Université Paris Nanterre

École doctorale 139 – Connaissance, Langage, Modélisation

Polysemy resolution with word embedding models and data visualization: the case of adverbial postpositions *-ey*, *-eyse*, and *-(u)lo* in Korean

par Seongmin Mun

Thèse présentée et soutenue publiquement le 18 juin 2021

en vue de l'obtention du grade de

docteur en Traitement Automatique des Langues

sous la direction de Guillaume Desagulier

Membres du jury:

Directeur: Dr. Guillaume Desagulier	Université Paris VIII & UMR 7114, MoDyCo
Rapporteur: Prof. Iksoo Kwon	Hankuk University of Foreign Studies
Rapporteur: Prof. Laurent Prévot	Aix-Marseille Université
Examinatrice: Dr. Caroline Brun	Naver Labs Europe
Examinatrice: Prof. Iris Taravella	Université Paris Nanterre & UMR 7114, MoDyCo
Examinatrice: Prof. Delphine Battistelli	Université Paris Nanterre & UMR 7114, MoDyCo

Acknowledgements

There are many who helped me along the way on this journey. First and foremost, I am extremely grateful to god.

I would also like to thank my esteemed supervisor, Dr. Guillaume Desagulier for his valuable advice, academic guidance, and continuous support during my doctoral program. His outstanding knowledge and plentiful experience have encouraged me in all the time of my academic research, including this dissertation project. Without his help, I would never have completed this journey. It was such a great blessing to have him as my advisor.

Besides my advisor, I would like to acknowledge my dissertation committee members, Prof. Iksoo Kwon, Prof. Laurent Prévot, Prof. Iris Eshkol-Taravella, Dr. Caroline Brun, and Prof. Delphine Battistelli for their effort for my dissertation. They are all outstanding experts in the field with respect to each part of my dissertation. It was such a great honor to have them as my committee members.

I also wish to thank all the colleagues and friends for their treasured support and encouragement during this journey.

Finally, I would like to express my gratitude to my parents, my wife, and my family. Without their tremendous understanding and encouragement in

the past few years, it would be impossible for me to complete this journey.

Abstract

This dissertation reports computational accounts of resolving word-level polysemy in a lesser-studied language—Korean. Postpositions, which are characterized as multiple form-function mapping and thus polysemous in nature, pose a challenge to automatic analysis and model performance in identifying their functions. In this project, I enhance the existing word-level embedding classification models (Positive Pointwise Mutual Information and Singular Value Decomposition; Skip-Gram and Negative Sampling) with the consideration of context window, and introduce a sentence-level embedding classification model (Bidirectional Encoder Representations from Transformers (BERT)) under the scheme of Distributional Semantic Modeling. I then develop two visualization systems that show (i) relationships of the postpositions and their co-occurring words for word-level embedding models, and (ii) clusters between sentences for the sentence-level embedding model. These visualization systems have an advantage to better understand how these classification models classify the intended functions of these postpositions. Results show that, whereas the performance of the word-level embedding models is modulated by the size of training corpora containing specific functions of the postpositions, the sentence-level embedding model performs in a stable way (i.e., less affected by the corpus size) and simulates how humans recognize the polysemy involving Korean adverbial postpositions more appropriately than the word-level embedding models do.

Keywords: polysemy, natural language processing, classification, word embedding models, data visualization, Korean

Contents

List of Tables

List of Figures

List of abbreviations

The following abbreviations are used to label the linguistic terms employed in this dissertation. I follow the Leipzig glossing rules[1] for the most abbreviations used in linguistic glosses. In addition, for the POS tags used in this dissertation, I follow the Sejong POS tagging rules[2].

[1] Available at: https://www.eva.mpg.de/lingua/pdf/Glossing-Rules.pdf
[2] Available at: https://github.com/seongmin-mun/Corpora/tree/main/SPTR

Abbreviation	Label
ACC	Accusative
AGT	Agent
CNT	Content
COM	Comitative
CRT	Criterion
DECL	Declarative
DIR	Direction
EFF	Effector
EXP	Experiencer
FNS	Final State
GOL	Goal
IND	Indicative
INS	Instrument
LOC	Location
MAG	Mental Agent
NOM	Nominative
PL	Plural
PRS	Present
PST	Past
PUR	Purpose
SRC	Source
THM	Theme
TOP	Topic

Chapter 1

Introduction

The project presented in this dissertation aims to address the possible ways and limitations in applying computational approaches to word-level polysemy in a lesser-studied language, Korean. Postpositions, which are characterized as having multiple form-function mapping and thus polysemous in nature, pose a challenge to Natural Language Processing (NLP)-based analysis of these function words. In this project, I enhance the previous approaches to this task by creating classification models based on word-level embeddings—Positive Pointwise Mutual Information and Singular Value Decomposition (PPMI-SVD; Turney and Pantel, 2010) and Skip-Gram and Negative Sampling (SGNS; Mikolov et al., 2013a)—as well as sentence-level embedding—Bidirectional Encoder Representations from Transformers (BERT; Devlin et al., 2018)—under the scheme of Distributional semantic modeling (DSM). In addition, to better understand how these classification models recognize the intended functions of the postpositions, this project implements visualization systems that show the relationships of the words and sentences for each model.

1.1 Background of beginning this project

I assume that a relationship of words (represented as probabilistic information) is one core factor in understanding how language works. This assumption has led me to explore the relationship obtained from small- or large-scale corpora empirically in two major directions. One is to develop an automatic classification system, that classifies language input into appropriate categories, combined with machine learning algorithms. The other is to create a visualization system that intuitively demonstrates the relationship between words or sentences. The two directions of my research stand on statistical inferences and probabilistic approaches to language.

The reason I chose the polysemy of Korean adverbial postposition as the topic of this dissertation began with a dissatisfaction that I had as I was working on many collaborative research projects. These covered various linguistic inquiries in a lesser-studied language—Korean. I found that during the data processing, a lot of NLP-based studies removed Korean adverbial postpositions as *stop words*, which are filtered out and not used. The reason for this is that the word-level polysemy of Korean adverbial postpositions creates problems making the results difficult to interpret (e.g., Bae and Lee, 2015, Lee et al., 2015). However, unlike other Indo-European languages, postpositions play a very important role in Korean (e.g., Ahn, 1983, Hong, 1978, Jeong, 2010, Lee, 1983, Nam, 1993, Park, 1999, Song, 2014). Moreover, they have a great influence on the interpretation of the results that are obtained from NLP-based analysis (e.g., Bae et al., 2015, Shin et al., 2005). Due to their importance, previous studies have worked on resolving the polysemy of Korean adverbial postpositions by applying computational approaches (e.g., Cho and Kim, 1996, Jeong, 2010, Nam, 1993, Park, 1999, Song, 2014). Likewise, for this doctoral dissertation, I chose this topic and aim to apply computational approaches to resolve the problems that occur with the word-level polysemy of these postpositions.

1.2 Polysemy in Korean adverbial postpositions

Korean, the language of interest in this dissertation, is a Subject-Object-Verb language, which marks case information with dedicated postpositions (Sohn, 1999). Korean postpositions are divided into two categories: (i) grammatical, indicating syntactic relationships between content words and (ii) semantic, indicating specific functions according to the context of the particular sentence (Sohn, 1999). Specifically, the ones classified as semantic, can involve many-to-many mappings of form and function, and are thus polysemous (Choo and Kwak, 2008). In this dissertation, I narrow down the scope to three adverbial postpositions: *-ey*, *-eyse*, and *-(u)lo*. This is because these three are the most frequently used ones and documented in the previous studies (e.g., Cho and Kim, 1996, Jeong, 2010, Nam, 1993, Park, 1999, Song, 2014). I then determine the number of functions of each postposition based on the definition of the Sejong project[1], which is the Korean national corpus involving several Korean universities and more than five hundred Korean linguists. For example, the adverbial postposition *-ey* is interpreted as having eight major functions: location (LOC), goal (GOL), effector (EFF), criterion (CRT), theme (THM), instrument (INS), agent (AGT), and final state (FNS) (Shin, 2008). Suppose the following sentence involving the postposition *-ey* as a function of

[1] Available at: https://www.korean.go.kr

LOC (Location) as in (1)[2].

(1) 지붕 위에 구멍이 났다.
cipung wi-ey kwumeng-i na-ss-ta.
Roof top-LOC hole-NOM be.out-PST-DECL
'There is a hole on the top of the roof.'

Native speakers of Korean (or someone who has good knowledge about Korean) can easily understand the intended function of *-ey* in (1). From this, the question arises as to how a speaker or computer can understand the function of *-ey* as LOC given its various functions.

1.3 Distributional Semantic Models (DSMs)

As a possible way to answer the aforementioned question, I make use of the distributional semantic models (DSMs) in this dissertation. The fundamental idea of DSMs is that the meaning of a word is closely related to the context that is created by a group of neighboring words (Bullinaria and Levy, 2007, Turney and Pantel, 2010). This idea originates from early works in theoretical linguistics by Harris (1954) and Firth (1957). Harris (1954) states that *words that occur in similar contexts tend to have similar meanings* while Firth (1957) states that *you shall know a word by the company it keeps*. For example, *house* and *apartment* frequently occur with context words like *rent*, *bedroom*, *sale*, etc., giving evidence to computational models that *house* and *apartment* may be similar to each other. Importantly, many studies reported the strength of distributional semantic models to resolve the word-level pol-

[2] This dissertation follows the Yale romanization of Korean, which is the standard romanization of the Korean language in linguistics.

ysemy (e.g., Bae et al., 2015, Lee et al., 2015, Mun and Shin, 2020, Shin et al., 2005). Hence, I choose the distributional framework as the main concept for the computational approaches for this dissertation.

The DSMs are composed of two types of word embedding models. One is a count-based model which is sensitive to the token frequency (Jurafsky and Martin, 2019). The other is a prediction-based model that relies on the type frequency (Mikolov et al., 2013a)). In addition, previous studies have shown that the traditional word embedding models have an advantage of representing the relationship between words (e.g., Bae et al., 2015, Lee et al., 2015, Mun and Shin, 2020, Shin et al., 2005). In this dissertation, I employ a combination of Positive Pointwise Mutual Information (PPMI; Church and Hanks, 1989) and Singular Value Decomposition (SVD; Eckart and Young, 1936) as a count-based model, and Skip-Gram and Negative Sampling (SGNS; Mikolov et al., 2013a) as a prediction-based model. This is because previous studies most frequently investigated these models and reported better performance than other models for the classification task (e.g., Baroni et al., 2014, Levy et al., 2015, Melamud et al., 2016, Riedl and Biemann, 2017).

In addition to these, I introduce BERT (particularly a multi-head self-attention model), the latest and cutting-edge deep learning, as an additional type of word embedding model. This type is called *contextualized word embedding model*. Unlike the traditional word embedding models, this one assigns vectors to all words differently, even if the forms of the words are the same as each other. For this reason, this model is used more for sentence-level embedding than for word-level. Various models have been suggested for contextualized word embedding such as Embeddings from Language Models (Peters et al., 2018), Generative Pre-Training (Radford et al., 2018), and Bidirectional Encoder Representations from Transformer (BERT; Devlin et al., 2018). However, among these, BERT shows the best performance in many tasks such as translation, classification, and question-answering (Devlin et al., 2018, Tang et al., 2019). Due to this, I chose BERT as the sentence-level embedding model for the classification task to identify the intended function of a postposition in a sentence.

1.4 Visualization system

Previous NLP-based research on polysemy resolution has an issue in that they focused on enhancing the model performance to classify the functions of postpositions and they did not try to explore the relationships around postpositions (e.g., Kim et al., 2006, 2007, Kim and Ock, 2016). As stated previously, BERT achieved superior performance in many tasks (e.g., Dai and Le, 2015, Peters et al., 2018, Radford et al., 2018). However, it is somewhat unclear how BERT deals with the polysemy resolution (e.g., Clark et al., 2019, Coenen et al., 2019, Devlin et al., 2018, Tang et al., 2019). Improving the performance of classification models is undoubtedly important, but it is also important to see how the relationship between postposition and co-occurring words changes with the particular function of postposition and how the model recognizes the intended function of postpositions in the sentence.

To remedy these issues, I propose two visualization systems of the respective (chosen) models. These visualization systems have the advantage of helping to identify relationships between words and to show changes in the relationships based on the contexts where these words manifest. Moreover, these systems can help the general audience understand (i.e., how model works, how the relationship between words/sentences changes) through an informative display of outcomes from each model (e.g., Coenen et al., 2019, Mun and Lee, 2016, Mun et al., 2014).

1.5 Outline of the Dissertation

This dissertation is organized as follows: **Chapter 2** provides a review of previous studies on the three adverbial postpositions: *-ey*, *-eyse*, and *-(u)lo*, which occur frequently in language use. This chapter also discusses the issues in previous studies, which focused mostly on improving the classification accuracy and did not pay attention to the environment around postpositions. **Chapter 3** provides an overview of the algorithms of how to calculate and apply the word-level embedding classification models (Positive Pointwise Mutual Information and Singular Value Decomposition; Skip-Gram and Negative Sampling) with the consideration of the context window. **Chapter 4** introduces three parts in relation to the use of the word-level embedding classification models: (i) methodological details, (ii) a hand-coded corpus, which tagged intended functions of postpositions manually, and (iii) design of visualization. **Chapter 5** reports on the results of the word-level embedding classification models and the visualization in relation to the three research questions, and the issues of the models. **Chapter 6** provides the history of how BERT was born and an overview of the algorithms of how to calculate and apply it. **Chapter 7** introduces the methodological details of the sentence-level embedding classification models and design of the BERT-based visualization. **Chapter 8** reports on the results of the models and the visualization by using BERT in relation to the two research questions. **Chapter 9** provides the overall discussion of this dissertation. Finally, **Chapter 10** provides the conclusions of this dissertation and suggestions for future works.

Chapter 2

NLP reaserch on adverbial postpositions in Korean: *-ey*, *-eyse*, and *-(u)lo*

Korean, a Subject-Object-Verb language, is agglutinative in that multiple postpositions or affixes with dedicated forms and meanings are attached to the stem of nominals or predicates. A postposition is a function word providing grammatical information to words it is attached (Sohn, 1999). Korean postpositions are divided into two categories. One category includes grammatical case markers such as nominative *-i/ka*, accusative *-(l)ul*, and possessive *-uy*, indicating syntactic relationships between content words. The other category consists of semantic postpositions that express adverbial functions, indicating specific functions such as locational and instrumental. Many of the semantic postpositions are polysemous due to their many-to-many mapping of form and function, which accompanies functional ambiguity. This chapter summarizes three adverbial postpositions, *-ey*, *-eyse*, and *-(u)lo*, which occur frequently in language use and thus frequently explored in studies on Korean postposition (e.g., Cho and Kim, 1996, Jeong, 2010, Nam, 1993, Park, 1999, Song, 2014), focusing on the multiple functions involving each form.

2.1 Previous research on polysemy of *-ey*, *-eyse*, and *-(u)lo*

2.1.1 *-ey*

The Standard-Korean dictionary (1999) defines the adverbial postposition *-ey* as a postposition that gives the preceding word the function of the *location*. Based on this definition, the primary function of *-ey* is *location*. However, this definition is neither accurate nor specific enough to capture all the essential functions of *-ey*. For this reason, previous research has investigated the functions of *-ey* in two lines. One line of research is concerned with various functions of *-ey* obtained through the semantic relationship between it and its noun or predicate (e.g., Ahn, 1983, Hong, 1978, Lee, 1983). The other line explores the basic functions of *-ey* (e.g., Jung, 1988, Lee, 1981). Some researchers also propose their own claims for the types of functions involving *-ey*. For example, Cho and Kim (1996) classified 10 types. Nam (1993) argued that the relationship between a (pro-)noun and a predicate combined with a postposition is important to determine its function, which yielded 14. From a more practical perspective, Song (2014) suggested as the main function an indication of a location or movement of a physical target. He also explained that the function could be extended regarding as scope, situation, criteria, time, goal, method, and reason. Together, there is no clear consensus as to the precise number/type of functions involving *-ey*.

To determine the number of functions of each postposition, this dissertation puts special emphasis on eight major functions of *-ey*, which are frequently attested in the Sejong dictionary. These are also commonly mentioned in the previous studies, with *location* and *goal* occupying the majority of the occurrences. Of the functions of *-ey* defined by the Sejong project[1] (Table 2.1), I selected the eight most frequent as the main one of *-ey*. Note that, although I classify the functions into designated types, each one is rather flexible due to the difficulty in creating standard definitions of individual functions that everyone agrees with (Kang and Park, 2003).

[1]Available at: https://www.korean.go.kr

Table 2.1: Functions of -*ey* and its frequency in Sejong dictionary (adapted from Sejong Electronic Dictionary)

Function	Abbreviation	Use
Location	LOC	1,328
Goal	GOL	665
Effector	EFF	150
Criterion	CRT	124
Theme	THM	58
Instrument	INS	17
Agent	AGT	13
Final State	FNS	11
Experiencer	EXP	5
Source	SRC	3
Mental Agent	MAG	2
Companion	COM	2
Content	CNT	2
Purpose	PUR	2

Location (LOC) is a function that represents the spatial domain where an event occurs. In the following sentence (i.e. this sentence is extracted from the file *V-phamwuthita* in the Sejong Electronic Dictionary), -*ey* is playing the same role as *in* in English.

(1) -*ey* as LOC (location)

그는	온종일	서재에	파묻혀	지낸다.
ku-nun	oncongil	secay-ey	phamwut-hi-e	cinay-n-ta.
He-TOP	all day	study.room-LOC	bury.in-PSV-PRS	be-PRS-DECL

‘He is buried in his study room all day.’

Goal (GOL) is a function that indicates the preceding word is where the object reaches. In the following sentence (i.e., this sentence is extracted from the file *V-naylyekkochita* in the Sejong Electronic Dictionary), *-ey* is playing the same role as *to* in English.

(2) *-ey* as GOL (goal)

철수가	던진	칼이	땅바닥에
Chelswu-ka	tenc-i-n	khal-i	ttangpatak-ey
Chelswu-TOP	throw-CST-PRS	knife-NOM	ground-GOL

내리꽂혔다.
naylyekkoc-hi-ess-ta.
stick-PSV-PST-DECL

'The knife that Chelswu threw stuck to the ground.'

Effector (EFF) is a function that indicates that the preceding word influences the theme to act or change when an event occurs. In the following sentence (i.e., this sentence is extracted from the file *V-kentultayta* in the Sejong Electronic Dictionary), *-ey* is playing the same role as *by* in English.

(3) *-ey* as EFF (effector)

문들이	거센	바람에	모두	건들댄다.
mwun-tul-i	keseyn	palam-ey	motwu	kentultay-n-ta.
door-PL-NOM	strong	wind-EFF	all	sway-PRS-DECL

'The doors all sway by the strong wind.'

Criterion (CRT) is a function that indicates that the preceding word is the degree of value of the theme. In the following sentence (i.e., this sentence is extracted from the file *V-nakchalhata* in the Sejong Electronic Dictionary), *-ey* is playing the same role as *for* in English.

(4) -*ey* as CRT (criterion)

영호는	20만원에	모니터를	낙찰했다.
Yenghuy-nun	20manwen-ey	monithe-lul	nakchalhay-ss-ta.
Yenghuy-TOP	200,000 won-CRT	moniter-ACC	sell-PST-DECL

'Yenghuy sold the monitor (to a bidder) for 200,000 won.'

Theme (THM) is a function that makes the preceding word as an entity that is affected by the action of the verb. In the following sentence (i.e., this sentence is extracted from the file *V-hekicita* in the Sejong Electronic Dictionary), -*ey* is playing the same role as *for* in English.

(5) -*ey* as THM (theme)

현대인들은	모두	참된	지식에
hyentayin-tul-un	motwu	chamtoy-n	cisik-ey
modern.people-PL-TOP	all	true-REL	knowledge-THM
허기져있다.			
hekicye-iss-ta.			
hungry-PRS-DECL			

'All modern people are hungry for true knowledge.'

Instrument (INS) is a function that indicates the preceding word engages in an action or a process as a tool. In the following sentence (i.e., this sentence is extracted from the file *V-nokita* in the Sejong Electronic Dictionary), -*ey* is playing the same role as *in* in English.

(6) *-ey* as INS (instrument)

그	어린	소년은	화롯불에	손을	녹이고
ku	eli-n	sonye-nun	hwalospwul-ey	son-ul	nok-i-ko
That	young-REL	boy-TOP	fire-INS	hand-ACC	melt-CST-and

있었다.
iss-ess-ta.
be-PST-DECL

'The young boy was using the fire to warm his hands.'

Agent (AGT) is a function that makes the preceding word as an entity that intentionally carries out the action of the verb. In the following sentence (i.e., this sentence is extracted from the file *V-cecitoyta* in the Sejong Electronic Dictionary), *-ey* is playing the same role as *by* in English.

(7) *-ey* as AGT (agent)

가두	진출이	경찰에	저지되었다.
katwu	cinchwul-i	kyengchal-ey	ceci-toy-ess-ta.
street	go.out-NOM	police-AGT	stop-PSV-PST-DECL

'By going out to the street was stopped by the police.'

Final state (FNS) is a function that allows the preceding word to present the current state. In the following sentence (i.e., this sentence is extracted from the file *V-chwuchenhata* in the Sejong Electronic Dictionary), *-ey* is playing the same role as *as* in English.

(8) *-ey* as FNS (final state)

김교수는	조교에	박군을	추천했다.
kimkyoswu-nun	cokyo-ey	park-kwun-ul	chwuchenhay-ss-ta.
professor.Kim-TOP	assistant-FNS	Park-Mr-ACC	recommend-PST-DECL

'Professor Kim recommended Park as an assistant.'

2.1.2 *-eyse*

The Standard-Korean dictionary (1999) defines the adverbial postposition -*eyse* as a postposition indicating that the preceding word is a location where an action is being made. -*eyse* has fewer functions than the other two postpositions -*ey* and -*(u)lo* (Choo and Kwak, 2008). However, the frequency of its use is equally high compared to that of the others (e.g., Cho and Kim, 1996, Song, 2014). Researchers generally agree with the primary function as the location which engages in departure of action (e.g., Cho and Kim, 1996, Park et al., 2000, Song, 2014) and the Sejong corpus also demonstrates the same tendency. As Table 2.2 shows, the two functions (*source* and *location*) are overwhelmingly more frequent than the others.

