in relatives with research done in this

documents are mentioned especially the baseline used in this research. In the following, Preliminaries chapter, base theories that considered to be needed are explained in details here. In the Method chapter, the method of solution proposed to the problem are explained. In the Implementation chapter, the method of solution proposed to the problem are described in technical way to show how it is implemented. In the Results and Discussion chapter, the results are shown and discussed the relation with the baseline. Lastly, Conclusion chapter talks about the conclusion that are able to pull from this research.

Chapter 2

Related Work

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

Polyglot embedding .....

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

uated using several word similarity tasks to show imporvements over vanilla implementation of word2vec and fasttext.

Part-of-Speech-Tagging (POStagging) is a process of determining grammatical category (tag) of given word in a certain sentence. In English, exist words that has ambiguous grammatical category, such as word "tag" can be either noun or verb depends on the usage of it (Cutting et al., 1992). To tackle this problems, many researchers proposed to use mathematical models or statistical models namely hidden markov model (Cutting et al., 1992), n-grams (Brants, 2000), and neural network model (Ling et al., 2015). In this research, the neural network model will be implemented to serve as the downstream task. This model is a recurrent neural network (RNN) that took sequence of word embeddings representing a sentence or parts of sentence then categorize each word embedding for its tag.

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

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

Chapter 3

Preliminaries

3.1 Feedforward Neural Network

Feedforward neural network or also known as multilayer perceptron are a mathematical model that inspired from how neuron works in biological body (Goodfellow et al., 2016). The model takes numerical input and produces numerical output. 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. Given input vector $\mathbf{x_i}$ with n-dimension from a dataset $D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_i}, y_i)\}$, the model tries to predict a target y_i correctly given a set of weights $\mathbf{w}$. First the output o are calculated by calculating the dot product between $\mathbf{x_i}$ and $\mathbf{w}$ as shown on equation 3.1.

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

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

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

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

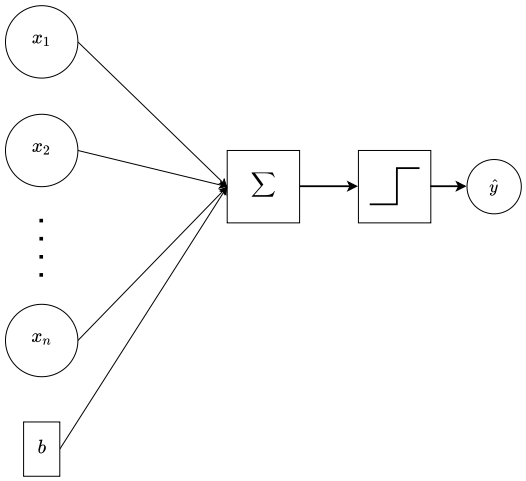

Figure 3.1: Perceptron

The input information fed through the model in chain with no feedback thus the name feedforward. When the output are fed back into the model, the model becomes recurrent network and it will be explained on the next section. The model depicted in figure 3.1.

The results then compared with the original output y_i in the dataset to fix the parameter weight $\mathbf{w}$ and bias b by using backpropagation algorithm. The algorithm defines cost funtion and calculate the gradient of the current loss then the gradient information is processed by another algorithm called stochastic gradient descent tries to find the optimal parameters that produced the minimum cost. If $\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}) = \mathbf{x} \cdot \mathbf{W}$ and the cost $z = J(\mathbf{y})$ the simple calculation of the gradient can be calculated as follows,

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

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

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

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

Notice that the gradient is multiplied by -1 because the gradient points uphill

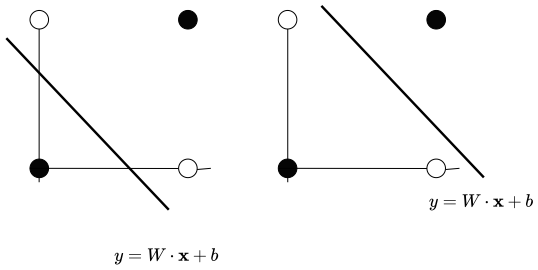

Figure 3.2: XOR Problem

and to find the minimum cost, the parameters needs to traverse downhill on the cost function.