Table 2.2: Functions of *-eyse* and its frequency in Sejong dictionary (adapted from Sejong Electronic Dictionary)

Function	Abbreviation	Use
Source	SRC	487
Location	LOC	197
Agent	AGT	6
Goal	GOL	4
Theme	THM	4
Criterion	CRT	4
Direction	DIR	2
Final State	FNS	1

This dissertation follows this skewedness in frequency, focusing on these two functions. Therefore, *-eyse* has only two functions, *source* (9) and *location* (10). Source (SRC) is a function that indicates the origin of an action, the point at which the action is initiated. In the following sentence (i.e., this sentence is extracted from the file *V-ppopaollita* in the Sejong Electronic Dictionary), *-eyse* is playing the same role as *from* in English.

(9) *-eyse* as SRC (source)

광부들이	바다에서	석유를	뽑아올린다.
kwangpwutul-i	pata-eyse	sekyu-lul	ppopaoll-i-n-ta.
miner-PL-NOM	sea-SRC	oil-ACC	pull-CST-PRS-DECL

'Miners pull oil from the sea.'

The definition of the location (LOC) is the same as described for *-ey* (location). In the following sentence (i.e., this sentence is extracted from the file *V-thayenata* in the Sejong Electronic Dictionary), *-eyse* is playing the same role as *in* in English.

(10) -*eyse* as LOC (location)

철수는	서울에서	태어났다.
Chelswu-nun	sewul-eyse	thayena-ss-ta.
Chelswu-TOP	seoul-LOC	born-PST-DECL

'Chelswu was born in Seoul.'

2.1.3 *-(u)lo*

The Standard-Korean dictionary (1999) defines the adverbial postposition -*(u)lo* as a postposition indicating the direction of movement, stating that the key concept to understanding the functions of -*(u)lo* involves direction. However, it is somewhat vague to pinpoint the essential and typical functions by using this concept. In fact, this postposition has a variety of functions, and many researchers have proposed different viewpoints on this issue. To illustrate, Park (1999) claimed that the central function is *instrumental*, and that there are 9 more functions, such as *path*, *direction*, *point of direction*, *time*, *state change*, *qualification*, *material*, *cause*, and *manner*. In contrast, Jeong (2010) puts the directional function at the center of the various ones and explained the relationship between the core function and the extended ones.

The classification in the Sejong project (Table 2.3) is somewhat different from these two studies, stating that there are six major functions of -*(u)lo*, with the top three (*final state*; *instrumental*; *directional*) occupying more than 80 per cent of the entire use. Because the Sejong corpus is widely used in studies on Korean (e.g., Kang and Park, 2003, Kim et al., 2007, Park and Cha, 2017, Shin et al., 2005), and this dissertation also employs this corpus for investigation, I follow the classification it provides.

Table 2.3: Functions of *-(u)lo* and its frequency in Sejong dictionary (adapted from Sejong Electronic Dictionary)

Function	Abbreviation	Use
Final State	FNS	857
Instrument	INS	561
Direction	DIR	324
Effector	EFF	38
Criterion	CRT	22
Location	LOC	9
Content	CNT	6
Source	SRC	5
Theme	THM	5
Experiencer	EXP	1
Agent	AGT	1

The definition of the final state (FNS) is the same as described in (8) above. In the following sentence (i.e., this sentence is extracted from the file *V-chopingtoyta* in the Sejong Electronic Dictionary), *-(u)lo* is playing the same role as *as* in English.

(11) *-(u)lo* as FNS (final state)

그는	대표	강사로	초빙되었다.
ku-nun	tayphyo	kangsa-lo	choping-toy-ess-ta.
He-TOP	representative	lecturer-FNS	invite-PSV-PST-DECL

'He was invited as a representative lecturer.'

The definition of instrument (INS) is the same as described in (6) above. In the following sentence (i.e., this sentence is extracted from the file *V-kamkita* in the Sejong Electronic Dictionary), *-(u)lo* is playing the same role as *with* in English.

(12) -*(u)lo* as INS (instrument)

전선이	연줄로	감겼다.
censen-i	yencwul-lo	kam-ki-ess-ta.
wire-NOM	connection.wire-INS	wind-PSV-PST-DECL

'The wire wound around with the connection wire.'

Direction (DIR) is a function to indicate the direction of the theme's movement. In the following sentence (i.e., this sentence is extracted from the file *V-talanata* in the Sejong Electronic Dictionary), -*(u)lo* is playing the same role as *toward* in English.

(13) -*(u)lo* as DIR (direction)

범인은	어두운	골목으로	달아났다.
pemin-un	etwuwun	kolmok-ulo	talana-ss-ta.
criminal-NOM	dark	alley-DIR	flee-PST-DECL

'The criminal fled into a dark alley.'

The definition of effector (EFF) is the same as described in (3) above. In the following sentence (i.e., this sentence is extracted from the file *V-koylowehata* in the Sejong Electronic Dictionary), -*(u)lo* is playing the same role as *due to* in English.

(14) -*(u)lo* as EFF (effector)

환자가	위암으로	매우	괴로워하고	있습니다.
hwanca-ka	wiam-ulo	maywu	koyloweha-ko	iss-supni-ta.
patient-NOM	stomach.cancer-EFF	very	suffer-and	be-HON-DECL

'The patient is suffering greatly due to stomach cancer.'

The definition of criterion (CRT) is the same as described in (4) above. In the following sentence (i.e., this sentence is extracted from the file *V-paychatoyta* in the Sejong Electronic Dictionary), -*(u)lo* is playing the same role as *at* in English.

(15) -*(u)lo* as CRT (criterion)

적당한	시간	간격으로	배차되었다.
cektangha-n	sikan	kankyek-ulo	paycha-toy-ess-ta.
appropriate-REL	time	interval-CRT	arrange-PSV-PST-DECL

'It was arranged at appropriate time intervals.'

The definition of LOC is the same as described in (1) above. In the following sentence (i.e., this sentence is extracted from the file *V-apsonghata* in the Sejong Electronic Dictionary), -*(u)lo* is playing the same role as *to* in English.

(16) -*(u)lo* as LOC (location)

경찰이	피해자를	검찰로	압송했다.
kyengchal-i	phiuyca-lul	kemchal-lo	apsonghay-ss-ta.
police-NOM	suspect-ACC	prosecution-LOC	transport.do-PST-DECL

'The police transported the suspect to the prosecution.'

2.2 Previous NLP research on adverbial postpositions

Studies on word-level polysemy in Korean have focused mainly on categorizing different meanings/functions of polysemous words for the essential interpretation of linguistic phenomena (e.g., Ahn, 1983, Hong, 1978, Lee, 1983, Maeng, 2016). Researchers working on computational linguistics in Korean have followed this trend and developed systems that automatically classify and recognize these multiple meanings/functions involving the words in order to deal with linguistic items in an easier and more efficient way (e.g., Bae and Lee, 2015, Kang and Park, 2003, Kim et al., 2007, Kim and Ock, 2015). Previous studies on automatic classification of functions involving Korean adverbial postpositions have employed two methods according to the types of information used: exclusive use of case frames in dictionaries, and heavy use of probabilistic information about grammatical relations from existing corpora.

2.2.1 Use of case frames in dictionaries only

The first method concerns the application of case frames (i.e., semantic relationships between words in a sentence), which are pre-defined and stored in a separate document, to a dictionary (i.e., a document that explains case frames that described manually according to the meanings of the words). Table 2.4 presents a summary of studies on automatic classification by using case frames in dictionaries only.

Table 2.4: Summary of previous studies on automatic classification of meanings/functions involving Korean adverbial postpositions by using case frames in dictionaries only

Study	Corpus type	Data size	Accuracy
Bae et al. (2014)	Korean PropBank	5,771 sentences	0.62
Jo et al. (2015)	Korean PropBank	1,000 sentences	0.80
Kang and Park (2003)	Sejong corpus and Kadokawa synonyms	208,088 ecel	0.88
Kim and Ock (2015)	UPropBank	65,529 sentences	0.72
Park and Kim (1998)	School textbook (elementary and middle)	2,012 sentences	0.81
Park and Cha (2017)	Sejong corpus	14,335 sentences	0.77

Kim and Ock (2015) created *UPropBank*, a case frame dictionary, based on the standard Korean dictionary and established a semantic role labeling system by using the frequency of words and case frames. To determine the functions of postpositions, 59,257 out of 65,529 sentences were used as a training set and the remaining 6,272 sentences were used as a test set for measuring model performance. The performance was measured in four ways: (i) using case frames only, (ii) using case frames and information of particles, (iii) using case frames and information about particles and predicates, and (iv) using case frames and information about particles and predicates but excluding preceding predicates. The results showed that, when only case frames were used, the accuracy rate was 0.78, which is a high accuracy rate compared to other methods.

Park and Cha (2017) conducted a similar study by combining various methods such as case frames, information of nouns and predicates, and information of clusters. Unlike Kim and Ock (2015), they found that the accuracy for semantic role labeling was the highest (at the rate of 0.79) when all the information was considered. The results of the two studies differed because the corpus used in the study by Kim and Ock (2015) and the one used by Park and Cha (2017) were different. In addition, the information used in the studies differed. Kim and Ock (2015) used information such as case frames, particles, predicates, whereas Park and Cha (2017) used case frames, nouns, predicates, clusters in their studies was different.

This line of research has shown high accuracy in determining the functions of postpositions, with the advantage that the semantic-functional characteristics of these being determined by calculating the similarities between grammatical structures and case frames being defined manually by the researchers (e.g., Kim and Ock, 2015, Park and Cha, 2017). However, creating accurate/appropriate case frames for this case frame-based method consumes considerable resources and time. This method also has the problem in that only the information described in the case frame dictionary is applicable to automatic processing, which leads a model to achieve a low coverage rate[2] for the data (e.g., Kang and Park, 2003, Kim and Ock, 2015, Park and Kim, 1998).

[2]This refers to how much the data is explained by the model.

2.2.2 Use of probabilistic information from existing corpora

The other method, using probabilistic information about grammatical relations from existing corpora, utilizes annotated corpus data with individual meanings and/or functions of a word (mostly by hand) and applies statistical learning techniques to classifying the functions. A case frame-based model is not applicable to data if the information in the model and those in the data do not match. For example, suppose the case frame involving the postposition *-(u)lo* as a function of FNS as in (17)[3].

(17) THM-i/JKS FNS-(u)lo/JKB ppophy/VV

We apply this case frame to two sentences ((18)-(19)).

(18)

hyeng/NNG-i/JKS	tayphyo/NNG-(u)lo/JKB	ppophy/VV-ass/EP-ta/EF./SF
hyeng-i	tayphyo-ulo	ppophy-ass-ta.
brother-THM	representative-FNS	elect-PST-DECL

'My brother was elected as a representative.'

(19)

tayphyo/NNG-(u)lo/JKB	hyeng/NNG-i/JKS	ppophy/VV-ass/EP-ta/EF./SF
tayphyo-ulo	hyeng-i	ppophy-ass-ta.
representative-FNS	brother-THM	elect-PST-DECL

'My brother was elected as a representative.'

In the case of (18), the *i/JKS*, *-(u)lo/JKB*, and *ppophy/VV* used in the case frame are all attested, and the word order is the same as what the case frame represents, thus is applied reliably. In contrast, in (19), the elements are attested in the sentence, but the word order does not match, thus impossible to apply. For this reason, if only case frame information is used in a model,

[3]The abbreviations of part of speech are available at: https://github.com/seongmin-mun/Corpora/tree/main/SPTR

sentences that do not follow the precise characteristics of the case frames cannot be processed. However, a probabilistic information-based model can be applied even though a mismatch arises between the model and the data with respect to key information (e.g., Bae et al., 2015, Shin et al., 2005). This probabilistic information-based method thus achieves a higher rate of coverage than the case frame-based method (e.g., Bae and Lee, 2015, Lee et al., 2015). Table 2.5 presents a summary of studies on automatic classification of functions involving postpositions by using probabilistic information.

Table 2.5: List of studies on automatic classification of meanings/functions involving Korean adverbial postpositions by using probabilistic information from existing corpora

Study	Corpus type	Data size	Case frame?	Probabilistic method?	Accuracy
Bae and Lee (2015)	Korean PropBank	4,882 sentences	No	Yes (Bidirectional Long Short-Term Memory model and Recurrent Neural Network)	0.78
Bae et al. (2015)	Korean PropBank	4,882 sentences	No	Yes (Structural Support Vector Machine and Feed-Forward Neural Network)	0.75
Kim et al. (2007)	Sejong corpus	34,371 sentences	Yes	Yes (Self-training algorithm)	0.83
Kim et al. (2006)	Sejong corpus	58,238 sentences	Yes	Yes (Bootstrapping algorithm)	0.88
Kim and Ock (2016)	UPropBank and UWordMap	23,966 sentences	Yes	Yes (Conditional Random Fields Model)	0.83
Lee et al. (2015)	Korean PropBank	4,882 sentences	No	Yes (Structural Support Vector Machine)	0.77
Shin et al. (2005)	Sejong corpus	Unclear (42,332 files)	Yes	Yes (Support Vector Machine)	0.71

Lee et al. (2015) employed an SVM to propose a semantic role labelling system. In the study, 4,096 sentences were used for learning and 786 sentences were used for test, which obtained an accuracy of 0.77 for classification. Bae and Lee (2015) proposed a method using Bidirectional Long Short-Term Memory models, Recurrent Neural Networks and Conditional Random Field as probabilistic method. In this study, several types of information were used for learning, such as a predicate, the target word, words before and after the target word, and Part-Of-Speech information. The result showed an accuracy of 0.78 in classifying functions of postpositions. Overall, probability-based methods achieved a high level of accuracy and coverage rate. Nevertheless, this accuracy is often affected by data size and/or genre(s).

Shin et al. (2005) proposed an alternative method that complemented shortcomings of both methods by using case frames in dictionaries and probabilistic information together. In the study, they used the case frame information first in order to determine the functions of postpositions; and if the input sentence was not applicable to use, they then employed the SVM algorithm. The result showed 0.71 when both methods were applied together, rather than only one or the other.

Although a few more studies used both methods in a hybrid manner to determine the functions of postpositions (e.g., Kim et al., 2006, 2007, Kim and Ock, 2016), they generally failed to address polysemy under linguistic perspectives, ignoring important questions such as how postpositions relate to the co-occurring words. One reason for this limitation is that previous research often lacked clear motivation that connected computational techniques and investigation of language phenomena, which made it harder to apply their approaches to addressing linguistic inquiries.

2.3 Issues of NLP research on polysemy resolution

Previous research has attempted to identify functions of postpositions using grammatical/semantic relationships between the postpositions and their neighbors in a sentence. However, they focused mostly on improving the accuracy of classifying the functions and did not pay attention to the environment around postpositions, such as co-occurring words, which generate a cluster centering around the postposition. From a linguistic perspective, a relationship of interlinked clusters of words is undoubtedly a valuable language resource because it shows how polysemy is interpreted through them. In this regard, the distributional semantic models (DSMs; Baroni et al., 2014), which argue that a word meaning is closely tied to a context that is created by a group of neighborhood words, draws attention to the computational understanding in human language (e.g., Bullinaria and Levy, 2007, Turney and Pantel, 2010).

In computational linguistics, the DSMs are generally used to investigate the meaning of a word in a sentence (see explanation in Chapter 3). They convert contextual information obtained through the words surrounding a target word into vectors (see explanation in Chapter 3). Based on this information, various computational techniques can be applied to these vectors in order to measure the semantic similarity of the word (e.g., Clark, 2015, Erk, 2012, Turney and Pantel, 2010). The model represents each word as a dimensional vector of the number of occurrence and the vectors close to each other appear to be semantically relevant (Levy et al., 2015). In addition, by visualizing the relationship of clusters representing the embedded words, we can intuitively identify the relationships of words.

Based on the DSMs, previous studies have been conducted to identify the meaning of words and their relationships with the surrounding words (e.g., Desagulier, 2014, Hilpert, 2016, Li et al., 2015).

Hilpert (2016) is one representative study in this respect. He conducted a diachronic corpus-based study of the English modal auxiliary *may*, focusing on changes in its collocational preferences during the past 200 years, and displayed a visualization of embedded word cluster. The point of this paper was the argument that constructional views need to consider the mutual associations between modal auxiliaries and the lexical elements with which they occur. In the study, 50-million-word samples of the Corpus of Contemporary American English (COCA; Davies, 2008) were used as a corpus and the distribution of 250 verbs that occur frequently with *may* was visualized over time applying word embeddings (Positive Pointwise Mutual Information; Church and Hanks, 1989). Results showed that *say* and *see* were important verbs in the period of 1800s-1860s, but their importance flattened out as time elapsed. It also showed that the use of *depend*, *exist*, *involve*, *enable*, and *indicate* was expanding and increasing over time. His research suggests that DSMs allow us to see changes of the relationship between one word and the co-occurring words by way of changes of clusters that these words produce.

However, some crucial questions about the DSMs remain unanswered. One relates to the effectiveness of various techniques of word embeddings on converting contextual information into vectors. The DSMs comprise of two types of word embeddings: count-based model (e.g., Singular Value Decomposition (SVD); Eckart and Young, 1936) and prediction-based model (e.g., Skip-Gram and Negative Sampling (SGNS); Mikolov et al., 2013a). Several studies have investigated the differences between several word embedding models (e.g., Baroni et al., 2014, Levy et al., 2015, Melamud et al., 2016, Riedl and Biemann, 2017).

For instance, Levy et al. (2015) suggested a study comparing four word embedding models; Positive Pointwise Mutual Information (PPMI, Church and Hanks, 1989, Dagan et al., 1995, Niwa and Nitta, 1994) and SVD as *count-based* word embeddings, and SGNS and Global Vectors for Word Representation (Pennington et al., 2014) as *prediction-based* embeddings. In their study, the English Wikipedia (August 2013 dump), pre-processed by removing non-textual elements, sentence splitting, and tokenization, was used as corpus. It contained 77.5 million sentences, spanning 1.5 billion tokens. The evaluation for the performance of the model was divided into *Word Similarity* and *Analogy*. For *Word Similarity*, datasets were adapted from the similarity score of human-assigned word pairs, such as WordSim353 (Finkelstein et al., 2002) and SimLex-999 (Hill et al., 2014). The word vectors were evaluated by ranking the pairs according to their cosine similarities and by calculating the correlation (Spearman's) with the ratings of humans. For *Analogy*, the correct answer data were divided into semantic and grammatical phrases, such as MSR's analogy dataset (Mikolov et al., 2013c) and Google's analogy dataset (Mikolov et al., 2013b). The accuracy of the correct answer was measured by using the match between queries recorded in the analogy datasets and answers obtained by each model. Results showed that SVD outperformed other models in the *Word Similarity* task, whereas SGNS yielded the best result in MSR datasets and PPMI dominated Google dataset in the *Analogy* task. Based on these results, it was recommended that SGNS and SVD with the window of size 4 is best in setting the hyperparameters for the embedding models.

Another unanswered question relates to the role of the context window size—a range of words surrounding a target word, which affects the determination of the characteristics of the word (Lison and Kutuzov, 2017)—in the calculation for word embeddings. It is important to consider the context window in the calculation because the size affects how the relationship between the target word and the surrounding words are represented. Previous studies have reported the effect of context window sizes on model performance (e.g., Bullinaria and Levy, 2007, 2012, Garcia and Gamallo, 2011, Henestroza Anguiano and Denis, 2011, Hung and Yang, 2009, Levy and Goldberg, 2014, Levy et al., 2015, Lison and Kutuzov, 2017, Peirsman et al., 2007).

To illustrate, Bullinaria and Levy (2007) showed that the semantic vector of Pointwise Mutual Information values achieved the best performance when the context window size was one. In contrast, Han et al. (2013) showed that context windows as large as 16 to 32 achieved high performance in dealing with polysemy. Despite a good amount of research on some major languages in exploring this issue (e.g., English: Bullinaria and Levy (2007), French: Henestroza Anguiano and Denis (2011), German: Peirsman et al. (2007), Spanish: Garcia and Gamallo (2011), Chinese: Hung and Yang (2009)), previous research still fell short of ensuring generalizability of methodology across languages. In particular, they did not address issues of word embeddings and context window size in searching for the appropriate clusters when it came to polysemy interpretation in languages typologically different from the researched languages, such as Korean.

2.4 Summary of the Chapter

Among the adverbial postpositions in Korean, *-ey*, *-eyse*, and *-(u)lo* have been studied actively because of the frequency of use in the language and the various functions of each postposition. In this dissertation, the specific functions of these postpositions are based on the Sejong dictionary. For *-ey*, there are eight functions, with LOC and GOL occurring most frequently. For *-eyse*, there are two major functions, SRC and LOC. And for *-(u)lo*, there are six major functions, with FNS, INS, and DIR occupying the majority of the occurrences.

The NLP studies on automatic classification of functions involving these postpositions are divided into two approaches: exclusive use of case frames in dictionaries, and major use of probabilistic information about grammatical relations from existing corpora. Recently, studies have been proposed that have increased the performance of automatic classification by merging these two types of approaches and classifying the functions of adverbial postpositions. However, these studies only cared about the accuracy of classification and did not pay attention to the environment between postpositions and surrounding words, which generates a cluster centering around the postposition.

By paying more attention to the environment between postpositions and surrounding words, DSMs (Baroni et al., 2014) are drawing attention to the computational understanding in human language, which allows us to obtain a cluster of interlinked words. However, they have two crucial aspects to consider: choice of word embedding models and context window sizes.

In this dissertation, I adopt the idea that DSMs provide clusters between the target word and the co-occurring words, and use this idea to identify environments between the three Korean adverbial postpositions and their surrounding words when the functions change.

Chapter 3

PPMI-SVD and SGNS for polysemy resolution

Linguists have benefitted from adopting quantitative approaches in addressing linguistic inquiries (Gries, 2015) by making full use of automatic processing techniques provided by computers (Turney and Pantel, 2010). One recent trend of quantitative studies is employing statistical learning (e.g., Bae and Lee, 2015, Bullinaria and Levy, 2007, Desagulier, 2014, Hilpert, 2016, Kim et al., 2006, 2007, Kim and Ock, 2016, Levy and Goldberg, 2014, Levy et al., 2015, Li et al., 2015). Among the various methods, the DSMs have drawn the attention of many researchers who aim at understanding word meaning (e.g., Baroni et al., 2014, Bullinaria and Levy, 2007). This is because the results generated through DSMs can be used to understand and visualize how the target word is interpreted and how its meaning changes based on the co-occurring words (e.g., Hilpert, 2016, Li et al., 2015).

3.1 Distributional Semantic Models

The distributional hypothesis (Firth, 1957, Harris, 1954) which is the idea behind the DSMs states that a word meaning is closely related to the context created by a group of neighboring words (Baroni et al., 2014). In its actual application, the DSMs convert contextual information that is obtained through the words surrounding a target word into vectors. They then apply machine learning algorithms to these vectors in order to measure the semantic similarity of the word (Clark, 2015, Erk, 2012, Turney and Pantel, 2010). The DSMs have two types of word embedding: count-based (e.g., Singular Value Decomposition (SVD): Eckart and Young, 1936) and prediction-based (e.g., Skip-Gram and Negative Sampling (SGNS): Mikolov et al., 2013a).