On the developement of perceptron, it is clear that model consist of single layer of input and single layer of output does not enough to solve XOR problem (Goodfellow et al., 2016), since what perceptron does is separating the space with a hyperplane or in XOR problem a hyperline as shown on figure 3.2. Thus multilayer perceptron were introduced. This model consist of single layer of input, single layer of output, and one or more hidden layer. On top of that, more non-linear activation function were introduced. Some of those are sigmoid, tanh, rectified linear unit (ReLU), and softmax described in equation 3.7 until 3.10.

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

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

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

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

3.2 Long-short Term Memory

As aformentioned before, recurrent neural network (RNN) is a neural network model that has feedback to its own neuron depicted in figure 3.3. This model

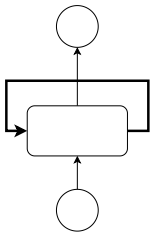

Figure 3.3: Recurrent Neural Network

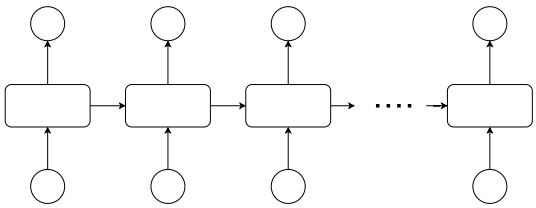

Figure 3.4: Unfolded Recurrent Neural Network

used for processing sequential data (Goodfellow et al., 2016). The idea is given sequence of input with length of t, $\mathbf{x}_i^{(t)} = x^{(1)}, x^{(2)}, \ldots, x^{(\tau)}$, the input is processed with shared parameter to give t-lengths output as depicted in figure 3.4. Common usage of RNN is for processing sentence. For example, given two sentences "I went to Paris in 2004" and "In 2004 I went to Paris". If we query the model to extract information from both sentences and ask when did the narator went to Paris, 2004 would be the relevant information regardless when it is appears in a sentence (Goodfellow et al., 2016). If such task is trained using a feedforward neural network that processes fixed size inputs, the parameters for each input feature will be separated for each sequence. In comparisson with feedforward neural network, RNN will use the same parameters to process all the input sequence.

The problem with RNN is that when given a really long sequence, either the gradient calculation will vanish since if weight is near zero and is multiplied several times sequentially over time or the gradient exploded if the weight bigger than one and it is multiplied several times (Goodfellow et al., 2016). Those problems making processing long sequence on RNN unfavorable. Hence another method called long-short term memory (LSTM) was introduced. LSTM

was created with idea in mind that some paths exist to give gradient ability to flow for long sequence. This was achieved by introducing self-loops inside the hidden layer that has time-scale mechanism that can change dynamically based on the input (Goodfellow et al., 2016). New components called cell gate C_t, forget gate f_t, and input gate i_t are introduced to let the reccurent network split the graph of the hidden unit thus gradient can be flowed longer by depending only from the output o_t or from both output and hidden states o_t and h_t respectively. The cell gate, interact with input and previous hidden state h_{t-1} to decide the current hidden state h_t. The calculation process of LSTM for each time step $t = 1$ to $t = \tau$ is as follows,

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

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

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

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

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

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

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

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

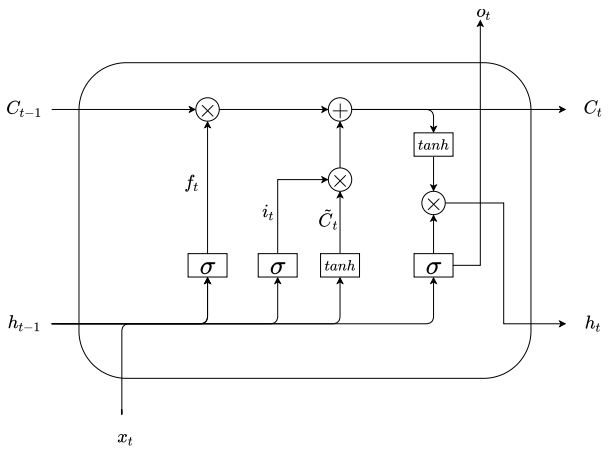

Figure 3.5: Gates inside LSTM hidden cell

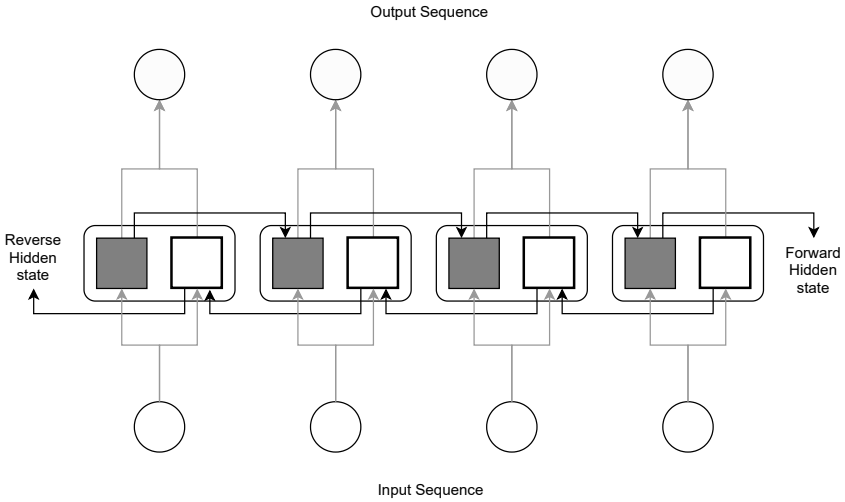

Figure 3.6: bi-LSTM with 4 sequence

3.3 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 1D grid data and for an image data, CNN can process this as a 2D grid or 3D grid given the number of channel presents on the image data. This model concept was first introduced for handwriting recognition (LeCun, 1989).

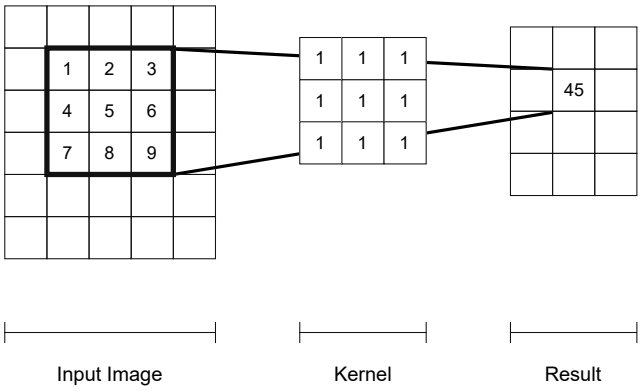

Figure 3.7: Convolution process of input image with kernel size 3×3

In general, convolution is operation on two function that is taking values from both function and multiplied together and summed to get the total overlaps between both function at time t. This process is easier to be explained with equation 3.19.

$$h(t) = (f * g)(t) \tag{3.18}$$

$$h(t) = \int_{-\infty}^{\infty} f(u)g(t-u)du \tag{3.19}$$

In discrete type signal, the calculation processes becomes as shown in equation 3.21.

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

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

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. 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 3.7.

Another intermediate layer that can also be used in CNN called maxpooling layer. Maxpooling is a process of finding a maximum value inside a given