This dissertation uses a combination of Positive Pointwise Mutual Information (PPMI: Church and Hanks, 1989) and SVD, and SGNS. These are the most frequently investigated models from previous studies (e.g., Baroni et al., 2014, Levy et al., 2015, Melamud et al., 2016, Riedl and Biemann, 2017). The following sections outline each technique, with an emphasis on how it works and is applied to this dissertation.

3.2 Count-based model

The count-based model learns vocabulary based on a corpus and models each word by counting the number of times each word appears (Bullinaria and Levy, 2007). The most fundamental task for this model is to convert corpus data into vectors, using several ways such as a word-word co-occurrence matrix (e.g., Davies, 2015, Hilpert, 2016) and a term-document matrix (e.g., Salton, 1971, Turney and Pantel, 2010). Researchers choose the particular way of vectorizing according to the purpose of their study (Jurafsky and Martin, 2019). For example, a word-word co-occurrence matrix is used to see the relationship between words, while a term-document matrix is used to see the relationship between documents (Jurafsky and Martin, 2019). This dissertation utilizes a word-word co-occurrence matrix to check the relation between postposition and its co-occurring words.

3.2.1 Word-word co-occurrence matrix and context window size

A word-word co-occurrence matrix is computed by counting instances in which two or more words occur together in a given corpus (Jurafsky and Martin, 2019). For example, suppose the following corpus extracted from the Sejong corpus involving the postposition *-(u)lo* as a function of DIR (Direction) as in ((1)-(3)):

(1) pang_07/NNG -(u)lo/JKB ka/VV ass/EP ta/EF ./SF

pang-ulo ka-ass-ta.
room-DIR go-PST-DECL

'(I) went to the room.'

(2) pakk/NNG -(u)lo/JKB nao/VV ass/EP ta/EF ./SF

pakk-ulo na-o-ass-ta.
outside-DIR be.out-come-PST-DECL

'(I) went out to the outside.'

(3) aph/NNG -(u)lo/JKB tallyeka/VV ass/EP ta/EF ./SF

aph-ulo tallyeka-ass-ta.
forward-DIR run-PST-DECL

'(I) ran forward.'

Table 3.1 presents the word-word co-occurrence matrix for this corpus.

Table 3.1: Word-word co-occurrence matrix

	pang_07/NNG	pakk/NNG	aph/NNG	-(u)lo/JKB	...	./SF
pang_07/NNG	0	0	0	1	...	1
pakk/NNG	0	0	0	1	...	1
aph/NNG	0	0	0	1	...	1
-(u)lo/JKB	1	1	1	0	...	3
...	...	...	...	...	...	...
./SF	1	1	1	3	...	0

Note. Columns and rows are labeled by words.

Each cell records the number of times the row (target) word and the column (context) word co-occur in the above context. In the case of *-(u)lo*, it has a value of one with each word, except *./SF* with a value of three (because both occur in each sentence).

A word-word co-occurrence matrix is generally used in combination with a context window, that is, a range of words surrounding a target word affecting the determination of the characteristics of the word (Lison and Kutuzov, 2017). Consider the same corpus with the context window size as one, counting one word to the left and one word to the right of the target word. This shows the number of times (in the training sentences) that the column word occurs in a one-word window around the row word (Jurafsky and Martin, 2019). This change produces a new word-word co-occurrence matrix as in Table 3.2. In this table, the co-occurrence count of *pang_07/NNG* and *./SF* is zero showing that the size of the context window has a direct influence on the embedding results.

Table 3.2: Word-word co-occurrence matrix with a context window size as one

	pang_07/NNG	pakk/NNG	aph/NNG	-(u)lo/JKB	...	./SF
pang_07/NNG	0	0	0	1	...	0
pakk/NNG	0	0	0	1	...	0
aph/NNG	0	0	0	1	...	0
-(u)lo/JKB	1	1	1	0	...	0
...	...	...	...	...	...	...
./SF	0	0	0	0	...	0

Note. Columns and rows are labeled by words.

Word embedding in consideration of context window size generally calculates co-occurrence with the words located on both sides of the target word (Lison and Kutuzov, 2017). Lison and Kutuzov (2017) presented a systematic analysis of the context window to understand its exact role for word embedding. Employing SGNS as an embedding model, they used two English language corpora: *Gigaword v5* (Parker et al., 2011), with approximately four-billion-word tokens of newswire, and the English version of *OpenSubtitles* (Lison and Tiedemann, 2016), with approximately 700 million-word tokens of movie and TV subtitles. They showed that the performance of word embedding using only the words on the left of the target word was worse than that of using words on both sides. For the *Gigaword* corpus, it also showed that using words on the right performed as well as using both sides, perform-ing only one percentage point less than using both sides. Based on these studies, I used words on both sides of the target word when using context window size in word embedding.

3.2.2 Positive Pointwise Mutual Information

Each cell of a word-word co-occurrence matrix represents the number of times two words occurred at the same time, but the number of occurrences may not serve as a good feature to present the relationship of two words. For example, consider the co-occurrence between the postposition *-(u)lo* and *./SF* in the previous example. They are both used in all the sentences, so the matrix in Table 3.1 shows that *./SF* is highly related to *-(u)lo* due to the high frequency of *./SF* although it is not related to *-(u)lo*.

PPMI (Church and Hanks, 1989) deals with this issue effectively by weighing the association between two words in the search of the co-occurrence of these words in a corpus (Jurafsky and Martin, 2019). To understand how PPMI works, we first need to look at Pointwise Mutual Information (PMI: Fano, 1961). PMI measures how often two words (a target word *w* and a context word *c*) occur compared to what is expected if they are independent of each other, as formalized in (3.1).

$$PMI(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)} \tag{3.1}$$

Suppose a hypothetical frequency table (Table 3.3) obtained from an imaginary corpus consisting of 1,000 sentences with 700 sentences involving *-(u)lo/JKB* and 600 sentences involving *ka/VV*.

Table 3.3: Frequency table from *-(u)lo/JKB* and *ka/VV*

	-(u)lo/JKB	¬ [*-(u)lo/JKB*]	count(*w*)
ka/VV	400	200	600
¬ [*ka/VV*]	300	100	400
count(*w*)	700	300	1000

Note. ¬ stands for 'not'

We can calculate the PMI score between the two words as follows:

$$P(w = \textit{-(u)lo/JKB}, c = \textit{ka/VV}) = \frac{400}{1000} = 0.4$$

$$P(w = \textit{-(u)lo/JKB}) = \frac{400}{700} = 0.571$$

$$P(c = \textit{ka/VV}) = \frac{400}{600} = 0.667$$

$$PMI(\textit{-(u)lo/JKB}, \textit{ka/VV}) = \log_2 \frac{0.4}{(0.571 * 0.667)} = 0.07038933$$

The PMI values range from negative to positive. However, if the corpus size is not large enough, the negative PMI value is less reliable and cannot be used. Furthermore, studies on the meaning of words do not use negative PMI because it does not express the meaning of the target word (Jurafsky and Martin, 2019). For this reason, it is more common to use Positive PMI (PPMI), which replaces all the negative PMI values with zero. But this method has the disadvantage of losing information that can be obtained from negative PMI (e.g., Church and Hanks, 1989, Dagan et al., 1995, Niwa and Nitta, 1994).

3.2.3 Singular Value Decomposition

Using PPMI as a weighting function for a word-word co-occurrence matrix is known to yield genuine co-occurrence relations of two words by suppressing unreasonable relationships between words. However, there still remain issues such as the size of a co-occurrence matrix. For example, suppose that the corpus size continues to increase. The column and row of a word-word co-occurrence matrix will then increase respectively (i.e., dimensions increase in proportion to the number of words). Handling multi-dimension data then requires more computational capacities and resources, rendering this line of research as challenge.

As a remedy for this issue, SVD (Eckart and Young, 1936) was devised by reducing the dimensions of a co-occurrence matrix while maintaining the information of the matrix (e.g., Bullinaria and Levy, 2007, 2012, Hachey et al., 2006, Landauer et al., 1998, Levy and Goldberg, 2014, Schütze, 1992). This is formalized as in (3.2) where A is an m by n rectangular matrix, U is an m by m orthogonal matrix composed of the left singular vector of A, Σ is an m by n diagonal matrix, and V is an n by n orthogonal matrix composed of the right singular vector of A.

$$A = U\Sigma V^T \tag{3.2}$$

The column vectors belonging to the matrix U and V are singular vectors and are orthogonal to each other.

$$U = \begin{bmatrix} u_1 & u_2 & \dots & u_m \end{bmatrix}$$

$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$$

$$U^T U = I$$

$$V^T V = I$$

The singular vectors of the matrix Σ are all greater than or equal to zero. The singular vector σ_k, the *k*th diagonal element of the matrix Σ, is equal to the value taken by the square root at the *k*th eigenvalue of the matrix AA^T.

$$\sigma_k = \sqrt{\lambda_k}$$

This below describes more detail about the SVD formula defined in (3.2).

$$A = U\Sigma V^T = \begin{bmatrix} u_1 & u_2 & \dots & u_m \end{bmatrix} \begin{bmatrix} \sqrt{\lambda_1} & 0 & \dots & 0 \\ 0 & \sqrt{\lambda_2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \sqrt{\lambda_k} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix}$$

To illustrate the calculation process, suppose a 2 by 2 square matrix as in Table 3.4.

Table 3.4: Frequency table (SVD)

	aph/NNG	*-(u)lo/JKB*
-(u)lo/JKB	4	0
ka/VV	3	5

In this table, the co-occurrence frequency of *aph/NNG* and *-(u)lo/JKB* is four, *-(u)lo/JKB* and *-(u)lo/JKB* is zero, *aph/NNG* and *ka/VV* is three, *ka/VV* and *-(u)lo/JKB* is five. The table is then represented by the matrix below:

$$A = \begin{bmatrix} 4 & 0 \\ 3 & 5 \end{bmatrix}$$

First of all, a diagonal matrix, Σ can be calculated as shown below:

$$AA^T = \begin{bmatrix} 4 & 0 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} 4 & 3 \\ 0 & 5 \end{bmatrix} = \begin{bmatrix} 16 & 12 \\ 12 & 34 \end{bmatrix}$$

$$AA^T - \lambda I = \begin{bmatrix} 16-\lambda & 12 \\ 12 & 34-\lambda \end{bmatrix}$$

$$((16-\lambda) * (34-\lambda)) - (12 * 12) = 0$$

$$(\lambda^2 - 50\lambda + 400) = 0$$

$$(\lambda - 40) * (\lambda - 10) = 0$$

The resulting calculated eigenvalues are λ_1,λ_2 = 40, 10 and singular values are σ_1, σ_2 = $\sqrt{40}$, $\sqrt{10}$. Then, the following Σ can be obtained through the given eigenvalues and singular values.

$$\Sigma = \begin{bmatrix} \sqrt{40} & 0 \\ 0 & \sqrt{10} \end{bmatrix}$$

The next step is to calculate *V*.

If $\lambda_1 = 40,$ $\begin{bmatrix} 16-\lambda & 12 \\ 12 & 34-\lambda \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$$\begin{bmatrix} -24 & 12 \\ 12 & -6 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$-12u_1 + 6u_2 = 0$$

$$u_1, u_2 = 1, 2$$

$$X_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

If $\lambda_1 = 10, \begin{bmatrix} 16-\lambda & 12 \\ 12 & 34-\lambda \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$$\begin{bmatrix} 6 & 12 \\ 12 & 24 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$18u_1 + 36u_2 = 0$$

$$u_1, u_2 = -2, 1$$

$$X_2 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

Then, the following V can be obtained through the given X_1, X_2.

$$V = \begin{bmatrix} X_1 & X_2 \end{bmatrix} = \begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix}$$

The final step is to calculate *U* using the previously calculated values.

$$\Sigma = \begin{bmatrix} \sqrt{40} & 0 \\ 3 & \sqrt{10} \end{bmatrix}, V = \begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix}, VV^T = \begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}$$

$$A = U\Sigma V^T$$

$$AV = U\Sigma V^T V = U\Sigma 5$$

$$\frac{1}{5} AV\Sigma^- 1 = U$$

$$U = \frac{1}{5} \begin{bmatrix} 4 & 0 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 0.1581 & 0 \\ 0 & 0.3162 \end{bmatrix}$$

$$U = \begin{bmatrix} 0.1264 & -0.5059 \\ 0.4111 & -0.6957 \end{bmatrix}$$

By applying the values of U, Σ and V, the following results can be obtained:

$$A = \begin{bmatrix} 0.1264 & -0.5059 \\ 0.4111 & -0.6957 \end{bmatrix} \begin{bmatrix} \sqrt{40} & 0 \\ 3 & \sqrt{10} \end{bmatrix} \begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 3 & 5 \end{bmatrix}$$

The above calculation process represents the general formula for SVD. To reduce the dimension, this technique selects the number of dimensions *K*, as shown in Figure 3.1.

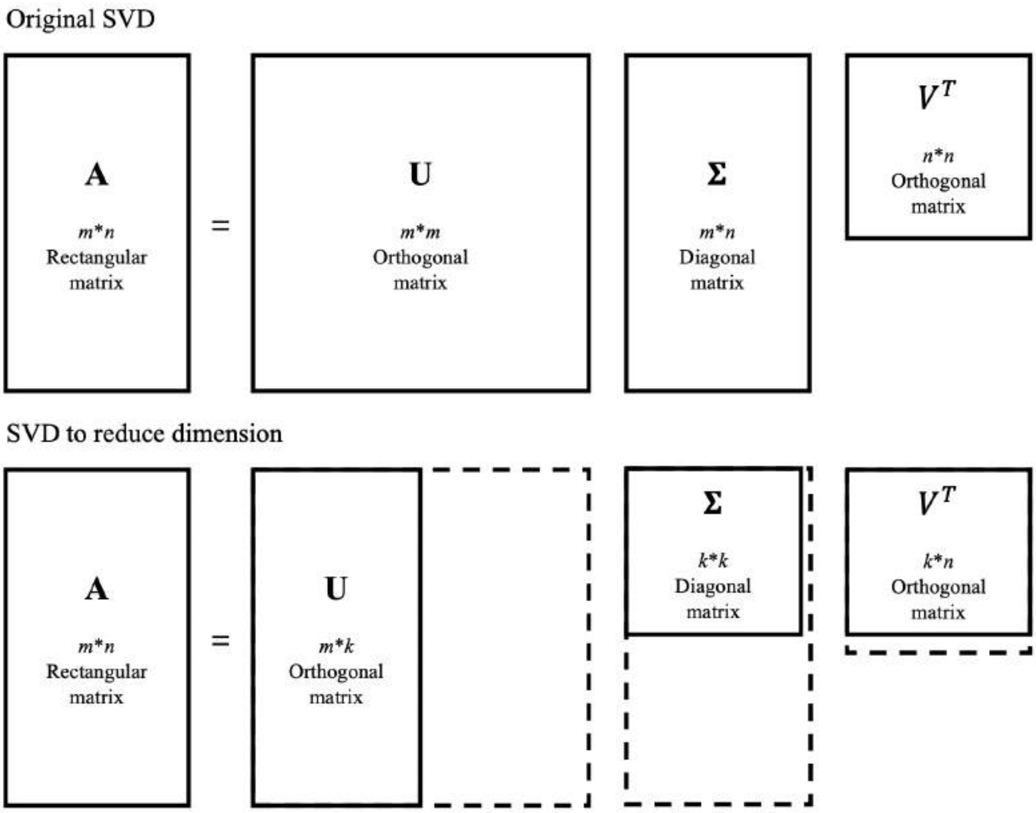

Figure 3.1: Original SVD and SVD to reduce dimension

For example, in the above calculation process, the results of the calculation by setting K as one is as follows:

$$A = \begin{bmatrix} 0.1264 \\ 0.4111 \end{bmatrix} * \sqrt{40} * \begin{bmatrix} 1 & 2 \end{bmatrix} = \begin{bmatrix} 0.7994 & 1.5988 \\ 2.6 & 5.2 \end{bmatrix}$$

In addition, reduction of the number of dimensions in the data from two to one produces the following values:

$$ReducedA = U + (V^T)^T = \begin{bmatrix} 0.1264 \\ \\ 0.4111 \end{bmatrix} + \begin{bmatrix} 1 \\ \\ 2 \end{bmatrix} = \begin{bmatrix} 1.1264 \\ \\ 2.4111 \end{bmatrix}$$

This reduction often leads to losing some information about the data, but the approximation of A is still calculated.

For this reason, SVD is generally used to analyze a large-sized corpus. However, there still remain concerns about the loss of information due to dimension reduction (Hachey et al., 2006). Many studies compared SVD-based results with those from unreduced word-word co-occurrence matrix representations (e.g., Hachey et al., 2006, Matveeva et al., 2005, Pedersen et al., 2005). Hachey et al. (2006) conducted a comparison between the word-word co-occurrence matrix applying SVD with an unreduced version. They used the DUC 2005 data (Dang, 2005) with approximately 100 million words and utilized the *Infomap* tool to build a semantic model based on SVD. Results showed that SVD dimensionality reduction improved performance over a word-word co-occurrence model for computing relevance and redundancy. Overall, these previous studies suggest that if the corpus is large enough, it is more efficient to use a reduced co-occurrence matrix by applying SVD than an unreduced version of the co-occurrence matrix (e.g., Hachey et al., 2006, Matveeva et al., 2005).

In summary, with regards to the count-based model, SVD was mainly used (e.g., Agirre and Lopez de Lacalle, Gliozzo et al., 2005, Hachey et al., 2006), and a combination of PPMI and SVD has also been used recently (e.g., Hilpert, 2016, Turney, 2008, Turney and Pantel, 2010). In this dissertation, PPMI and SVD are applied to word embedding, in order to evaluate which model is more suitable for exploring polysemy issues in Korean.

3.3 Prediction-based model

The prediction-based model is another way to obtain dense vectors such as SVD. This model is based on probability information about the meaning between words and is efficient in conducting tasks such as word similarity (e.g., Mikolov et al., 2013a,b). Similar to the count-based model, converting contexts into vectors is a priority for the prediction-based model. A representative study is Mikolov et al. (2013b) which introduced *Word2Vec*. Generally, the one-hot encoding method is used for this model (Mikolov et al., 2013a,b).

3.3.1 The one-hot encoding

The one-hot encoding is a method that uses 0 and 1 to represent a unique index for each word (Ammar et al., 2016). For example, suppose the following corpus involving the postposition *-(u)lo* as a function of DIR (Direction) as in ((4)-(5)).

(4) pang_07/NNG -(u)lo/JKB ka/VV n-ta/EF ./SF

pang-ulo ka-n-ta.
room-DIR go-PRS-DECL

'(I am) going to the room.'

(5) pakk/NNG -(u)lo/JKB nao/VV ta/EF ./SF

pakk-ulo na-o-ta.
outside-DIR be.out-come-DECL

'(I) go outside.'

This corpus has eight word types (*pang_07/NNG*, *pakk/NNG*, *-(u)lo/JKB*, *ka/VV*, *nao/VV*, *n-ta/EF*, *ta/EF* and *./SF*). The one-hot encoding converts these word types into an eight-dimensional space as in Table 3.5.

Table 3.5: The one-hot encoding table

Words	Encoding
pang_07/NNG	[1,0,0,0,0,0,0,0]
pakk/NNG	[0,1,0,0,0,0,0,0]
-(u)lo/JKB	[0,0,1,0,0,0,0,0]
ka/VV	[0,0,0,1,0,0,0,0]
nao/VV	[0,0,0,0,1,0,0,0]
n-ta/EF	[0,0,0,0,0,1,0,0]
ta/EF	[0,0,0,0,0,0,1,0]
./SF	[0,0,0,0,0,0,0,1]

Each word has its own encoding value as an independent vector. However, one-hot encoding does not present similarity between the words through the given vectors (Ammar et al., 2016). For instance, the words in Table 3.5 are expressed in eight dimensions, and if these words are expressed in a two-dimensional chart, each word is represented by 11.25 degrees (i.e., 90 degrees divided by eight). Because mathematically the position of each word can be obtained by dividing the angle of the two-dimensional space by the number of multi-dimensional spaces. As in Figure 3.2, the distance between each word is same as each other and every word vector is orthogonal with the 90 degrees. For this reason, the similarity such as cosine and Euclidean cannot be calculated with the vector of words obtained by one-hot encoding. Instead, it can be calculated by applying the embedding models such as SVD and Word2Vec, which converts the word into dense vectors.

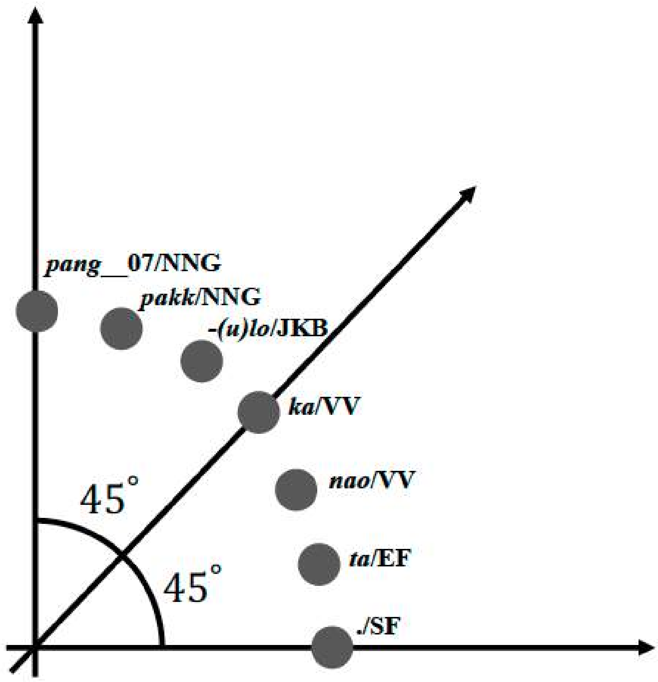

Figure 3.2: Visualization of results through the one-hot encoding

3.3.2 Continuous Bag Of Words

Word2Vec is not a single algorithm but a combination of two techniques: continuous bag of words (CBOW, i.e., predicting the target word from bag-of-words contexts; Mikolov et al., 2013b) and skip-gram and negative sampling (SGNS, i.e., predicting context words given the target word; Mikolov et al., 2013a). Both of these are neural networks using the relation between the target word and co-occurring words and learning weights of word vector representations.

We start from CBOW. For instance, suppose the following sentence as in (6).

(6) pang_07/NNG -(u)lo/JKB ka/VV n-ta/EF ./SF

pang-ulo ka-n-ta.
room-DIR go-PRS-DECL

'(I am) going to the room.'

CBOW uses the surrounding words such as *pang_07/NNG*, *ka/VV*, *n-ta/EF*, *./SF* to predict the target word *-(u)lo/JKB*. The word being predicted is called the target word and the words being used for prediction are called the context word. A context window is used to determine the number of surrounding words to be used to predict the target word. If the window size is m, the number of context words used to predict the target word is *2m* (Mikolov et al., 2013a,b).

For example, if the context window size is 2, information about the target word and the context words for the sentence (6) is represented as in Table 3.6.

Table 3.6: Target word and context words in CBOW

Target word	Encoding	Context words
pang_07/NNG	[1,0,0,0,0]	[0,1,0,0,0],[0,0,1,0,0]
-(u)lo/JKB	[0,1,0,0,0]	[1,0,0,0,0],[0,0,1,0,0],[0,0,0,1,0]
ka/VV	[0,0,1,0,0]	[1,0,0,0,0],[0,1,0,0,0],[0,0,0,1,0],[0,0,0,0,1]
n-ta/EF	[0,0,0,1,0]	[0,1,0,0,0],[0,0,1,0,0],[0,0,0,0,1]
./SF	[0,0,0,0,1]	[0,0,1,0,0],[0,0,0,1,0]

In this table, two words (*-(u)lo/JKB*, *ka/VV*) are used to predict *pang_07/NNG*, and three words (*pang_07/NNG*, *ka/VV*, *n-ta/EF*) are used to predict *-(u)lo/JKB*. Figure 3.3 illustrates the framework of CBOW, which uses three-word information (i.e., one context word immediately left of the target word, the other immediately right) to predict one word.

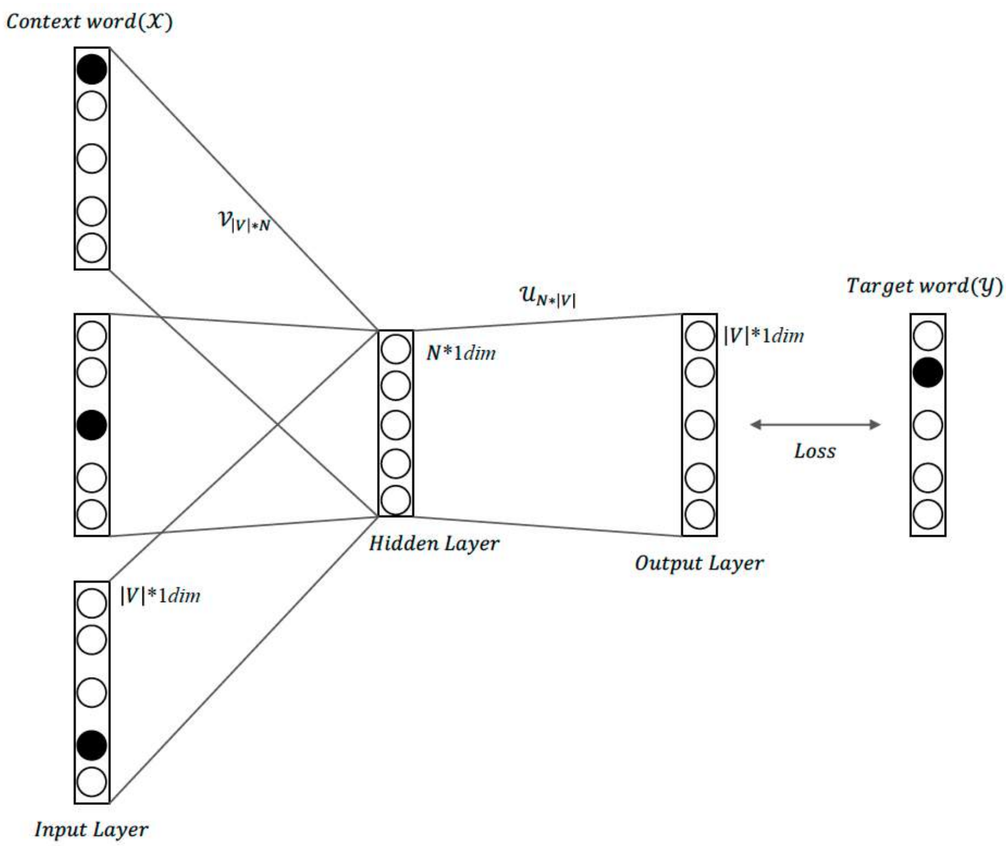

Figure 3.3: A framework of the CBOW model

Each step of the CBOW workflow is as follows: first, one-hot vector for context is inputted to the input layer. For example, if the context window size is *m*, *2m* one-hot vectors are entered into the input, where c is the position of target word and m is context window size used in the process.

$$|V| * 1\text{dim} = X^{(c-m)}, \ldots, X^{(c-1)}, X^{(c+1)}, \ldots, X^{(c+m)}$$

Second, in the process toward the input layer to the hidden layer, the one-hot vector used as the input is multiplied with the input word matrix composed of random numbers. $X_{c\pm m}$ is the one-hot vector of the surrounding words in the range of the context window from the target word and $V_{|V|*N}$ is the input word matrix which is randomly generated.

$$X_{c\pm m} * V_{|V|*N} = v_{c\pm m}$$

Third, the hidden layer calculates the average of the results of the second process.

$$\hat{v} = \frac{v_{c-m}, \ldots, v_{c+m}}{2m}$$

Fourth, in the process from the hidden layer to the output layer, the results of the third process are multiplied with the output word matrix composed of random numbers.

$$z = \hat{v} * U_{N*|V|}$$

Fifth, the probability is calculated in the output layer, using the softmax function to represent the results obtained in the fourth process as probabilities.

$$\hat{y} = softmax(z)$$

The formula for the softmax function is as shown below:

$$softmax = P_i = \frac{e^{zi}}{\sum_{j=1}^{k} e^{zj}} \text{for } i = 1, 2, \ldots, k$$

Suppose the number of classes entered into input is *k*. The softmax estimates probabilities for each class by entering the total classes. This means that the sum of the probability values of total classes is one (Mikolov et al., 2013b).

Finally, an error between the one-hot vector of the target word, y and $\hat{y}$ obtained from the output layer, is measured by cross-entropy function. In this process, the cross-entropy function as shown below is used.

$$cross\text{-}entropy = H(P, Q) = -\Sigma P(x) * \log_Q(x)$$

For example, suppose the correct answer (P) with two categories is $\begin{bmatrix} 1 & 0 \end{bmatrix}$. Q is calculated to approximate P, and if the calculation result is $\begin{bmatrix} 0 & 1 \end{bmatrix}$, the loss becomes an infinite value as shown below:

$$P(x) * \log_Q(x) = -\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \log 0 \\ \log 1 \end{bmatrix} = -(-\infty + 0) = \infty$$

If the calculated Q matches the correct P, the loss is zero as shown below:

$$P(x) * \log_Q(x) = -\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \log 1 \\ \\ \log 0 \end{bmatrix} = -(-0 + 0) = 0$$

If the cross-entropy value between the $\hat{y}$ calculated in the CBOW model and the one-hot vector value of the target word is 0, then the value of $\hat{v}$ in the hidden layer is used as the dimension value of target word. However, if the cross-entropy value is not zero, the CBOW model repeats back propagation to update the $V_{|V|*N}$ and $U_{N*|V|}$.

To illustrate the entire workflow with concrete values, suppose the same corpus involving the postposition *-(u)lo* as a function of DIR (Direction) in (6) is revisited, as in (7).

(7) pang_07/NNG -(u)lo/JKB ka/VV n-ta/EF ./SF

pang-ulo ka-n-ta.
room-DIR go-PRS-DECL

'(I am) going to the room.'

Suppose that we predict *-(u)lo/JKB*, $\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix}$ using the word *pang_07/NNG*, $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}$ from the given sentence. Then the X of the context word entered in the input layer should be $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}$, and the Y in the output layer should be $\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix}$. If the input word matrix is $\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$, through the process of going from the input layer to the hidden layer, then the estimated v_c in the hidden layer is as follows:

$$v_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

After that, if the output word matrix is $\begin{bmatrix} 0 & 3 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 1 \end{bmatrix}$, moving from the hidden layer to the output layer, then the estimated z is obtained as follows:

$$z = \begin{bmatrix} 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 3 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 0 & 0 & 0 \end{bmatrix}$$

The z is then expressed as probabilities using the *softmax* formula and the cross-entropy value between Y and $\hat{Y}$ is calculated to see whether the value is zero or not.

$$\hat{Y} = softmax(z) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$-P(x) * \log_{Q}(x) = -\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \log 0 \\ \log 1 \\ \log 0 \\ \log 0 \\ \log 0 \end{bmatrix} = -(0+0+0+0+0) = 0$$

If the calculated cross-entropy value is 0, then the v_c of the hidden layer is used as a two-dimensional vector for target word, *-(u)lo/JKB*.

The prediction-based models such as CBOW and SGNS, unlike count-based models, can embed words without information on how often the words appear.

3.3.3 Skip-Gram and Negative Sampling

In contrast to CBOW, Skip-gram is an algorithm that predicts context words using the target word. However, similar to CBOW, Skip-gram uses one-hot vector as input and output. The framework (using one word to predict three words) is shown in Figure 3.4.

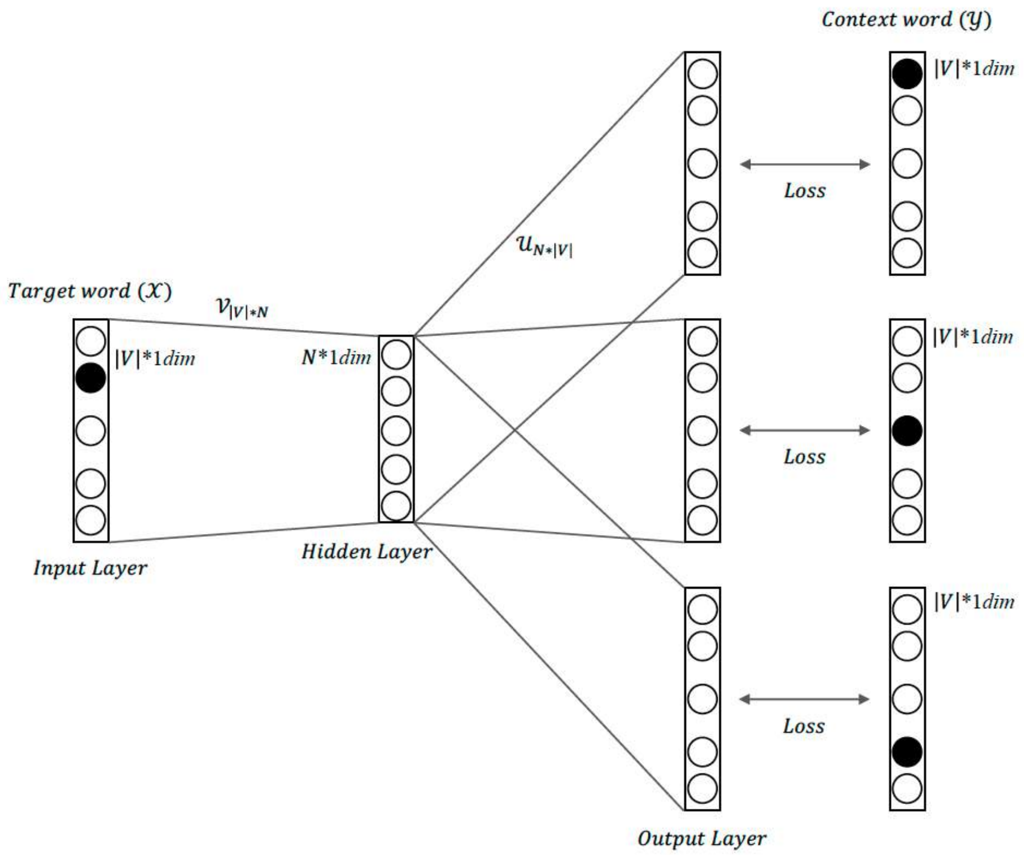

Figure 3.4: A framework of the Skip-gram model

Each step of the Skip-gram workflow is as follows: first, one-hot vector for target word is inputted to the input layer.

$$|V| * 1\text{dim} = X$$

Second, in the process from the input layer to the hidden layer, the input of the target word is multiplied with the input word matrix composed of random numbers.

$$X_c * V_{|V|*N} = v_c$$

Third, moving from the hidden layer to the output layer, the results of the second process are multiplied with the output word matrix composed of random numbers.

$$z = v_c * U_{N*|V|}$$

Forth, the probability is calculated in the output layer using the *softmax* function.

$$\hat{Y} = softmax(z)$$

Finally, an error is measured between Y (the one-hot vector of the context words), and $\hat{Y}$ (obtained from the output layer) by using cross-entropy. In order to further improve the performance of the Skip-gram model, Mikolov et al. (2013a) proposed a negative sampling as in 3.3.

$$\log \sigma({v'_{wo}}^T v_{wI}) + \sum_{i=1}^{k} E_{wi \sim p_n(w)} [\log \sigma(-{v'_{wi}}^T v_{wI})] \tag{3.3}$$

Negative sampling involves calculating probabilities by randomly selecting five to twenty words from the total words as negative samples to reduce the number of calculations required by the softmax function. Then, the samples are combined with the context words to create a set of total words. The number of total words is then used in the softmax calculation process to obtain probability of $\hat{Y}$.

Word2Vec includes two techniques (i.e. CBOW and Skip-gram), which has brought forth questions about which technique is better for word embedding. Some studies compared the performance of the two algorithms for Word2Vec (e.g., Mikolov et al., 2013a, Pennington et al., 2014, Yogatama et al., 2014). Mikolov et al. (2013a) introduced Word2Vec for the first time and compared the performance of the CBOW and Skip-gram algorithms. They found that in semantic tasks, Skip-gram and CBOW reached an accuracy of 0.55 and 0.24, respectively. In contrast, in syntactic tasks, the models yielded similar rates of accuracy (0.59 for Skip-gram and 0.64 for CBOW). This suggests that the performance of the Skip-gram and the CBOW techniques is contingent on task types.

Yogatama et al. (2014) did a similar comparison, together with existing word embedding models such as the Principal Component Analysis and Recurrent Neural Network. They found that in semantic tasks, the CBOW technique showed an accuracy of 0.12 and the Skip-gram technique showed an accuracy of 0.39, and in syntactic tasks, the CBOW technique showed an accuracy of 0.52 and the Skip-gram technique showed an accuracy of 0.54. Overall, the Skip-gram technique showed a higher accuracy rate than the CBOW technique, but once again, the performance was dependent on the task type. Inspired by these studies, a recent trend of research on prediction-based word embedding is to employ a combination of Skip-gram and negative sampling as a prediction-based model. This dissertation also follows this trend by adopting Skip-gram as a specific method of the prediction-based model for word embedding.

3.4 Summary of the Chapter

The DSMs are divided into two types: count-based model (e.g., Singular Value Decomposition (SVD), Eckart and Young, 1936) and prediction-based model (e.g., Skip-Gram and Negative Sampling (SGNS), Mikolov et al., 2013a).

The count-based model embedded each word by counting the number of times they appear. As a fundamental task for this model, a word-word co-occurrence matrix is used to see the relationship between words. However, the number of occurrences may not present the correct relationship of two words. To remedy this, Positive Pointwise Mutual Information (PPMI; Church and Hanks, 1989) is generally used, by weighing the association between two words in search of the co-occurrence of these words in a corpus (Jurafsky and Martin, 2019). However, there still remain the issue that when the corpus size increased, the dimensions of the matrix also increased. Hence, SVD (SVD; Eckart and Young, 1936) is used to reduce the dimensions of a co-occurrence matrix while maintaining the information of the matrix (e.g., Bullinaria and Levy, 2012, Levy and Goldberg, 2014). Furthermore, a method combining PPMI with SVD has recently been frequently used as a count-based model.

The prediction-based model embedded each word based on probability information about the meaning between words. To represent each word independently, one-hot encoding method is used, using 0 and 1 to represent a unique index for each word (Ammar et al., 2016).

As a representative of the model, there is Word2Vec which includes CBOW and SGNS. Since Word2Vec contains two different algorithms, many comparative studies have been conducted on the two (e.g., Mikolov et al., 2013a, Pennington et al., 2014, Yogatama et al., 2014). As a result, many have reported that SGNS performs better than CBOW.

Based on the previous studies that employ PPMI with SVD as a count-based model and SGNS as a prediction-based model, I also implement two DSMs models: a combination of PPMI and SVD (Turney and Pantel, 2010) as a count-based model, and SGNS (Mikolov et al., 2013a) as a prediction-based model, with the manipulation of context window size from one to 10.

Chapter 4

Methodological set-up: PPMI-SVD and SGNS

Improving the accuracy of classifying the functions of postpositions is undoubtedly important, however, revealing the precise environments around postpositions for particular classification is also crucial. In particular revealing them through the window of a cluster of interlinked words because it shows how polysemy resolution is situated in that cluster. In this regard, DSMs draw attention to the computational understanding of human language (see Chapter 3). In order to identify the changes of relationships between postpositions and their co-occurring words, I implement a combination of PPMI and SVD Turney and Pantel (2010) as a representative model of the count-based account, and SGNS Mikolov et al. (2013a) as a representative model of the prediction-based account, with manipulation of context window from one to 10.

This chapter outlines the methodological details of this task, with three specific research questions in mind.

- Research question 1: How does the number of functions a postposition has, affect classification performance for each word-level embedding model?
- Research question 2: What is the role of the context window in the classification performance of each word-level embedding model?
- Research question 3: How does the cluster of postpositions and their co-occurring words change as the environments of word-level embedding change?

4.1 Corpus

4.1.1 Sejong corpus: General description

I use the representative corpus data in Korean known as the Sejong corpus (Kim et al., 2006, combined with the detailed dictionary). The corpus was created by the 21st Century Sejong Project, a ten-year-long project that was launched in 1998. This project aimed to provide large-scale Korean corpora of both written and spoken genres (Shin, 2008). It is composed of six sub-parts: (i) creation of primary/special corpora, (ii) creation of electronic dictionaries of predicates and their case frames that describe semantic relationships between words in a sentence, (iii) distribution of computer-aided information about Korean, (iv) standardization of technical terminologies, (v) support for non-standard characters, and (vi) management of information about Korean (Shin, 2008).

Among the sub-parts described above, this study used the primary corpus to make a list of the functions of the postpositions and the electronic dictionary to obtain the function of each postposition. The primary corpus includes datasets with different types of annotations: a raw corpus (63,899,412 ecel[1]), a grammatically tagged corpus (15,226,186 ecel), a parsed corpus (570,064 ecel), and a semantically tagged corpus (10,132,348 ecel). For the task at hand, I used the semantically tagged corpus in particular. As shown in Figure 4.1, this corpus does not directly provide information about the intended functions of postpositions. Instead, if a noun used in a sentence has multiple meanings, its correct meaning is tagged with an index (e.g., *sacin_07/NNG*). This type of information can reduce the ambiguity that may happen in model learning.

[1]An ecel is defined as a white-space-based unit serving as the minimal unit of sentential components.

BSAA0001-00001596	생산자의	생산자/NNG + 의/JKG
BSAA0001-00001597	얼굴	얼굴/NNG
BSAA0001-00001598	사진이	사진__07/NNG + 이/JKS
BSAA0001-00001599	붙어	붙/VV + 어/EC
BSAA0001-00001600	있는	있/VX + 는/ETM
BSAA0001-00001601	농산물이	농산물/NNG + 이/JKS
BSAA0001-00001602	나오고	나오/VV + 고/EC
BSAA0001-00001603	있다.	있/VX + 다/EF + ./SF

Figure 4.1: Example of the semantically tagged corpus

The electronic dictionary (written in an XML format) describes a frame, which shows the semantic relationships between words in a sentence. It is composed of two types of sub-dictionaries. One is a basic dictionary with 18 grammatical categories, 13 small dictionaries about non-grammatical categories such as idiomatic expression and special words, and 461,163 specifics describing information of individual words such as part of speech, meaning, and brief examples of when it comes to use. The other is an additional dictionary, which is an elaboration of parts of the basic dictionary, with 155,866 more specifics added. The dictionary provides case frames as combinations of postpositions and predicates (Figure 4.2).

```
<sense n="03">
    <sem_grp>
        <sem_class>결과행위</sem_class>
        <trans>be earlier</trans>
    </sem_grp>
    <frame_grp type="FTR">
        <frame>X=N0-이 Y=N1-에 V</frame>
        <subsense>
            <sel_rst arg="X" tht="AGT">인간</sel_rst>
            <sel_rst arg="Y" tht="CRT">시간|(기대)</sel_rst>
            <eg>우리는 예정 시간에 앞질러 도착했다.</eg>
            <eg>그는 내 기대보다 앞질러 그 일을 끝마쳤다.</eg>
        </subsense>
    </frame_grp>
</sense>
```

Figure 4.2: Example of a case frame in the Sejong electronic dictionary

The Sejong electronic dictionary consists of 31,093 frames involving 15,181 verbs and 8,115 frames involving 4,398 adjectives (e.g., Seong, 2007). Of these frames, 2,384 are frames with *-ey* (2,115 verbs; 269 adjectives), 766 are frame with *-eyse* (752 verbs; 14 adjectives), and 1,991 are frames with *-(u)lo* (1,782 verbs; 109 adjectives).

4.1.2 Composition of a corpus with respect to the three adverbial postpositions

For exploratory purposes, I analyzed the corpus in order to see how many sentences contained the three target postpositions *-ey*, *-eyse*, and *-(u)lo*. Through Java environment, I confirmed the sentences one by one and sort out the sentences containing these postpositions. The results showed a total of 698,002 sentences with the postpositions (349,118 instances of *-ey*, 121,532 instances of *-eyse*, and 227,352 instances of *-(u)lo*). These postpositions (i.e., *-ey*, *-eyse*, and *-(u)lo*) were ranked as the most frequent ones used out of adverbial postpositions in the corpus. Regarding the functions of these postpositions (see Section 2.1), the number of functions diverges according to the postposition types, as shown in Table 4.1.

Table 4.1: By-function frequency list of *-ey*, *-eyse*, and *-(u)lo*

-ey		*-eyse*		*-(u)lo*	
Function	Frequency	Function	Frequency	Function	Frequency
LOC	1,328	SRC	487	FNS	857
GOL	665	LOC	197	INS	561
EFF	150			DIR	324
CRT	124			EFF	38
THM	58			CRT	22
INS	17			LOC	9
AGT	13				
FNS	11				

-ey has 8 functions (see Section 2.1.1), with LOC and having most occurrences. *-eyse* has only two functions, SRC and LOC (see Section 2.1.2). *-(u)lo* has six functions (see Section 2.1.3), with the top three functions (FNS, INS, and DIR) having more than 80 per cent of the occurrences.