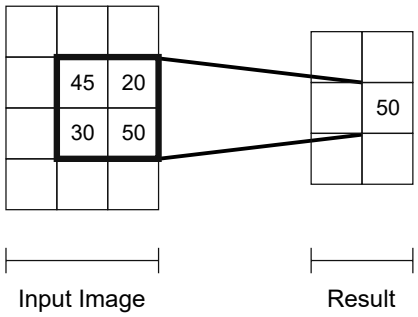

Figure 3.8: Maxpooling process of input image with window size 2×2

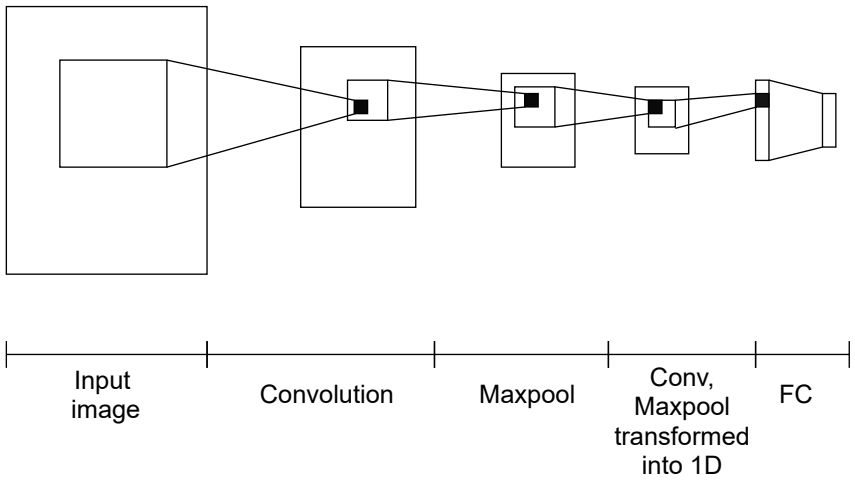

Figure 3.9: Example of CNN architecture

window from a given grid. In CNN, maxpooling is used for finding within the input data that gives the highest response with a given kernel. Then the process of convolution and maxpooling is repeated until desired architecture is produced. The process of one patch of an input image is depicted in figure 3.8

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 3.9. In most cases, after doing convolution, non-linear activation function are applied.

3.4 N-grams

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

`Please do not sit ...`

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

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

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

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_n|w_1, w_2, \ldots, w_{n-1}) \tag{3.22}$$

$$= \prod_{i=1}^{n} P(w_i|w_1, \ldots, w_{i-1}) \tag{3.23}$$

As shown on equation 3.22, each occurence of preceeding sequence that followed by the desired word is estimated by counting the occurence as shown in equation 3.4 for the whole history.

Instead of previous calculation, better way to calculate the word W given history h is needed because as stated above language is creative making calculating exact probability impossible and there will be too many estimation if there is long sequence that preceeds 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 preceeding sequences will be taken into calculation. In other words, instead of following probability calculation,

$$P(\text{on}|\text{Please do not sit}) \tag{3.24}$$

the approximation of the probability will be as follows,

$$P(\text{on}|\text{not sit}) \tag{3.25}$$

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

$$P(w_n|w_1, w_2, \ldots, w_{n-1}) \approx P(w_n|w_{n-2}, w_{n-1}) \tag{3.26}$$

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

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i|w_{i-2}, w_{i-1}) \tag{3.27}$$

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

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

Chapter 4

Method

4.1 Out-of-Vocabulary Model

4.1.1 Sequence Feature Extraction

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

Formally, when $C_t = 0$ from equation 3.14, hidden state from equation 3.16 will be reset to 0 rendering hidden states prior to time t gone. This problem can be solved by using bi-lstm, since bi-lstm process sequence in forward and reverse order making both early and later sequence held by the last hidden state for each forward lstm and reverse lstm respectively. Another problem might arise when we need to divide sequence into more than three subsequence as shown on figure 4.1. Hence another approach is needed since intermediate subsequence might get deleted or carried along with the later sequence even

$un|recogniz|able$

$inter|national|ities$

$oto|rhino|laryngolog|ical$

$hepatico|chol|angio|gastro|stomy$

Figure 4.1: Word examples with three or more subsequences

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

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

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

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

Figure 4.2: 4-grams examples

with bi-lstm.

We can find many examples especially from more technical areas as shown on figure 4.1. Since on Mimick (Pinter et al., 2017) implementation only the last hidden state is used for inferencing word embedding, another approach is proposed.