4.1.3 Creation of a hand-coded corpus

To see the relationship between postpositions and their surrounding words, I needed a corpus with the intended functions of postpositions tagged in each sentence. However, the current corpus data does not code the functions of postpositions directly. Therefore, I annotated the corpus manually with the help of three native speakers of Korean. Among the three, one was an instructor who teaches Korean to children and the other two were Ph.D. candidates in linguistics. They managed all the details of the corpus annotation, from the development of the annotation manual to the manual annotation of the intended function of postposition in each sentence.

Regarding the process of creating a hand-coded corpus, I extracted sentences having only one postposition and predicate. Although this manipulation omits many sentences already extracted from the original corpus, it is beneficial for controlling any additional confounding factors which could have interfered with the performance of my model. If a sentence contains more than one postposition, including the three postpositions that I focus on, they become less independent of each other. This means the model performance of each postposition will be affected by each other. This reduction process results in a total of 27,720 sentences, with 14,096 sentences for *-ey*, 5,078 sentences for *-eyse*, and 8,546 sentences for *-(u)lo*. I then extracted 5,000 sentences randomly for each postposition to keep an equal number of sentences for each one.

The final corpus data were then hand-coded by the three native speakers of Korean, following the functions of the individual postpositions. The interrater reliability of the data was measured with the Fleiss's Kappa (Landis and

Koch, 1977). The results were a score of 0.948 for *-ey*, 0.928 for *-eyse*, and 0.947 for *-(u)lo*, which are considered 'almost perfect' according to the Kappa scale. I decided to exclude sentences that caused disagreement among the human annotators (i.e., 285 sentences for *-ey*, 147 sentences for *-eyse*, and 292 sentences for *-(u)lo*). After which, I obtained the final corpus data for each postposition. This yield 4,715 sentences for *-ey*, 4,853 sentences for *-eyse*, and 4,708 sentences for *-(u)lo*. Table 4.2 presents the detailed by-function frequency list of the three postposition types[2].

Table 4.2: By-function frequency list of *-ey*, *-eyse*, and *-(u)lo* in cross-validated corpus

-ey		*-eyse*		*-(u)lo*	
Function	Frequency	Function	Frequency	Function	Frequency
LOC	1,780	LOC	4,206	FNS	1,681
CRT	1,516	SRC	647	DIR	1,449
THM	448			INS	739
GOL	441			CRT	593
FNS	216			LOC	158
EFF	198			EFF	88
INS	69				
AGT	47				
Total	4,715	Total	4,853	Total	4,708

[2] The hand-coded corpus is available at: https://github.com/seongmin-mun/Corpora/tree/master/APIK

In this hand-coded corpus, the order of frequency of the functions for each postposition differed from the Sejong dictionary. For example, LOC, GOL, and EFF were the most frequent functions of *-ey* in the Sejong dictionary, but LOC, CRT, and THM were used the most in the hand-coded corpus. For *-eyse*, the LOC occupied a larger proportion than the SRC. For *-(u)lo*, the functions occupied in the same order of FNS, INS, and DIR as in the Sejong dictionary. And although the same functions were found to be most frequent in the hand-coded corpus, they occurred in a different order: FNS, DIR, and INS. These results do not pose a problem in conducting this dissertation, but this means that the functions used most frequently in the dictionary are different from the ones in the actual corpus.

4.1.4 Training and test sets

Every instance of the hand-coded corpus was lemmatized and POS-tagged before the actual data processing stage. Using the corpus for this task requires the functions of each postposition to be marked overtly with the form of each postposition (e.g., *에/JKB_CRT*). Therefore, I tagged the functions of the postpositions manually. Figure 4.3 illustrates the format of instances used for model training and testing.

12,입/NNG 에/JKB_CRT 맞_01/VV 아/EF
13,밤_01/NNG 에/JKB_CRT 만/JX 들어오/VV 아요/EF
14,내일/NNG 몇/MM 시/NNB 에/JKB_CRT 오/VV 시/EP ㅂ니까/EF
15,다음_01/NNG 에/JKB_CRT ㄴ/JX 아무것/NNG 도/JX 없/VA 어/EF
16,낮/NNG 에/JKB_CRT 자_01/VV 요/EF
17,순아/NNP 는/JX 불시/NNG 에/JKB_CRT 들이닥치/VV ㄴ다/EF
18,증가_01/NNG 에/JKB_CRT 그치/VV 었/EP 다/EF
19,아침/NNG 10/SN 시/NNB 에/JKB_CRT 나서/VV 었/EP 다/EF

Figure 4.3: Example sentences used in model training (-*ey*, CRT)

The data for training and testing should be independent. Thus, I made the model divide the corpus into two sub-sets, one with 90 percent of the corpus for the training and the remaining 10 percent for the testing. In order to obtain a normalized result from each model, I employ the *k*-fold cross-validation technique (Salton, 1971), which evaluates the model by partitioning the original corpus into *k* equal size subsamples. Of the *k* subsamples, a single subsample is retained as the test set, and the remaining *k*-1 subsamples are used as training sets. Another commonly used way of sampling is random sampling, which is to extract the training and test sets randomly with several iterations to obtain a normalized estimate. However, fully random sampling has the risk that the sentences used as training sets will remain as training sets, and likewise, test sets will remain as test sets only. The *k*-fold cross-validation technique has the advantage that no overlap occurs between the training and test sets, while all instances are still used for both training and testing. I set the value of *k* as 10 and repeat the cross-validation 10 times, with each of the 10 subsamples use exactly once as the test set (Figure 4.4).

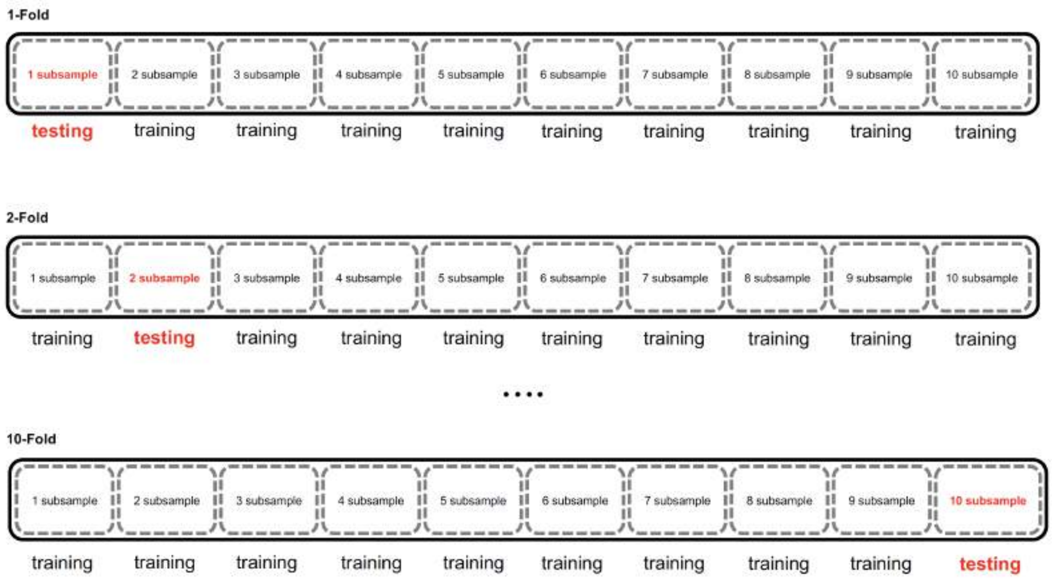

Figure 4.4: The process of the *k*-fold cross-validation technique

4.2 Model training

Model training consists of two parts: (i) word-level embeddings to check the relationship between words, and (ii) similarity-based estimation (Dagan et al., 1995) to determine the intended functions of the postpositions used in the test set.

4.2.1 Word-level embedding: PPMI-SVD and SGNS

I use a combination of PPMI and SVD (Turney and Pantel, 2010) as a count-based model and SGNS (Mikolov et al., 2013a) as a prediction-based model as word-level embeddings to see how clusters between postpositions and their co-occurring words change. In addition, I manipulate the context window size from one to ten. For this, an algorithm for word-level embedding was developed (see Appendix A.1 for a comprehensive view of the workflow). The general flow is as follows: first, the model creates a list of words that exist in the training sets obtained from the 10-fold cross-validation technique. Second, based on the word list, a word-word co-occurrence matrix (for the count-based model) and one-hot vectors (for the prediction-based model) are generated. Third, the model produces word-level embeddings using PPMI-SVD and SGNS.

The algorithm was developed in a Python environment. *Linalg* from the *scipy* package was used for the PPMI-SVD model training. *Word2Vec*, from the *gensim* package, was used for the SGNS model training. The word-level embeddings generated by each model had 500 dimensions, each of which was stored in a database. A total of 600 embeddings were made through this process (2 models * 3 postpositions * 10 folds * 10 window sizes).

4.2.2 Similarity-based estimation

Based on the word-level embeddings generated by the first algorithm, the second algorithm was developed to classify the intended function of postpositions used in the test set. This was done by calculating similarity-based estimation (Dagan et al., 1995); classifying the meaning of the target word that was never used in the training sets by using calculated similarity scores between words. This is a classic method considering the recent development of word embedding research (e.g., Auger and Barrière, 2008, Hazem and Morin, 2013, Kazama et al., 2010, Zhitomirsky-Geffet and Dagan, 2009), but it enhances classification performance through similarity scores indicating that the relationship between words are used to determine the meaning of the target word (Zhitomirsky-Geffet and Dagan, 2009). It can also be used to estimate that the target word even has more than one meaning.

A similarity-based estimation is proposed by Dagan et al. (1995) for the first time. They discussed how to estimate the meaning of a target word that does not occur in the training data. They proposed a method by obtaining information about the words around the target word in order to estimate the intended meaning of target word. Figure 4.5 shows how they did so.

(w_1, w_2)	$\hat{I}(w_1, w_2)$	$f(w_1, w_2)$	$f(w_1)$	$f(w_2)$
(introduction, describes)	6.85	5	464	277
(book, describes)	6.27	13	1800	277
(section, describes)	6.12	6	923	277
Average:	6.41			

Figure 4.5: The similarity-based estimation as an average on similar pairs (Dagan et al., 1995, p. 167)

Suppose that we have a relation from the training set involving the words (*chapter*, *introduction*, *book* and *section*), one from the test set that contains three of the same words (*introduction*, *book* and *section*), and one word (*describes*) that does not occur in the training set. In this situation, the task is to calculate the similarity scores between *describes* and *chapter*. The method proposed by Dagan et al. (1995) calculates the average score between the target word (*describes*) and the three words (*introduction*, *book* and *section*) that appear in test set and the training set. The average of similarity score (6.41) is used as the estimated similarity score of the two words (*chapter* and *describes*).

The postpositions in my training sets are tagged with their intended functions in a sentence (e.g., *에/JKB_CRT*), but the ones in the test set are not (e.g., *에/JKB*). I aim to determine the intended function of the postpositions in the test set based on the cluster obtained from word-level embeddings from the training set. From this point of view, the similarity-based estimation is consistent with the aims of this dissertation, so I use it as the main algorithm of the classification model.

4.2.3 Classification model adapted from similarity-based estimation

To apply a similarity-based estimation to my algorithm, I made the training and test sets differently. This is because the postpositions used in the test set should be recognized as ones whose function is unknown so that they can be classified into designated functions by the model trained. This is illustrated in Figure 4.6. The postpositions in the training set are tagged with the intended functions while the one in the test set is not.

Training set

집__01/NNG 에서/JKB_LOC 동화책/NNG 을/JKO 보/VV 았/EP 다/EF ./SF
집__01/NNG 에서/JKB_SRC 다시/MAG 오/VV 았/EP 다/EF ./SF

Test set
그/MM 와/JKB 집__01/NNG 에서/JKB 만나/VV 았/EP 다/EF ./SF

Figure 4.6: The training sets and test set used in this dissertation (-*eyse*)

The classification algorithm (Appendix A.2) works as follows[3]: first, the algorithm loads a total of 600 word-level embeddings (2 models * 3 postpo-

[3]The entire code for the word-level embedding models that I developed are available at: https://github.com/seongmin-mun/PhD_dissertation/tree/main/Python/PPMI-SVD and https://github.com/seongmin-mun/PhD_dissertation/tree/main/Python/SGNS

sitions * 10 folds * 10 window sizes) generated by the first algorithm and calculates the similarity between the postpositions and the surrounding words. Second, the algorithm loads a test set and makes the list of words in it. Third, the algorithm compares the list of words used in the test set to the one used in the training set and generates a list of words that are shared with each other. Fourth, the algorithm calculates the average score between each function of the postpositions and a list of words that are shared with each other. Finally, the algorithm determines the function of the postpositions used in test set with the highest average.

For an illustration, suppose a relationship based on cosine similarity scores from the training set as in Figure 4.7.

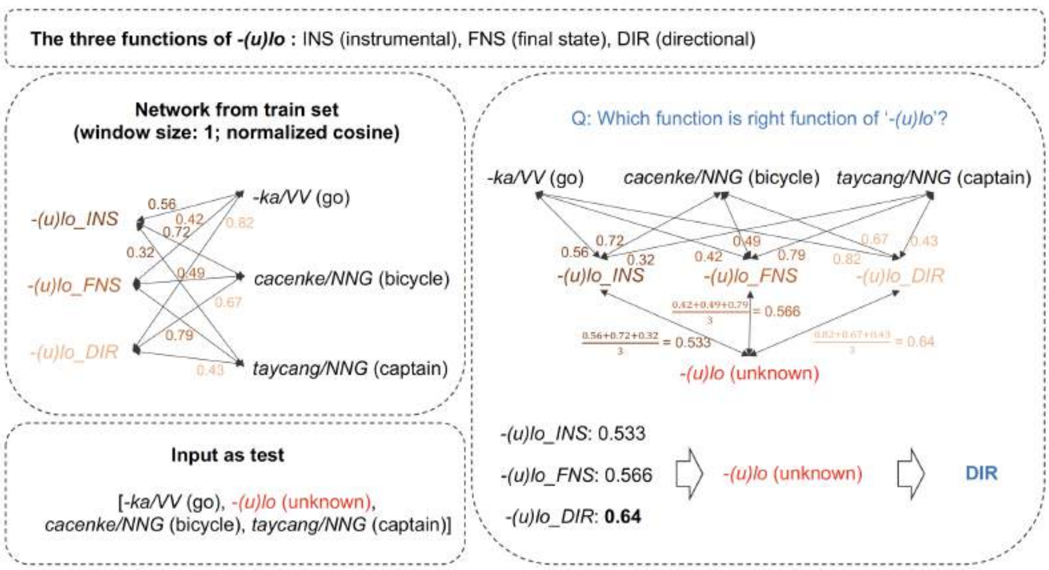

Figure 4.7: The classification model process adapted from Dagan et al. (1993): a case of *-(u)lo*

The similarity-based relationship contains the same form of *-(u)lo* with three different functions and shares three words (*-ka/VV*, *-cacenke/NNG*, *-taycang/NNG*), but the functions have different similarity scores with the three words. The issue here is how to determine the function of this postposition when the test set involves the same postposition with an unspecified function and the same three words used in the training set. To recognize the intended function of *-(u)lo*, the classification model calculates average scores from each of the three different *-(u)lo* (*-(u)lo_INS*: 0.533, *-(u)lo_FNS*: 0.566, *-(u)lo_DIR*: 0.64). Based on the score, the model classifies the intended function of *-(u)lo* in the test set as directional. Through similarity-based estimations, the relationship between word-level embeddings can be used with DSMs to determine the intended function of postposition in the test set that does not occur in the training set.

4.3 Visualization: PostEmbedding

The relationship between words embedded in multiple dimensions through DSMs is difficult to identify at first glance because it is composed of a complex matrix. However, reducing the multi-dimensional embedding matrix into a two-dimension one and visualizing it makes the identification of the relation more recognizable (Mun and Lee, 2016). For example, Hilpert (2016) performed a diachronic corpus-based study of the English modal auxiliary *may*, focusing on changes in its collocational preferences over the past 200 years. He visualized the relationship of the embedded words by reducing the dimensions to two and the changes in the relation between words over time through density maps. Based on the visualization results, he was able to easily and accurately identify the relation between words that varied over time. Desagulier (2014) selected four English adverbs, *rather*, *quite*, *pretty*, and *fairly*, and conducted a study to identify the conceptual contents they presumably share. In his work, he used two-dimensional visualizations and interpreted the relationship of words located around the four adverbs at a glance and accurately.

By using visualization techniques, at least three types of information are identified easily: (i) the degree of similarity across words through their locational distance, (ii) changes of word relations according to change of environments, and (iii) designated word properties by way of colors and sizes. For these reasons, I also use visualization to express and interpret the results in this dissertation.

4.3.1 t-SNE and the cosine similarity

The visualization system aims to see the relationship between the postpositions and their surrounding words in the hand-coded corpora. Rather than employing the 600 word-level embeddings above (2 models * 3 postpositions * 10 folds * 10 window sizes), 60 word-level embeddings per postposition were used (2 models * 3 postpositions * 10 window sizes). In order to express the word-level embeddings of DSMs involving the multi-dimensional matrix into the two-dimensional visualization, dimension reduction techniques should be employed.

Various techniques have been suggested such as the t-distributed Stochastic Neighbor Embedding (t-SNE; Maaten and Hinton, 2008), Principal Components Analysis (PCA; Hotelling, 1933) and Classical Multidimensional Scaling (MDS; Torgerson, 1952). These techniques that reduce high-level dimensions to low-level dimensions differ from each other according to what kind of data points they focused on during the reduction process (Maaten and Hinton, 2008). For instance, conventional dimensional reduction techniques such as PCA and MDS are linear techniques that concentrate on maintaining low-dimensional representations of similar data points, thus performed well to express similar data points (e.g., Hotelling, 1933, Torgerson, 1952). However, they have a disadvantage wherein they lose the information of dissimilar data points because they do not focus on maintaining low-dimensional representations of dissimilar data points. In contrast, the t-SNE technique considers whether variables are (dis-)similar simultaneously, and thus having the advantage over the other techniques due to its higher accuracy (Maaten and Hinton, 2008).

For example, Figure 4.8 shows whether t-SNE takes into account both the similar and dissimilar data points in the process of reducing a two-dimensional plot to a one-dimensional plot. (a) represents the process where a two-dimensional plot is transformed into a one-dimensional plot relative to the x-axis. This way produces one group of three words (*pang_07/NNG*, *-(u)lo/JKB*, *ka/VV*) and the another of two words (*ta/EF*, *./SF*). (b) shows the process where the same two-dimensional plot is transformed into a one-dimensional plot relative to the y-axis. This treatment generates one group of three words (*ka/VV*, *ta/EF*, *./SF*) and another group of two words (*pang_07/NNG*, *-(u)lo/JKB*). However, these results do not seem to be correct intuitively because the two-dimensional plot produces three groups of words by similarity: one group of two words (*pang_07/NNG*, *-(u)lo/JKB*), another of two words (*ta/EF*, *./SF*), and one of one word (*ka/VV*). With this in mind, (c), using t-SNE, represents proper relationships between the words in the plot. Here, the words that are close to each other in a two-dimensional plot are also close in the one-dimensional plot, and likewise the words that are far remain far. This is because t-SNE used both dissimilar and similar data points simultaneously (Maaten and Hinton, 2008). Considering the accuracy of this method in representing word relations, I employ t-SNE for the task of dimension reduction.

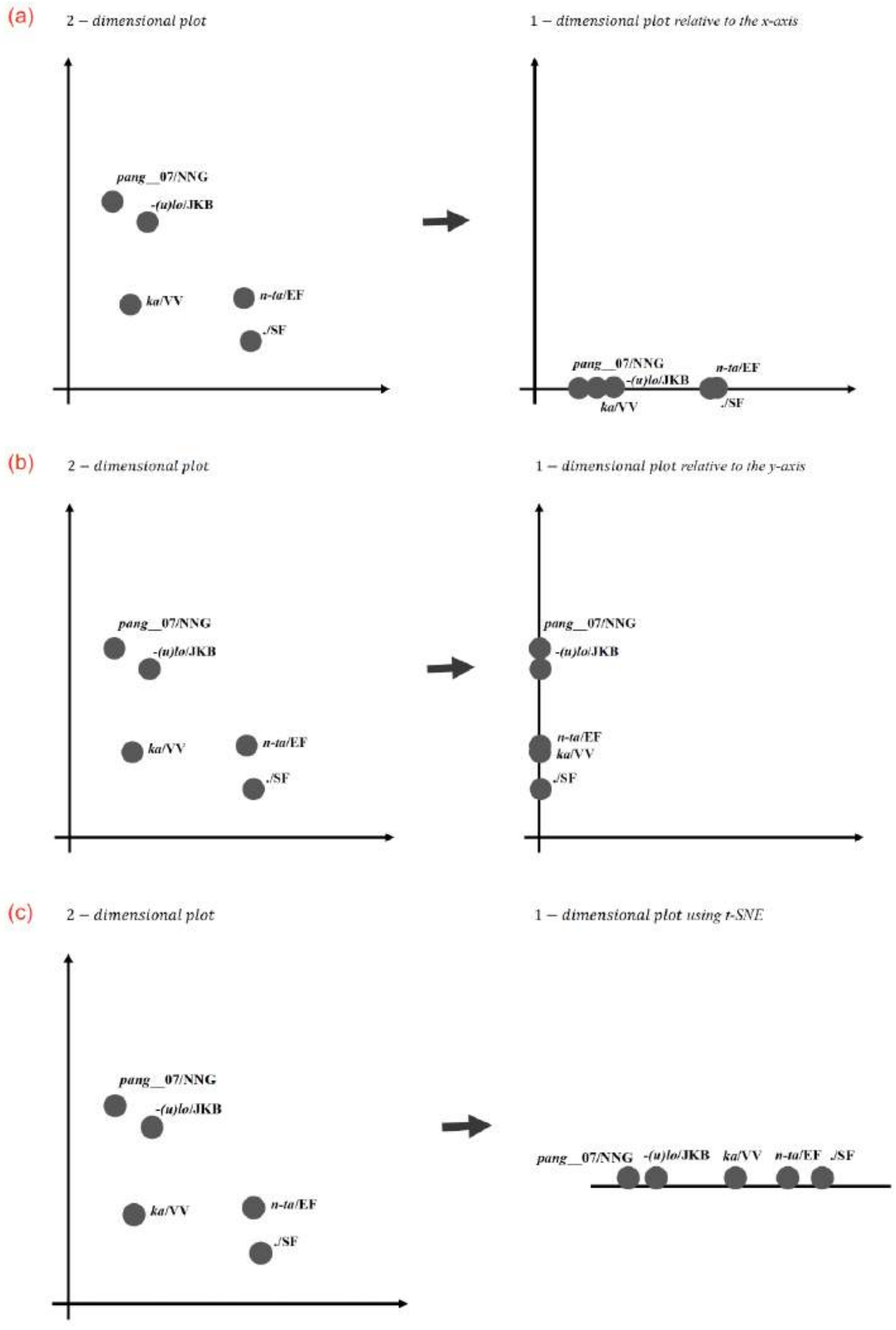

Figure 4.8: Reducing a two-dimensional plot to a one-dimensional plot using the t-SNE

After reducing the 500-dimensional word-level embeddings to two-dimension using t-SNE, I visualize the two-dimensional values as (X, Y) co-ordinate values. By visualizing the reduced results in the two dimensions the relation between a postposition and its co-occurring words can easily be seen. Additionally, the cosine similarity formula as formalized in (4.1) calculates the similarity score between a postposition and its co-occurring words to see how similar they are.

$$Similarity = cos(\theta) = \frac{A * B}{\|A\|\|B\|} = \frac{\sum_{i-1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} * \sqrt{\sum_{i=1}^{n}(B_i)^2}} \tag{4.1}$$

By using the cosine similarity formula, which words are related to each function of the postpositions can be seen. In this dissertation, I design and develop a visualization system to better interpret the changes of the relationship between a postposition and its co-occurring words intuitively.