For all subsequence to be processed, we need a method that accounts for all sequence yet still able to divides the whole sequence into subsequences. Consequently, n-grams is chosen because this method splits word into sequence of characters depends on the chosen window size as shown on figure 4.2 inspired

from CNN word n-grams (Kim, 2014). Those sequences of characters then feed into learning algortihm. This idea is similar to how human tries to recognize an unseen word by reading subword that is understandable beforhand when no explanation or context were given.

This model then will be used to estimate OOV embeddings. In other words, given sets of vocabulary $\mathcal{V}$ with size $|\mathcal{V}|$ and pretrained embeddings $\mathcal{W}^{|\mathcal{V}|\times d}$ for each word $w_i \in \mathcal{V}$ that is represented as a vector e_i with d dimension, the model is trained to map function $f : \mathcal{V} \rightarrow \mathbb{R}^d$ that minimizes $|f(w_i) - e_i|^2$. This approach is similar to MIMICK (Pinter et al., 2017) approach. The text input is represented as a sequence of character $[c_1, c_2, \ldots, c_m]$ for $c_i \in \mathcal{C}$. Those sequence then transformed as sequence of vectors g_i with b dimension by using character embeddings $\mathcal{G}^{|\mathcal{C}|\times b}$. For simplicity, sequence of $[g_1, g_2, \ldots, g_m]$ will be called $[\hat{g}]$. $[\hat{g}]$ becomes 2-dimensional matrix that has size of $m \times b$. In summary, given word w will be transformed using function h into $[\hat{g}]$ as shown on equation 4.1.

$$h : w \rightarrow [\hat{g}] \tag{4.1}$$

To process $[\hat{g}]$ like an n-grams, CNN is used. CNN is basically a method to do convolution on matrix by using a kernel $k_i^{n\times b} \in K$. This operation is represented with $*$ symbol. To mimick n-grams, kernel with size of $n \times b$ is used, producing another vector $\hat{l}$ that represents the value of each grams, then non-linearity is applied to this vector by using ReLU activation function as shown in equation 4.2.

$$f(x) = ReLU(x) = max(0, x) \tag{4.2}$$

Several kernel is used to learn several features for producing embeddings. Each of these kernel will be responsible to find grams that are affecting the results, thus the vector $\hat{l}_i$ that are results of convolution $[\hat{g}] * k_i$ will be max-pooled to produce one number. In details, from given sequence of character

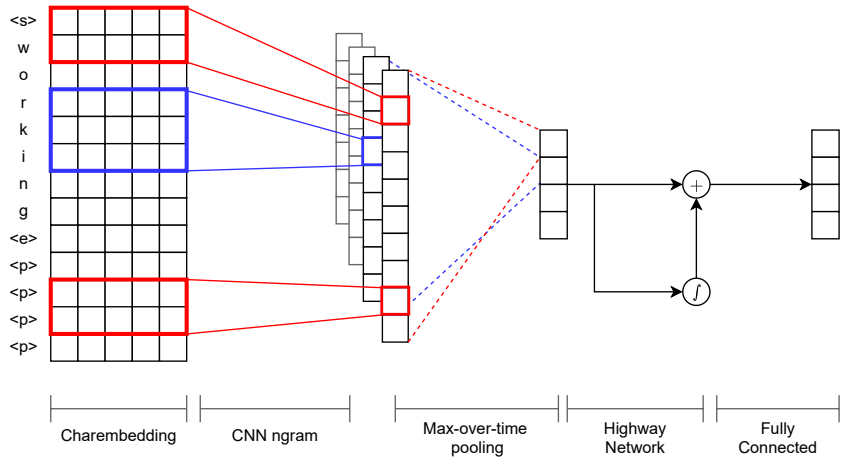

Figure 4.3: OOV Inferencing Model

embedding $[\hat{g}]$, only gram that produces the highest value when convoluted by using kernel k_i will be processed. Since, there are $|K|$ number of filter, $|K|$ number of grams will be considered to be important to the results. Furhtermore, by using several window sizes for n-grams (bigram, trigram, etc.), more features will be able to be learned.