4.3.2 Tasks and design objectives

Visualization can support evaluating result by exploring data and drawing meaningful interpretations from the data efficiently (Mun et al., 2017). Hence, I design my visualization system by specifying tasks and objectives as follows:

Task 1: Visually represent different clusters using the embedding models, the context window sizes, and the postposition types.

Design Objective: Design options for users to select the embedding models, the context window sizes, and the postposition types.

Task 2: Identify the real corpus data used for training and the details of each word in the cluster (e.g., part-of-speech, frequency of occurrence, word meaning).

Design Objective: Add separate pop-up views to represent the aforementioned information about the cluster when the user moves the cursor over the circle (i.e., each word).

Task 3: Identify the relation between the functions of postpositions and the nearest words.

Design Objective: Add the similarity scores calculated by the cosine similarity formula in the system so that users can more accurately identify the similarity between the postposition and its co-occurring words.

4.3.3 System development

Considering the tasks and design objectives, I developed a visualization system (available at: PostEmbedding) that helps to interpret the clusters between the postpositions and their surrounding words intuitively[4]. The system was developed through Java, JavaScript, HTML, and CSS environments. The development process of the visualization consists of three parts: (i) data pro-

[4]More details of PostEmbedding is available at: https://github.com/seongmin-mun/VisualSystem/tree/master/Major/PostEmbedding

cessing, (ii) front-end, and (iii) back-end.

For the data processing, I transformed the obtained t-SNE outcomes in CSV format which is the delimited text file using the comma to separate values, into JSON format (i.e, the standard text-based format for representing structured data based on JavaScript object syntax). In this part, I generated three types of data through Java programming while adapting *JSONObject* and *JSONArray*. The first data contains t-SNE outcomes that I obtained from the similarity-based estimation algorithm. This data is connected with the distributional semantic map of the visualization system to show the clusters between word embeddings (see Figure 4.9 (b)). The second data includes raw sentences involving each function of postpositions. This data is connected with the concordance table view (see Figure 4.9 (c)). The third data contains the similarity information between each postposition and co-occurring words. This data is used in the Force-directed graph view and the Nearest words view (see Figure 4.9 (d)). After the data processing, these JSON data were stored in the database which is connected with the visualization system.

In the front-end part of the visualization, I used *Bootstrap* in order to design interface components of the visualization system. Moreover, I used *Media queries* from CSS. This makes the visualization system modify the size of the interface automatically depending on a device's general type that is currently used by the user.

Finally, in the back-end part of the visualization, I used *D3.js* to create interactive visualization in the browser. By using *D3.js*, I manipulated the elements of a webpage such as SVG, or Canvas elements according to the contents of the data set. Moreover, I used *jQuery* in order to make several functions through JavaScript more easily.

4.3.4 Interface of visualization system

For the interface of the visualization system, I propose three views to effectively explore the relationships between each postposition and co-occurring words.

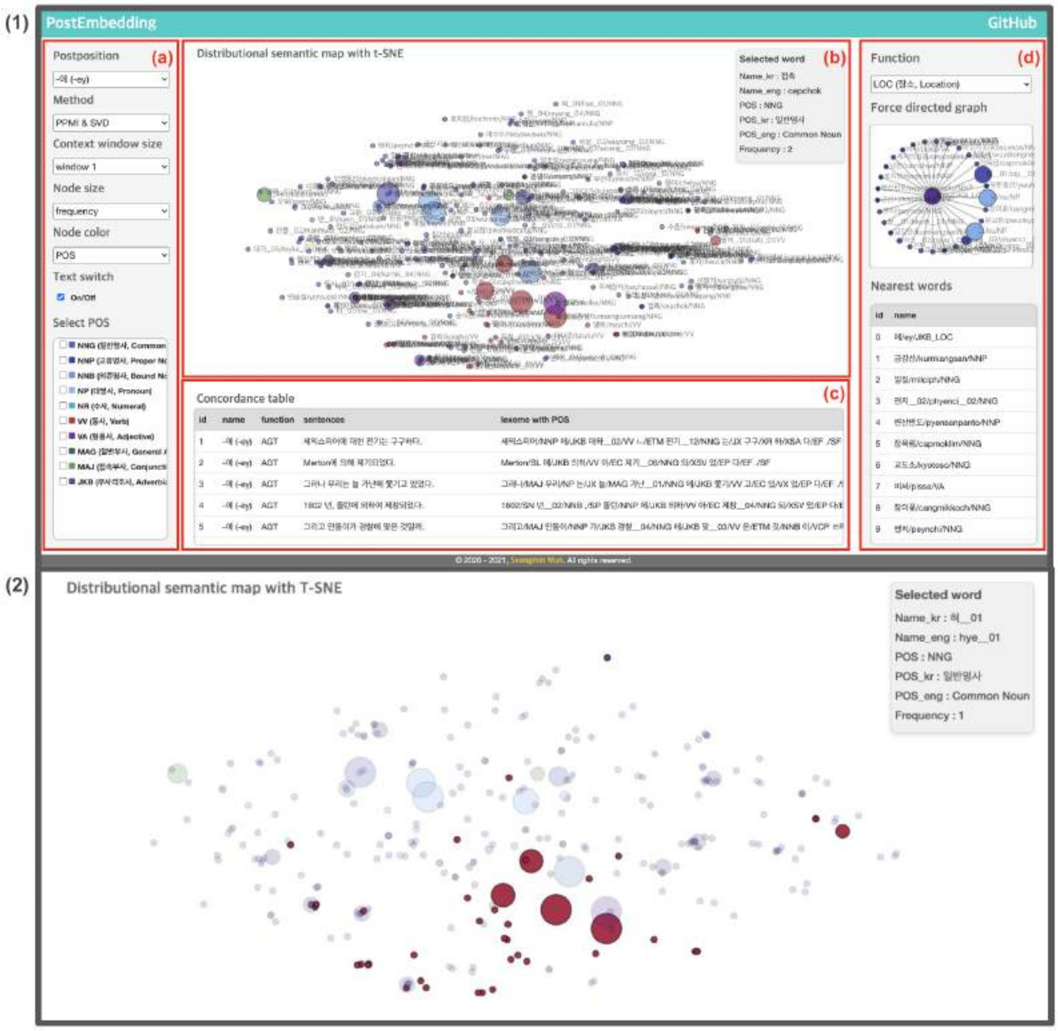

Figure 4.9: Interface of visualization system (1) and the main view of the system (2)

In Figure 4.9, (1) shows the overall composition of the developed visualization system. (a) provides menus to select the postpositions, the models, and the window sizes to check word-level embeddings results. It also allows users to adjust the color and size of the circle representing each word, to turn on and off the text above the circle, and to highlight the circle according to the selected parts of speeches. (b) shows a distributional semantic map of the word-level embeddings reduced to two dimensions using t-SNE. (c) shows the hand-coded corpus actually used in the selected postposition for each function. (d) allows users to choose particular functions of the postpositions and check the information about surrounding words relative to the function. The developed system shows changes of the relationship between one word and the co-occurring words using the changes of clusters that are generated by combinations of these words.

4.4 Summary of the Chapter

I made a hand-coded corpus based on the Sejong corpus (Kim et al., 2006, combined with the detailed dictionary). Because it does not indicate the functions of postpositions directly onto the postpositions themselves. After annotating the corpus manually, I obtained the total 4,715 sentences for *-ey*, 4,853 sentences for *-eyse*, and 4,708 sentences for *-(u)lo*. The hand-coded corpus of each postposition were used in the model training process.

Model training was divided into two steps. The first step was the word-level embeddings to check the relationship of words. For this, I used a combination of PPMI and SVD (Turney and Pantel, 2010) as a count-based model and SGNS (Mikolov et al., 2013a) as a prediction-based model, with manipulation of context window from one to ten. The 10-fold cross-validation technique (Salton, 1971) was used to evaluate the model by dividing the original corpus into 10 equal size subsamples. Through this step, I obtained 600 embeddings (2 models * 3 postpositions * 10 folds * 10 window sizes) to see the clusters between postpositions and their co-occurring words.

The second step was a similarity-based estimation (Dagan et al., 1995) to make a classification model based on word-level embeddings. In order to adapt this concept, the training set and the test set were revised differently. The training set was tagged with the intended function of the postpositions (e.g., *에/JKB_CRT*) and the testing set was not (e.g., *에/JKB*). I designed an algorithm for the classification model adapted from the similarity-based estimation which used the relationship between postposition and co-occurring words.

After training the model, I made a visualization system to interpret the relationship for each word-level embedding easily. The resulting system has several options to use and can identify each word-level embedding reduced as a two-dimensional plot using t-SNE. It also shows the user more details of each word in the word-level embeddings.

In conclusion, I made the word-level embeddings by employing PPMI-SVD and SGNS. Then, based on these embeddings, I developed a classification model by using the concept of similarity-based estimation. I then developed a visualization system to see the word-level embeddings interactively to check the changes of the clusters between each function of postpositions and the co-occurring words.

Chapter 5

Results: word-level embeddings

This chapter provides results of the classification models that I developed, starting from my hypothesis on the research questions (see Chapter 4) to by-model and by-postposition accuracy levels of each model.

- Research question 1: How does the number of functions a postposition has affect classification performance for each word-level embedding model?
- Research question 2: What is the role of the context window in the classification performance of each word-level embedding model?
- Research question 3: How does the cluster of postpositions and their co-occurring words change as the environments of word-level embedding change?

5.1 Hypotheses

Hypotheses were made according to the three research questions regarding the accuracy levels of my classification models and the changes of clusters involving the three Korean adverbial postpositions (*-ey*, *-eyse*, and *-(u)lo*) and their surrounding words.

- Hypothesis 1: The accuracy of the classification should be inversely proportionate to the number of functions of a postposition.

Co-existence of multiple (and related) functions of one form (i.e., polysemy) involving a postposition renders the recognition and use of the postposition ambiguous (e.g., Choo and Kwak, 2008). Given this fact, I predicted that the more functions a postposition has, the lower the accuracy the classification models would demonstrate.

- Hypothesis 2: The accuracy of the classification should be higher in smaller window sizes.

Previous studies have shown the benefits of smaller sizes of context window in addressing word-level polysemy (e.g., Bullinaria and Levy, 2012, Levy and Goldberg, 2014). I thus predicted that the classification accuracy of my models should increase as the context window sizes decrease.

- Hypothesis 3 (on hyperparameters): The clusters and their co-occurring words should vary depending on the environments of word-level embedding (2 models * 3 postpositions * 10 window sizes).

Previous studies have shown different embedding results depending on the models, window sizes, or corpus used in their study (e.g., Bullinaria and Levy, 2007, 2012, Hilpert, 2016, Levy and Goldberg, 2014, Turney and Pantel, 2010). I thus predicted that different clusters and their co-occurring words should be created according to the different environments used by manipulating model types, postposition types, and window sizes.

5.2 Model performance: Classification

5.2.1 Overall accuracy by model: PPMI-SVD and SGNS

PPMI-SVD (count-based)

Figure 5.1 presents the classification accuracy of the PPMI-SVD model adjusting the context window sizes of each postposition.

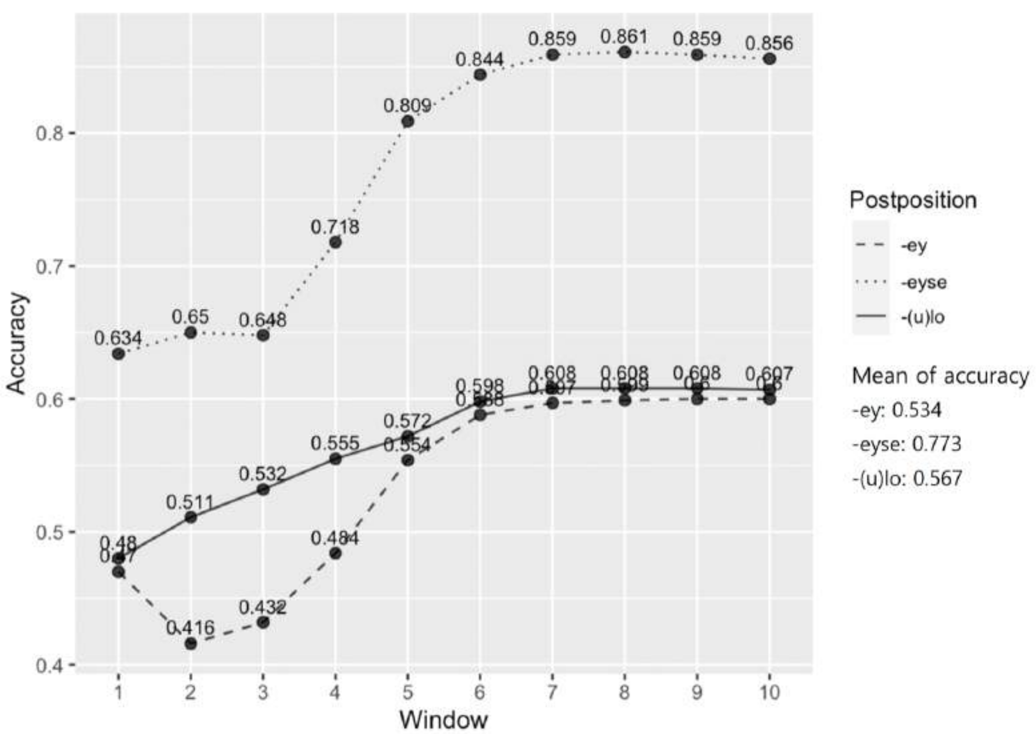

Figure 5.1: Classification accuracy by window size for the PPMI-SVD model

It was found that the model performed better for *-eyse* than the other two postpositions (*-ey* and *-(u)lo*). The reason being that *-eyse* has only two functions, SRC and LOC, with the latter occupying more than 85 percent of the entire corpus. This means that even if all the sentences were classified as LOC, the model accuracy would be higher than 0.85. Statistical analysis of pairwise comparisons (Table 5.1) further showed that the performance in *-eyse* was significantly better than that of the other two postpositions. In contrast, the accuracy of *-ey* and *-(u)lo* were statistically the same.

Table 5.1: Statistical comparison of each postposition (PPMI-SVD): Two-sample *t*-test

Comparison	$\|t\|$	p
-ey vs. *-eyse*	6.080	$<$ .001***
-ey vs. *-(u)lo*	1.208	.243
-eyse vs. *-(u)lo*	5.929	$<$ .001***

Note. *** $<$ .001

SGNS (prediction-based)

Similar to the PPMI-SVD model, *-eyse* outperformed the other two postpositions in the SGNS model, as Figure 5.2 shows. This happened because of the same reason as with the PPMI-SVD model (i.e., *-eyse* has only two functions with LOC occupying a majority of the total corpus size).

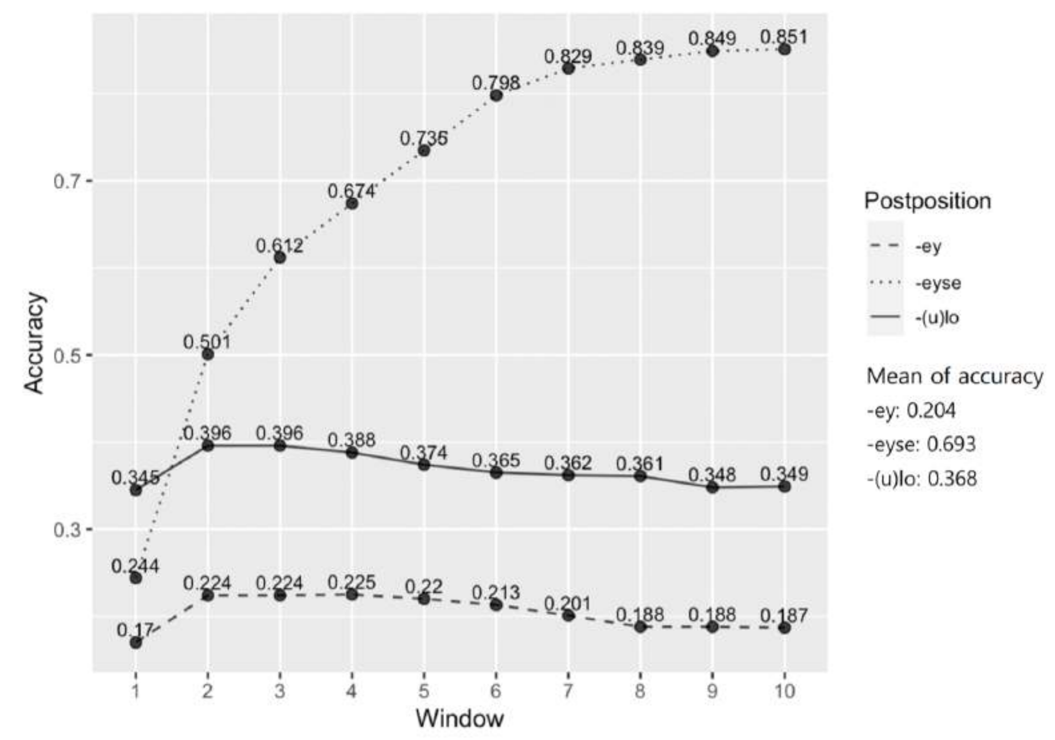

Figure 5.2: Classification accuracy by window size for the SGNS model

However, unlike the results from the PPMI-SVD model, the statistical analysis of pairwise comparisons (Table 5.2) shows that the accuracy levels of all the postpositions were different.

Table 5.2: Statistical comparison of each postposition (SGNS): Two-sample *t*-test

Comparison	$\|t\|$	p
-ey vs. *-eyse*	7.835	$< .001$***
-ey vs. *-(u)lo*	18.74	$< .001$***
-eyse vs. *-(u)lo*	5.203	$< .001$***

Note. *** $< .001$

5.2.2 Overall accuracy by postpositions: *-ey*, *-eyse*, and *-(u)lo*

-ey

Figure 5.3 shows the classification accuracy of each model for *-ey*. The PPMI-SVD model outperformed the SGNS model, with the classification accuracy levels being around 0.534 and 0.204, respectively. The mean accuracy levels of the two models were also significantly different from each other (t = 13.39, $p <$ .001 from a two-sample *t*-test). They showed different tendencies in terms of context window. The PPMI-SVD model achieved better classification accuracy as the context window size increased, whereas the SGNS model demonstrated low classification accuracy, regardless of context window sizes.

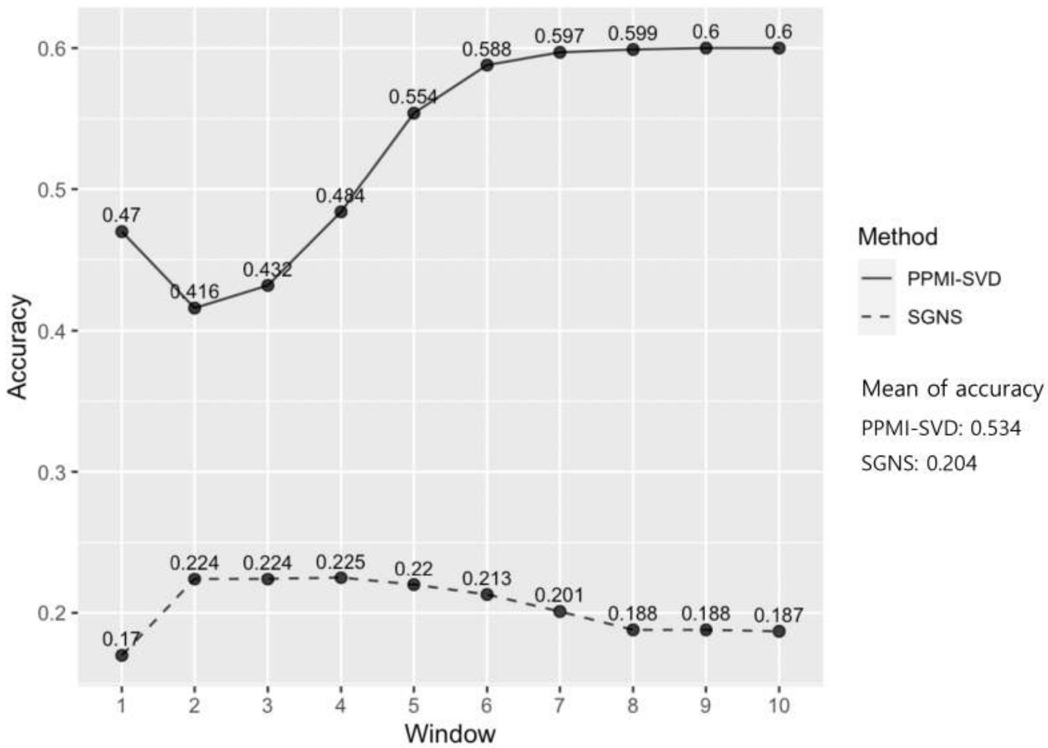

Figure 5.3: By-window-size accuracy for the two models: *-ey*

Model performance of *-ey* for the PPMI-SVD model varied by the types of functions, as shown in Figure 5.4 and Table 5.3. The average classification accuracy was the highest in LOC (0.602) and the lowest in INS (0.238); the other functions yielded accuracy ranging from 0.337 to 0.577. The by-function classification accuracy either increased or decreased. The functions whose classification accuracy increased were CRT, EFF, and LOC. Among these, LOC showed an accuracy of 0.396 in the window size of one but increased to 0.776 in the window size of ten. The remaining functions, which saw a decrease, were AGT, FNS, GOL, INS, and THM. Their accuracy tended to decrease as the window size increased. Although the functions which saw an increase in accuracy were fewer in number, the overall trend of accuracy change replicated the trend they produced. This was because they accounted for a larger portion of the entire corpus than the decreasing ones.

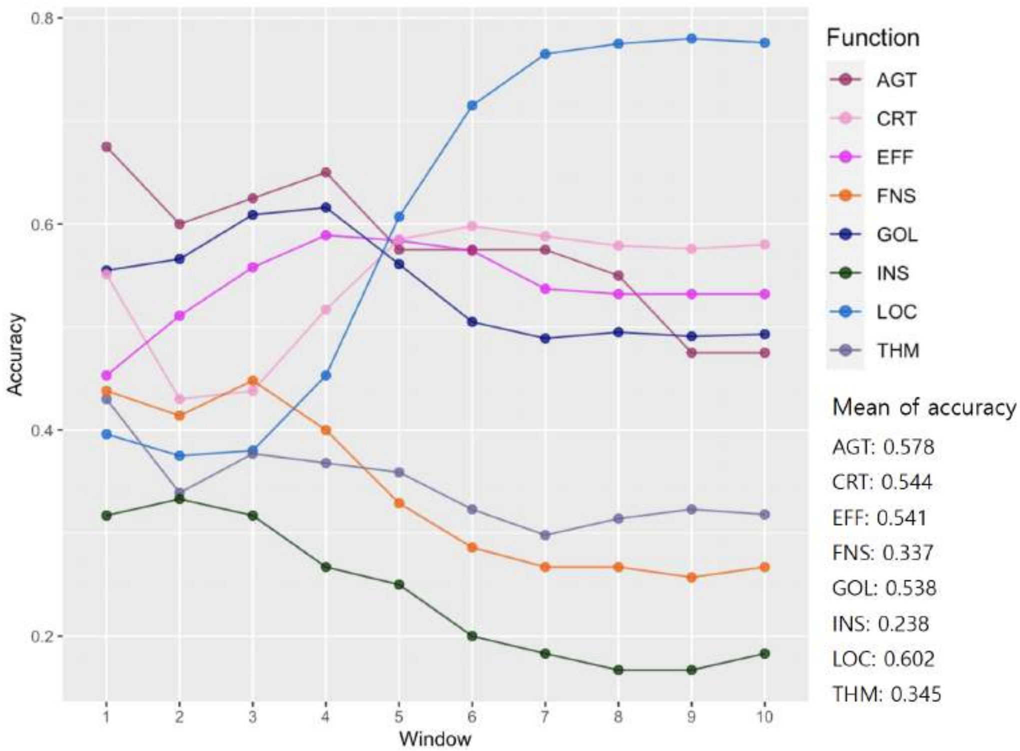

Figure 5.4: By-function accuracy curve for the PPMI-SVD model: *-ey*

Note. Abbreviation: AGT = agent; CRT = criterion; EFF = effector; FNS = final state; GOL = goal; INS = instrument; LOC = location; THM = theme

Table 5.3: By-function accuracy for the PPMI-SVD model: *-ey*

Window size	Classification accuracy							
	AGT	*CRT*	*EFF*	*FNS*	*GOL*	*INS*	*LOC*	*THM*
1	0.675	0.551	0.453	0.438	0.555	0.317	0.396	0.430
3	0.625	0.438	0.558	0.448	0.609	0.317	0.380	0.377
5	0.575	0.585	0.584	0.329	0.561	0.250	0.607	0.359
7	0.575	0.588	0.537	0.267	0.489	0.183	0.765	0.298
9	0.475	0.576	0.532	0.257	0.491	0.167	0.780	0.323
10	0.475	0.580	0.532	0.267	0.493	0.183	0.776	0.318

The classification accuracy of *-ey* for the SGNS model varied by the types of functions, as presented in Figure 5.5 and Table 5.4. The mean of classification accuracy was the highest in AGT (0.878) and the lowest in CRT (0.089); the other functions performed accuracy ranging from 0.092 to 0.616. The rate of change in accuracy by functions for this postposition seemed stable, except for AGT and INS, with no significant change as the window size increased. INS showed an accuracy of 0.417 in the window size of one, but its accuracy dropped in the window size of two to 0.100. For AGT, the window size of one produced an accuracy of 0.675, but as the window size increased, the variation of accuracy was huge, with the highest accuracy at 0.95.

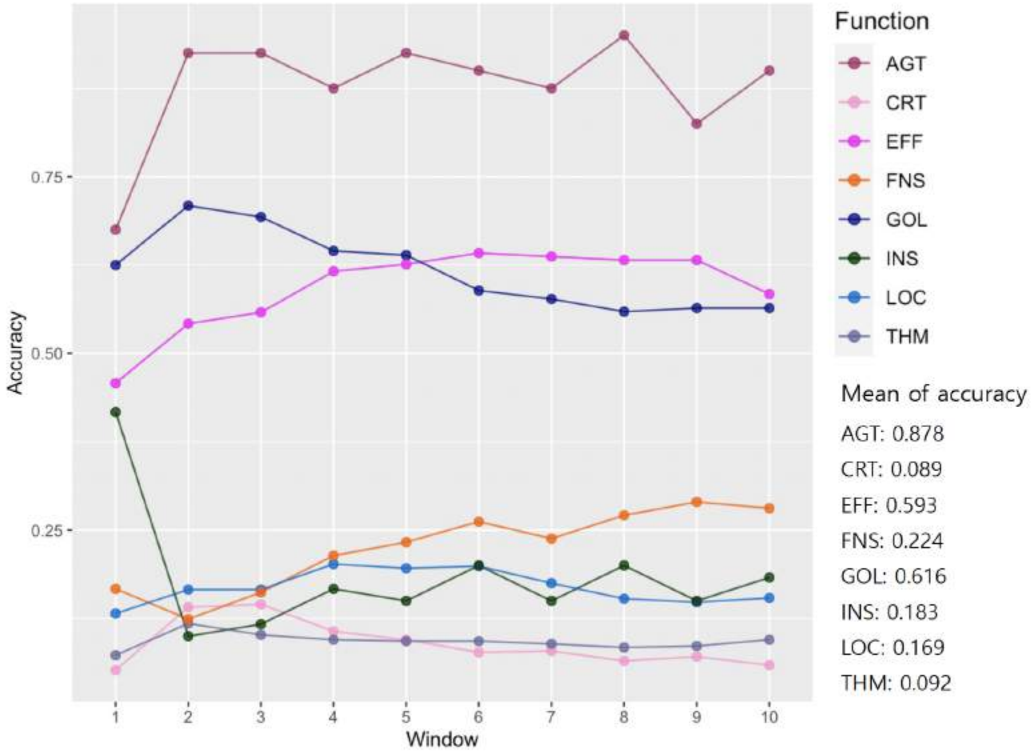

Figure 5.5: By-function accuracy curve for the SGNS model: *-ey*

Note. Abbreviation: AGT = agent; CRT = criterion; EFF = effector; FNS = final state; GOL = goal; INS = instrument; LOC = location; THM = theme

Table 5.4: By-function accuracy for the SGNS model: *-ey*

Window size	Classification accuracy							
	AGT	*CRT*	*EFF*	*FNS*	*GOL*	*INS*	*LOC*	*THM*
1	0.675	0.052	0.458	0.167	0.625	0.417	0.132	0.073
3	0.925	0.145	0.558	0.162	0.693	0.117	0.166	0.102
5	0.925	0.095	0.626	0.233	0.639	0.150	0.196	0.093
7	0.875	0.079	0.637	0.238	0.577	0.150	0.175	0.089
9	0.825	0.071	0.632	0.290	0.564	0.150	0.148	0.086
10	0.900	0.059	0.584	0.281	0.564	0.183	0.154	0.095

-eyse

Figure 5.6 shows the classification accuracy of each model for *-eyse*. The average levels for the PPMI-SVD model and the SGNS model were around 0.773 and 0.693, respectively. As shown in Figure 5.6, the PPMI-SVD model and the SGNS model demonstrate a similar trend in which the accuracy of each model increased as the context window size increased. Statistical analysis of pairwise comparisons showed that there was no difference in the overall classification accuracy of these two models (t = 1.157, p = 0.262 from a two-sample t-test).

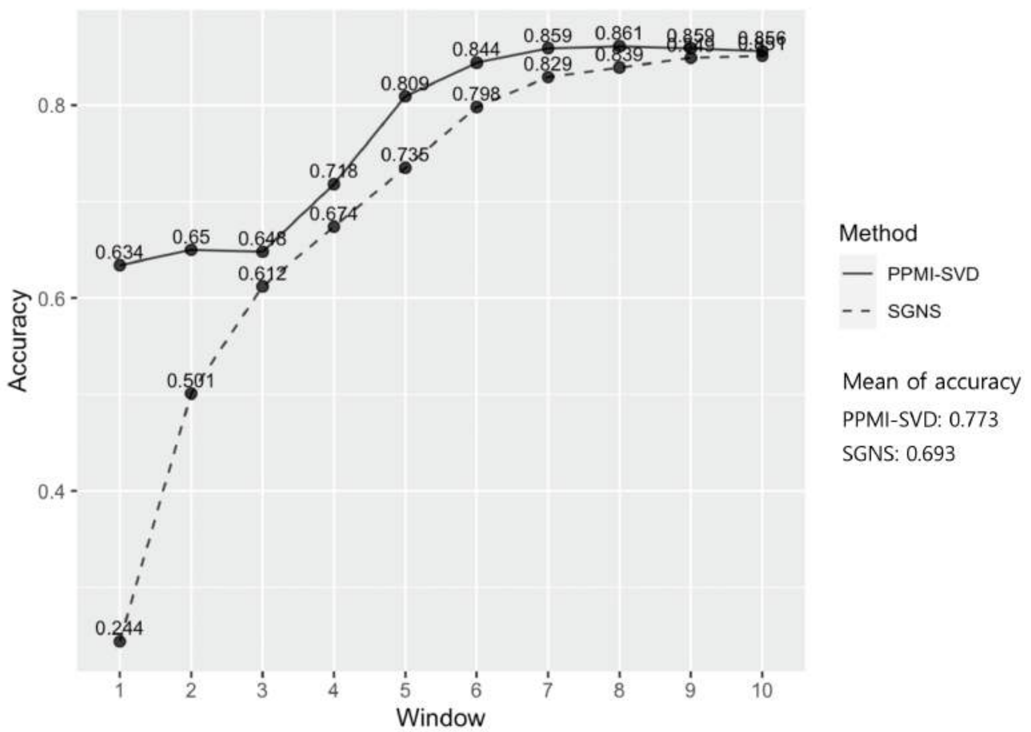

Figure 5.6: By-window-size accuracy for the two models: *-eyse*

Model performance of *-eyse* for the PPMI-SVD model showed different outcomes by the types of functions, as shown in Figure 5.7 and Table 5.5. LOC achieved an accuracy of 0.627 at the window size of one, but increased up to 0.981 as the context window size increased. In contrast, SRC reached an accuracy of 0.678 at the window size of one, but decreased to 0.062 as the context window size increased.

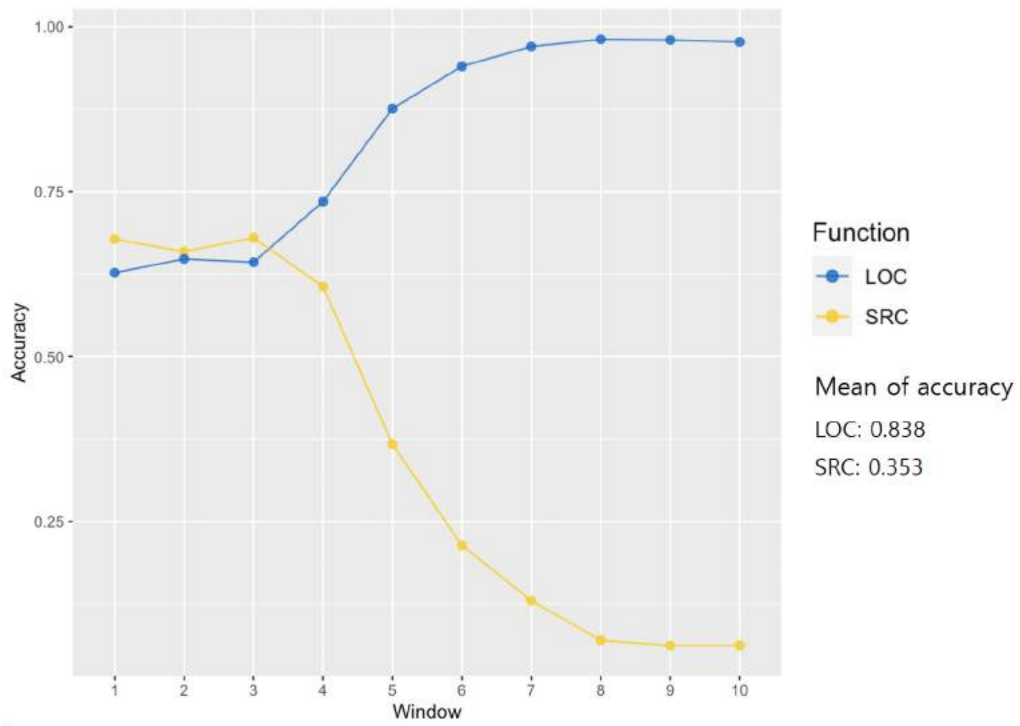

Figure 5.7: By-function accuracy curve for the PPMI-SVD model: *-eyse*
Note. Abbreviation: LOC = location; SRC = source

Table 5.5: By-function accuracy for the PPMI-SVD model: *-eyse*

Window size	Classification accuracy	
	LOC	*SRC*
1	0.627	0.678
3	0.643	0.680
5	0.876	0.367
7	0.970	0.130
9	0.980	0.062
10	0.977	0.062

Similar to the PPMI-SVD model, LOC performed better than SRC in the SGNS model. Figure 5.8 and Table 5.6 show that LOC had an accuracy of 0.131 in the window size of one and increased to 0.919 in the window size of ten. In contrast, SRC reached an accuracy of 0.988 at the window size of one, but it decreased as the window size increased. The overall trend of accuracy change was similar to that of LOC. This was because the occurrence of LOC in the corpus accounted for a larger portion.

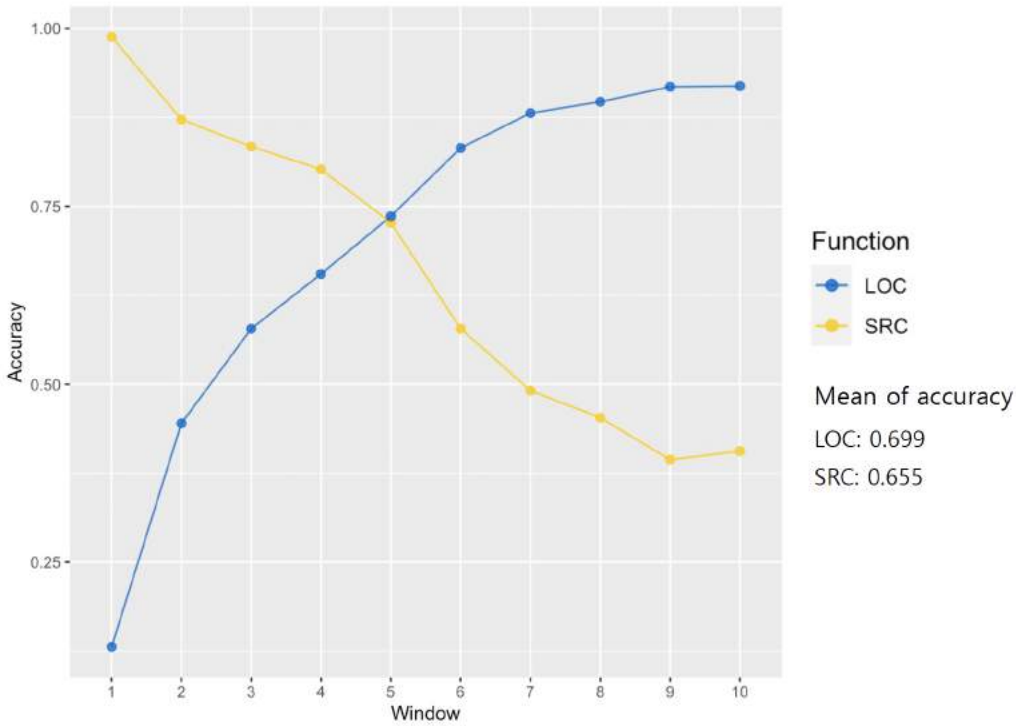

Figure 5.8: By-function accuracy curve for the SGNS model: -*eyse*

Note. Abbreviation: LOC = location; SRC = source

Table 5.6: By-function accuracy for the SGNS model: -*eyse*

Window size	Classification accuracy	
	LOC	*SRC*
1	0.131	0.988
3	0.578	0.834
5	0.736	0.727
7	0.881	0.491
9	0.918	0.394
10	0.919	0.406

Overall, the PPMI-SVD and SGNS models showed similar results to each other, i.e., LOC showed a low accuracy in the smaller window size, but increased as the window size increased. In contrast, the accuracy of SRC reached high accuracy in the smaller window size, but decreased as the window size increased. Considering that the smaller windows work better for syntactic representation and the larger for semantic (e.g., Jurafsky and Martin, 2019, Levy et al., 1999), LOC may perform more semantically than syntactically, and vice versa for SRC.

-(u)lo

The average classification accuracy levels of *-(u)lo* for the PPMI-SVD model and the SGNS model were around 0.567 and 0.368, respectively. As shows in Figure 5.9, the PPMI-SVD model outperformed the SGNS model, and the mean accuracy levels of the two models were significantly different from each other (t = 12.458, $p <$ .001 from a two-sample t-test).

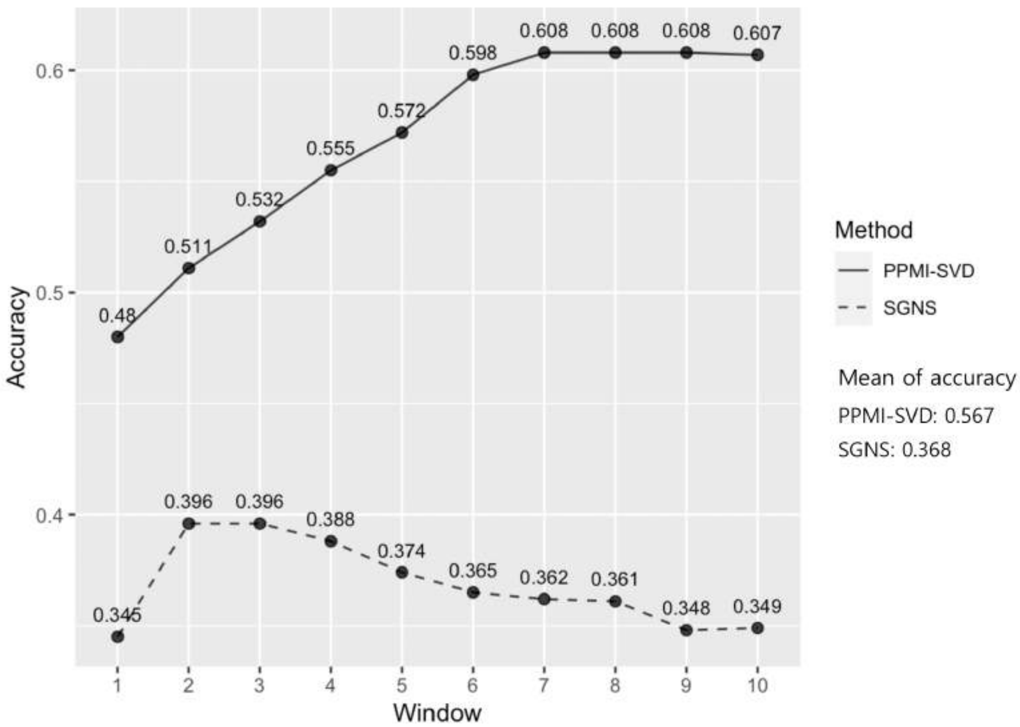

Figure 5.9: By-window-size accuracy for the two models: *-(u)lo*

Model performance of the PPMI-SVD model for *-(u)lo* varied by the types of functions, as shown in Figure 5.10 and Table 5.7. The average classification accuracy was the highest for DIR (0.777) and the lowest for LOC (0.233); the other functions yielded accuracy ranging from 0.344 to 0.583. The by-function classification accuracy either increased or decreased. The functions whose classification accuracy increased were FNS and DIR. The remaining functions, which decreased, were CRT, EFF, INS, and LOC. This result may be due to the possibility that the accuracy of the PPMI-SVD model was affected by the corpus size of each function, because DIR and FNS are the functions that account for a majority of the total corpus size.