4.1.2 Embedding Generation

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

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

On figure 4.3, starting and ending token is added at the beginning and the end of the word respectfuly. Furthermore, padding token is added if the input word is shorter than the length of the input size of the model. The padding token is a zero vector $\vec{0}$ and $\vec{0} * k = \vec{0}$ for any k. This is to ensure that part of input that got padded does not goes through maxpool layer since only grams

that has highest value can goes through the next layer and minimum value of ReLU is 0.

4.1.3 Error and Backpropagation

The output of the neural network $\tilde{e}$ then compared with the original embedding e to learn new parameters for the neural network using mean squard error function as shown on 4.3

$$Error = \frac{1}{2}\|e - \tilde{e}\|^2 \tag{4.3}$$

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

4.2 Measuring Performance on Donwstream Tasks

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

4.2.1 Part-of-Speech Tagging

Part-of-speech tagging or POStagging is a task of classifying words in sentence or corpus based on the grammatical usage of the word {cite}, for example: verb, noun, adverb, etc. Given sentence $S = \{w \in \mathcal{V}|((w_1, t_1), (w_2, t_2), \ldots, (w_n, t_n)\}$ with its POS-tag t_i, each word w_i that exist in the vocabulary $w_i \in \mathcal{V}$ and $w_i \in S$ is transformed into embedding e_i. For the OOV, the sequence of the characters building a word transformed into sequence of character embedding $[\hat{g}]_i$ using model represented in equation 4.1 then the embedding e_i is predicted using the OOV handling model.

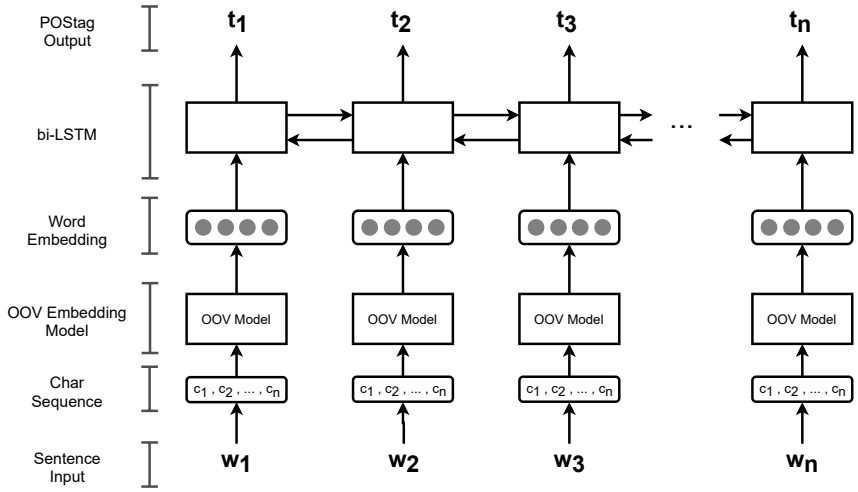

Figure 4.4: Postagging Process

The sequence of embeddings e_i then fed into bi-lstm with logsoftmax activation function at the output to get the POS-tag. To ease up computation time, adaptive logsoftmax is used (Grave et al., 2016). Instead of calculating the whole classification, the frequent and infrequent classes are separated thus there are many chance that only frequent classes needs to be calculated. The complete process of postagging process is shown in figure 4.4.

4.2.2 Word Similarity Tasks

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

equation 4.4.

$$cosine\ similarity = \frac{e_1 \cdot e_2}{\|e_1\|\|e_2\|} \tag{4.4}$$

After all of the cosine distance for all pairs are calculated, Spearman rank's correlation from the dataset and the generated embedding are calculated by using equation 4.5 and by using equation 4.6 when no tied ranks exists.

$$\rho = \frac{n\sum_{i=1}^{n} u_i v_i - \left(\sum_{i=1}^{n} u_i\right)\left(\sum_{i=1}^{n} v_i\right)}{\sqrt{\left[n\sum_{i=1}^{n} u_i^2 - \left(\sum_{i=1}^{n} u_i\right)^2\right]\left[n\sum_{i=1}^{n} v_i^2 - \left(\sum_{i=1}^{n} v_i\right)^2\right]}} \tag{4.5}$$

$$= 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)} \text{ where } d_i = u_i - v_i \tag{4.6}$$

Chapter 5

Implementation

5.1 Preparation

5.1.1 Dataset Preparation

Character Embedding

The character dictionary consists of first 128 ASCII characters with deletion for non character types (the first 32 ASCII symbols). Then the character embedding was initialized by randomizing from normal distribution with $\mu = 0$ and $\sigma = 1$.

Pretrained Word Embedding

In order to train the model, pretrained embedding is needed since the model will tries to predict embedding of from known vocabulary $v \in \mathcal{V}$ with its known embedding $w \in \mathcal{W}$. For this purpose, word2vec which trained on Google news dataset using skip-gram model containing 3 million words and phrases (name, hyperlink, connected words, etc.)(Mikolov et al., 2013). Word2vec [1] contains phrases from negative sampling and for the purpose of this research those phrases did not included. The embedding has 300 dimensional vectors. Only

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

top 40 thousands words with removal if the word contains "_" (underscore).

Another pretrained embedding is polyglot [2] (Al-Rfou et al., 2013) which contains multilingual embeddings. For this research, only English embedding that contains around 100 thousands with 60 dimensional vector representation. This pretrained embedding is also used in OOV handling model MIMICK (Pinter et al., 2017) which used as baseline model.

Dict2vec[3] is yet another word embedding that trained based on the definition of a word in dictionary (Tissier et al., 2017b). Originally, this embedding was tested using word similarity tasks, hence this embedding is used as baseline model for word similarity tasks.

Word Similarity Dataset

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

5.1.2 Programming Language and Tools

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

5.1.3 Hardware

The model was trained on the freely available Google Colaboratory[4] which gives randomized hardware specification, thus the exact hardware used cannot

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

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

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

be determined. 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 splitted up into train-val set with 80% and 20% randomized split respectively with batch size of 64. The word $w_i \in V$ becomes the input of the model and the word embedding $e_i \in \mathcal{W}$ becomes the target. The input word splitted into sequence of characters and start token and end token were added at the beginning and at the end of the word respectively. The maximum length of a word was fixed to be 20, thus words, with added starting and ending token, that has length less than the maximum length was appended with padding token. These sequence of characters then transformed into character embeddings. These character embeddings then processed by the model producing the predicted word embedding. The model was trained with learning rate $lr = 0.01$ with no dropout as dropout does not work well just as in MIMICK (Pinter et al., 2017).

The character embeddings were processed with CNN n-gram following (Kim, 2014) architecture for word n-gram. N-gram with window size $n = [2, 3, 4, 5, 6, 7]$ were used with respective kernel size $n \times d$ to simulate n-grams.

After max-over-time pooled, the vector representation then passed into fully connected network with highway network to produce the predicted word embedding. The error then calculated using Mean Squared Error as mentioned on equation 4.3 then back-propagated to update the weights.

Similar datasets are used to train MIMICK (Pinter et al., 2017) with parameters taken from the original paper. Both model trained with 1000 epochs then the error graph were compared.

5.3 Evaluating with Downstream Tasks

5.3.1 Part-of-Speech Tagging

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

5.3.2 Word Similarity

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

The baseline embeddings used for this task, dict2vec (Tissier et al., 2017b), was also tested using random OOV embedding by randomly giving OOV word random embedding and choosing the maximum results of several tries. The results then compared with if the OOV were predicted using the OOV models trained with dict2vec pretrained embedding (Tissier et al., 2017b).

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

Chapter 6

Results and Discussion

6.1 OOV model

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

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

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

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

Table 6.1: Nearest Neighbors Mimick (word2vec)

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

Table 6.2: Nearest Neighbors CNN (word2vec)

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

6.2 POStagging results

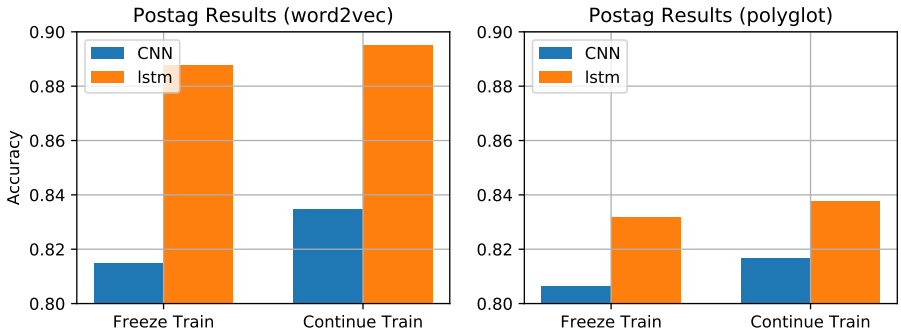

Figure 6.1: POStagging results

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

Table 6.3: Nearest Neighbors Mimick (polyglot)

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

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

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

6.3 Word Similarity

On Table 6.5, the model trained with word2vec (Mikolov et al., 2013). The predicted embedding then used for calculating the spearman rank correlation. For easier reading, all the results of word similarity task shown in the table

Table 6.4: Nearest Neighbors CNN (polyglot)

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

and the texts were multiplied by 1000. From 12 word similarity datasets, CNN model has higher Spearman correlation coefficient on 6 datasets with averaged Spearman correlation coefficient of 504.23 compared to MIMICK that achieved 501.97.

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

For baseline embedding dict2vec (Tissier et al., 2017b). the original embedding added with random embedding for OOV handling were also compared with CNN and MIMICK models. Dict2vec only able to achieve Spearman cor-

Table 6.5: Word Similarity Task Results (word2vec)

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

* multiplied by 1000

relation coefficient of 519.22. In contrast CNN model an Mimick model were able to achieve 554.05 and 551.54 on average respectively as shown on Table 6.7. This shows that both OOV models can handle OOV better than initializing embedding randomly for OOV. On top of that, from three pretrained embeddings, CNN model performs better than Mimick for word similarity task.

Table 6.6: Word Similarity Task Results (polyglot)

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

* multiplied by 1000

Table 6.7: Word Similarity Task Results (dict2vec)

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

* multiplied by 1000

Chapter 7

Conclusion

I was right all along.

7.1 What was I right about?

I was right about the following things.

7.1.1 Previous theories were wrong

People thought they understood, but they didn't.

7.1.2 My new idea is right

Of course.

Bibliography

Al-Rfou, R., Perozzi, B., and Skiena, S. (2013). Polyglot: Distributed word representations for multilingual NLP. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 183–192, Sofia, Bulgaria. Association for Computational Linguistics. 25, 29, 32

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. 6

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. 6

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

Ghosh, S. and Kristensson, P. O. (2017). Neural networks for text correction and completion in keyboard decoding. *CoRR*, abs/1709.06429. 2

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org. 7, 9, 10, 11, 12

Grave, E., Joulin, A., Cissé, M., Grangier, D., and Jégou, H. (2016). Efficient softmax approximation for gpus. *CoRR*, abs/1609.04309. 22

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. 15, 16

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. 19, 26

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2015). Character-aware neural language models. *CoRR*, abs/1508.06615. 6

Kutuzov, A. and Kunilovskaya, M. (2018). *Size vs. Structure in Training Corpora for Word Embedding Models: Araneum Russicum Maximum and Russian National Corpus*, pages 47–58. 2

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

LeCun, Y. (1989). *Generalization and network design strategies*. Elsevier. 12

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

Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., and Luis, T. (2015). Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal. Association for Computational Linguistics. 2, 6

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546. 5, 24, 31, 32

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*. 25

Pinter, Y., Guthrie, R., and Eisenstein, J. (2017). Mimicking word embeddings using subword rnns. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2, 6, 18, 19, 21, 25, 26, 27

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

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

Tissier, J., Gravier, C., and Habrard, A. (2017a). Dict2vec : Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263, Copenhagen, Denmark. Association for Computational Linguistics. 5

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. 25, 27, 32

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

Appendix A

Code

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