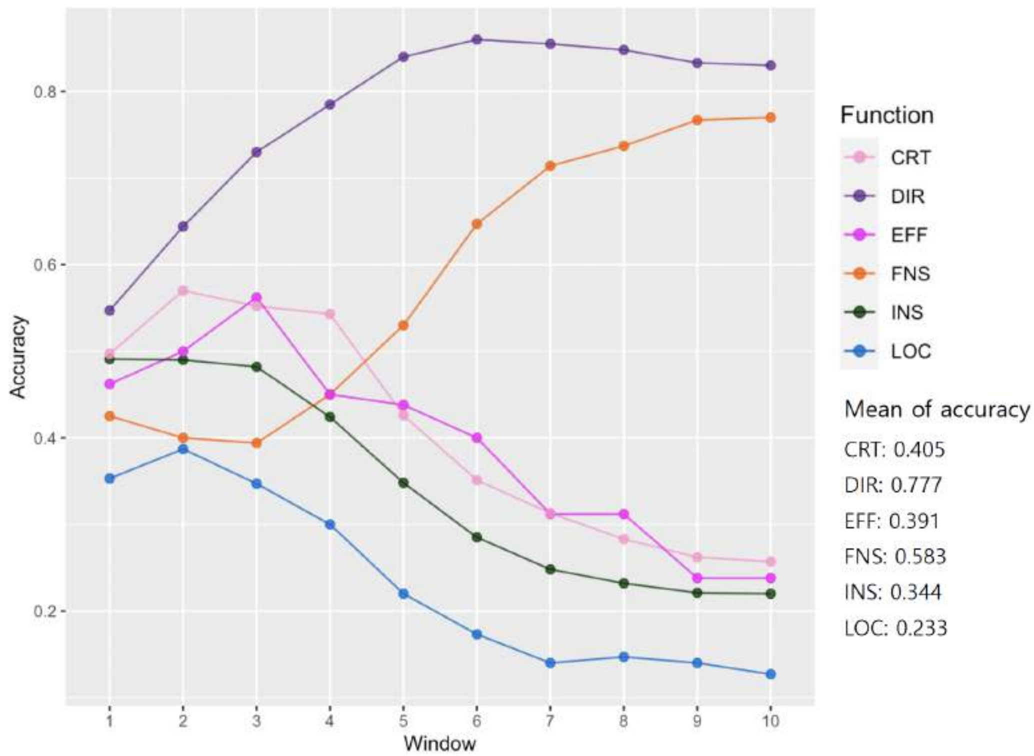

Figure 5.10: By-function accuracy curve for the PPMI-SVD model: *-(u)lo*

Note. Abbreviation: CRT = criterion; DIR = direction; EFF = effector; FNS = final state; INS = instrument; LOC = location

Table 5.7: By-function accuracy for the PPMI-SVD model: *-(u)lo*

Window size	Classification accuracy					
	CRT	*DIR*	*EFF*	*FNS*	*INS*	*LOC*
1	0.497	0.547	0.462	0.425	0.491	0.353
3	0.552	0.730	0.562	0.394	0.482	0.347
5	0.426	0.840	0.438	0.530	0.348	0.220
7	0.313	0.855	0.312	0.714	0.248	0.140
9	0.262	0.833	0.238	0.767	0.221	0.140
10	0.257	0.830	0.238	0.770	0.220	0.127

The classification accuracy of the SGNS model for *-(u)lo* also varied by the types of functions, as presented in Figure 5.11 and Table 5.8. The mean of classification accuracy was the highest for DIR (0.774) and the lowest for FNS (0.058); the other functions performed accuracy ranging from 0.141 to 0.634. The change of accuracy for all the functions seemed stable.

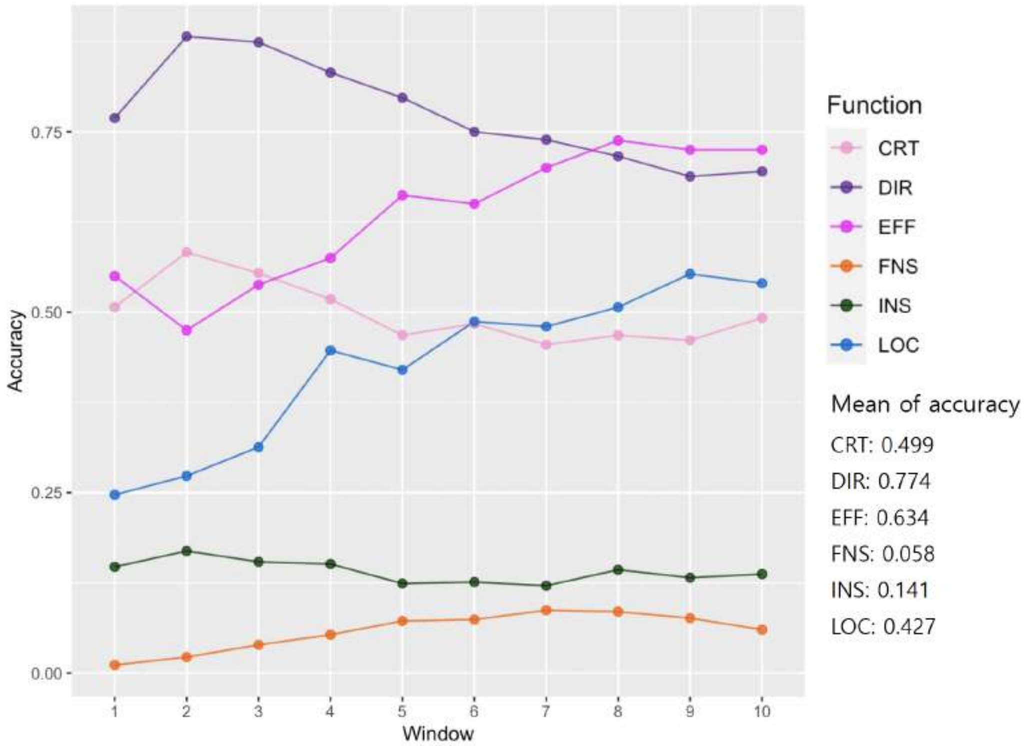

Figure 5.11: By-function accuracy curve for the SGNS model: *-(u)lo*

Note. Abbreviation: CRT = criterion; DIR = direction; EFF = effector; FNS = final state; INS = instrument; LOC = location

Table 5.8: By-function accuracy for the SGNS model: -*(u)lo*

Window size	Classification accuracy					
	CRT	*DIR*	*EFF*	*FNS*	*INS*	*LOC*
1	0.507	0.769	0.550	0.011	0.147	0.247
3	0.554	0.874	0.538	0.039	0.154	0.313
5	0.468	0.797	0.662	0.072	0.124	0.420
7	0.455	0.739	0.700	0.087	0.121	0.480
9	0.461	0.688	0.725	0.076	0.132	0.553
10	0.492	0.695	0.725	0.060	0.137	0.540

5.2.3 Correlation between corpus size and classification accuracy

As shown in Sections 5.2.1 and 5.2.2, the model performance was similar to the accuracy patterns of the functions of each postposition occurring the most in the corpus data. This implies that the classification accuracy may be affected by the corpus size of each function. To further explore possible relationships between the size of training corpora and the models' by-function classification accuracy, I conducted a correlation analysis by postposition. For this task, I calculated the Pearson Correlation between the mean accuracy of each model and of each function for these postpositions per context window size.

-ey

Of the eight functions of *-ey*, LOC and CRT accounted for the largest portion of the total corpus. As shown in Table 5.9, the mean accuracy of the PPMI-SVD model correlated highly with that of LOC and CRT. In contrast, the rest of functions yielded negative correlation values (except for EFF) because they accounted for a small portion of the total corpus size. On the other hand, the SGNS model did not seem to demonstrate any meaningful correlation between the corpus size and the model performance. One possible reason for this difference is that the SGNS model was not based on token frequency but on type frequency.

Table 5.9: Correlation between the accuracy of each model and of each function for *-ey* by window size

Function	Corpus size	Correlation	
		PPMI-SVD	SGNS
LOC	1,780	0.983	0.797
CRT	1,516	0.907	0.854
THM	448	-0.687	0.765
GOL	441	-0.854	0.669
FNS	216	-0.967	-0.377
EFF	198	0.207	0.299
INS	69	-0.972	-0.713
AGT	47	-0.737	0.631

Note. Abbreviation: AGT = agent; CRT = criterion; EFF = effector; FNS = final state; GOL = goal; INS = instrument; LOC = location; THM = theme

-eyse

The occurrence of LOC accounted for more than 85% of the total corpus. As shown in Table 5.10, the overall accuracy has a strong positive correlation with LOC and a negative correlation with SRC for both models. This convergence of results across the two models may be due to an overwhelmingly larger number of LOC than SRC in the corpus, which increased word types as well.

Table 5.10: Correlation between the accuracy of each model and of each function for *-eyse* by window size

Function	Corpus size	Correlation	
		PPMI-SVD	SGNS
LOC	4,206	0.998	0.998
SRC	647	-0.971	-0.904

Note. Abbreviation: LOC = location; SRC = source

-(u)lo

Of the six functions of *-(u)lo*, FNS and DIR accounted for the largest portion of the total corpus. As presented in Table 5.11, the PPMI-SVD model showed that the mean accuracy of the model and of each function were highly correlated with FNS and DIR. On the other hand, the other functions showed negative correlation values because they accounted for the smaller portion of the corpus size. However, the result showed no clear tendency in the correlation between the corpus size and the overall accuracy of each function in the SGNS model. This is possibly due to the same reason as *-ey*, which was operated based on type frequency rather than token frequency.

Table 5.11: Correlation between the accuracy of each model and of each function for *-(u)lo* by window size

Function	Corpus size	Correlation	
		PPMI-SVD	SGNS
FNS	1,681	0.903	-0.273
DIR	1,449	0.952	0.907
INS	739	-0.952	0.564
CRT	593	-0.862	0.716
LOC	158	-0.949	-0.446
EFF	88	-0.797	-0.671

Note. Abbreviation: CRT = criterion; DIR = direction; EFF = effector; FNS = final state; INS = instrument; LOC = location

Overall, the PPMI-SVD model was affected by the corpus size more than the SGNS model. The performance of the PPMI-SVD model was similar to accuracy patterns of the functions occupying the larger portion of each postposition. This is because, the word-word matrix was used in the process of converting words to vectors, so it was sensitive to the token frequency (Jurafsky and Martin, 2019). On the other hand, one-hot encoding was used for the SGNS model in the same process, so it relied on the type frequency (Mikolov et al., 2013a).

5.3 Visualization system: clusters and co-occurring words

The visualization system aimed to identify the word-level embeddings interactively in order to see the changes of the clusters between each function of the postpositions and the co-occurring words. In this section, I provide findings of the visualization that I developed. I recommend seeing these findings while demonstrating the visualization system together [1].

5.3.1 Changes of clusters by environments (model and window size)

The visualization system showed word clusters through distributional semantic maps. To statistically explore changes of the clusters by model and window size, I performed a series of cluster analysis. This allows exploratory data analysis in which observations are divided into groups that share common characteristics (Romersburg, 1984). Among the many kinds of cluster analyses such as Hierarchical clustering (Sibson, 1973), *K*-means clustering (MacQueen, 1967), and Density-based clustering (Sander et al., 1998), I used the Density-based clustering for analysis. This is due to the advantage it has of generating groups based on the density of the distribution data that allows us to discover groups of arbitrary shape as well as to distinguish noise (Sander et al., 1998). For the cluster analysis, I used the *dbscan* package (Hahsler et al., 2019) through R (R version 3.6.2; R Core Team, 2019). I then created density maps for each distributional map to see the optimal number of groups by *dbscan*.

Figures 5.12-5.17 present the bar chart for the number of grouping results obtained from the density cluster for each postposition per window size, together with a distributional semantic map where the postposition showed the best classification accuracy. In the PPMI-SVD model (Figures

[1]PostEmbedding, the first visualization system is available at: https://seongmin-mun.github.io/VisualSystem/Major/PostEmbedding/index.html

5.12-5.14), the bar chart showed that each distributional semantic map engaged in one or two groups in the end, and high-frequency words were located in the center of each distributional semantic map. This is because the PPMI-SVD model worked based on the frequency of tokens.

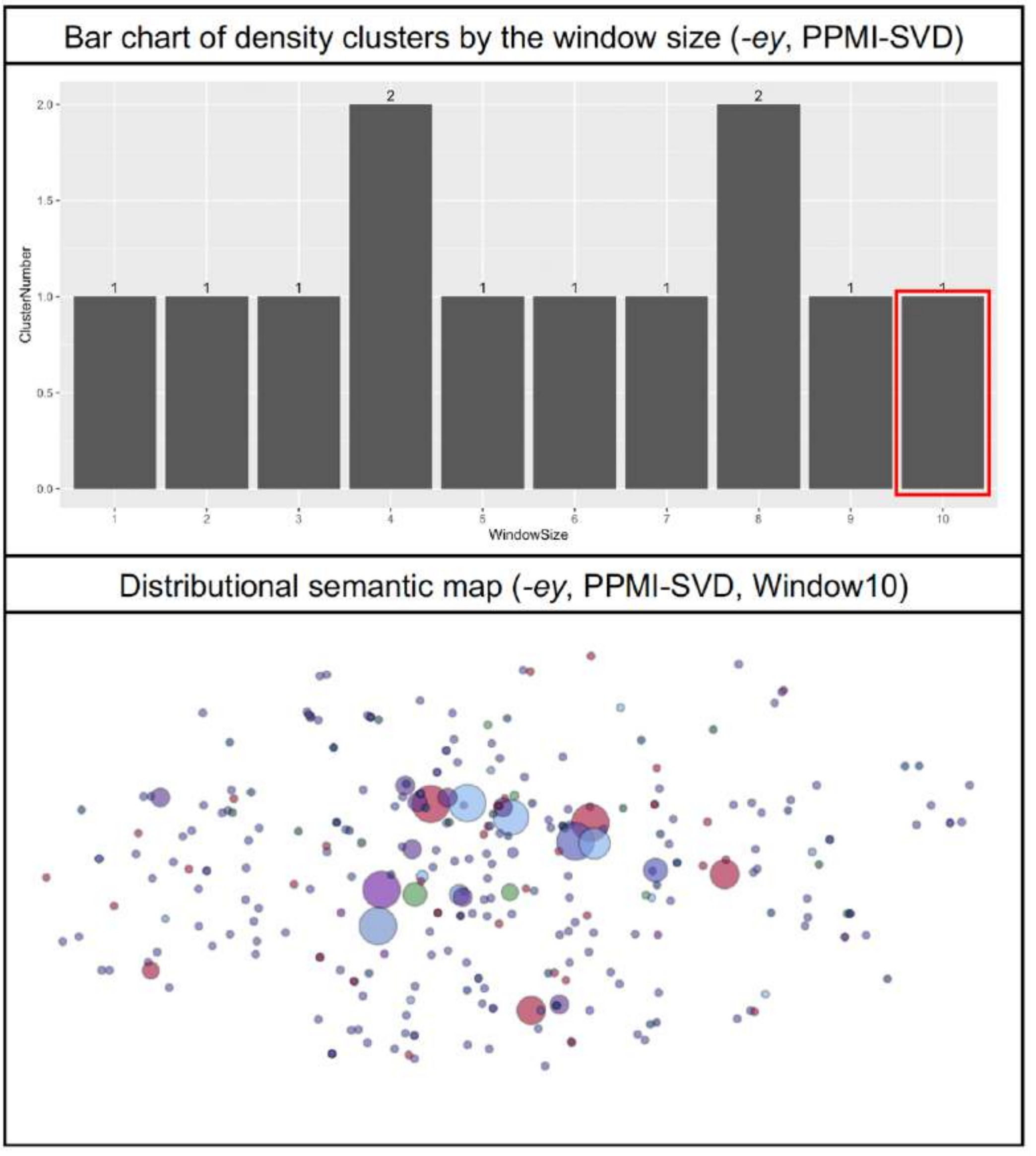

Figure 5.12: Bar chart of density cluster result and distributional semantic map for *-ey* (PPMI-SVD). Red in graph = the size of window showing the highest accuracy.

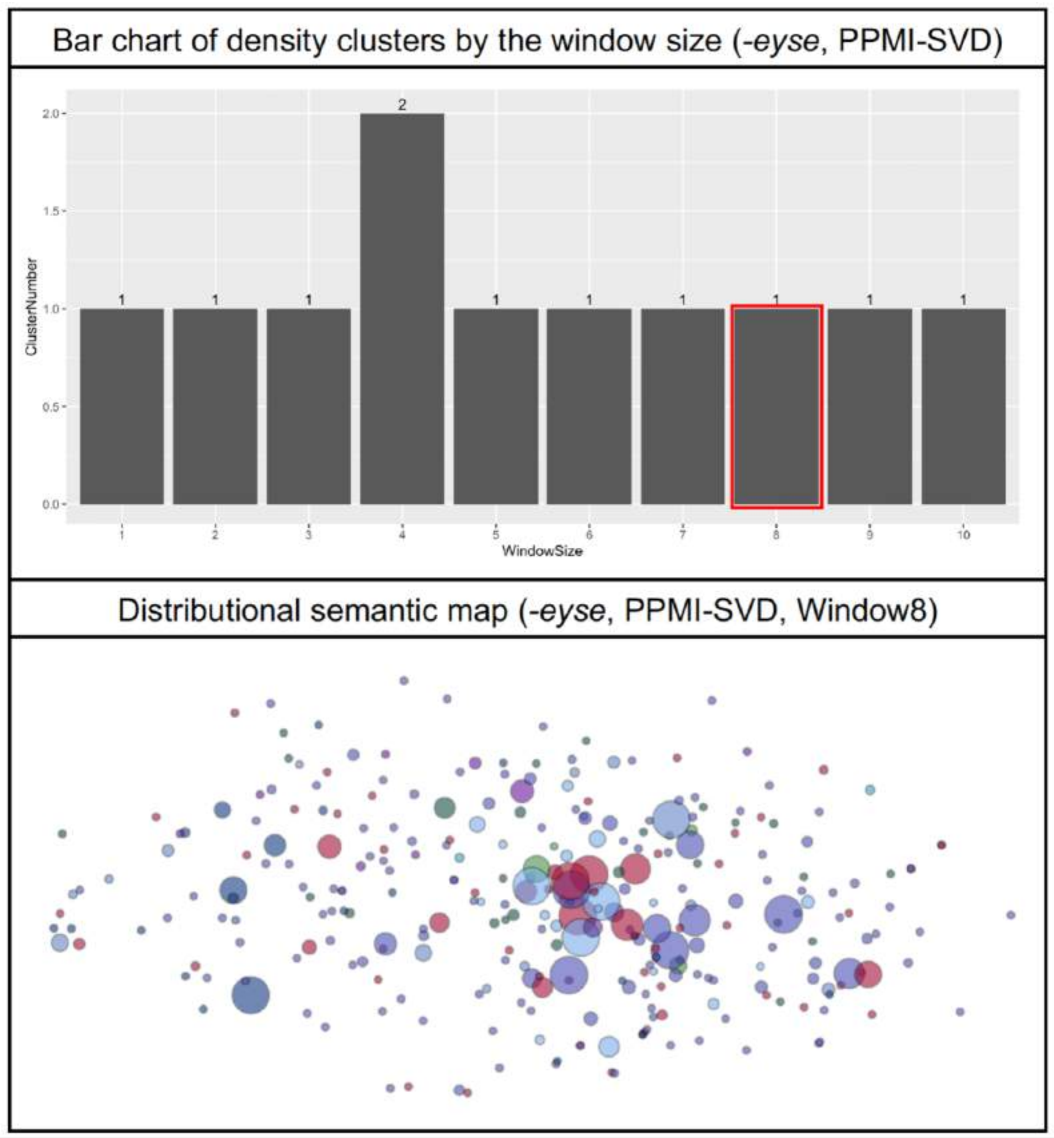

Figure 5.13: Bar chart of density cluster result and distributional semantic map for *-eyse* (PPMI-SVD). Red in graph = the size of window showing the highest accuracy.

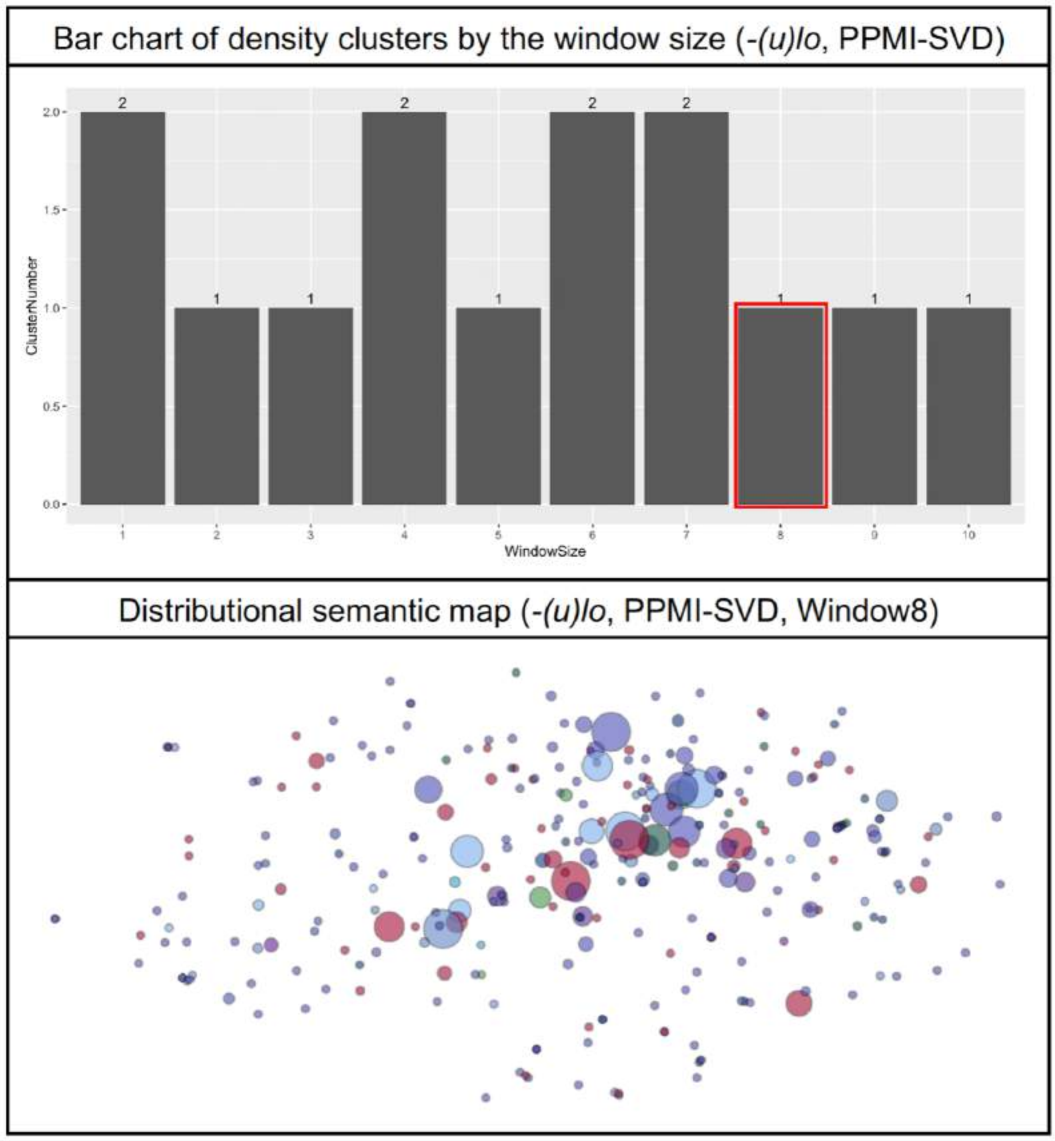

Figure 5.14: Bar chart of density cluster result and distributional semantic map for -*(u)lo* (PPMI-SVD). Red in graph = the size of window showing the highest accuracy.

In contrast, the SGNS model (Figures 5.15-5.17), the outcomes of the density cluster result showed that all distributional semantic maps were gathered as one group. Moreover, the words seem to be widely distributed, regardless of word frequency.

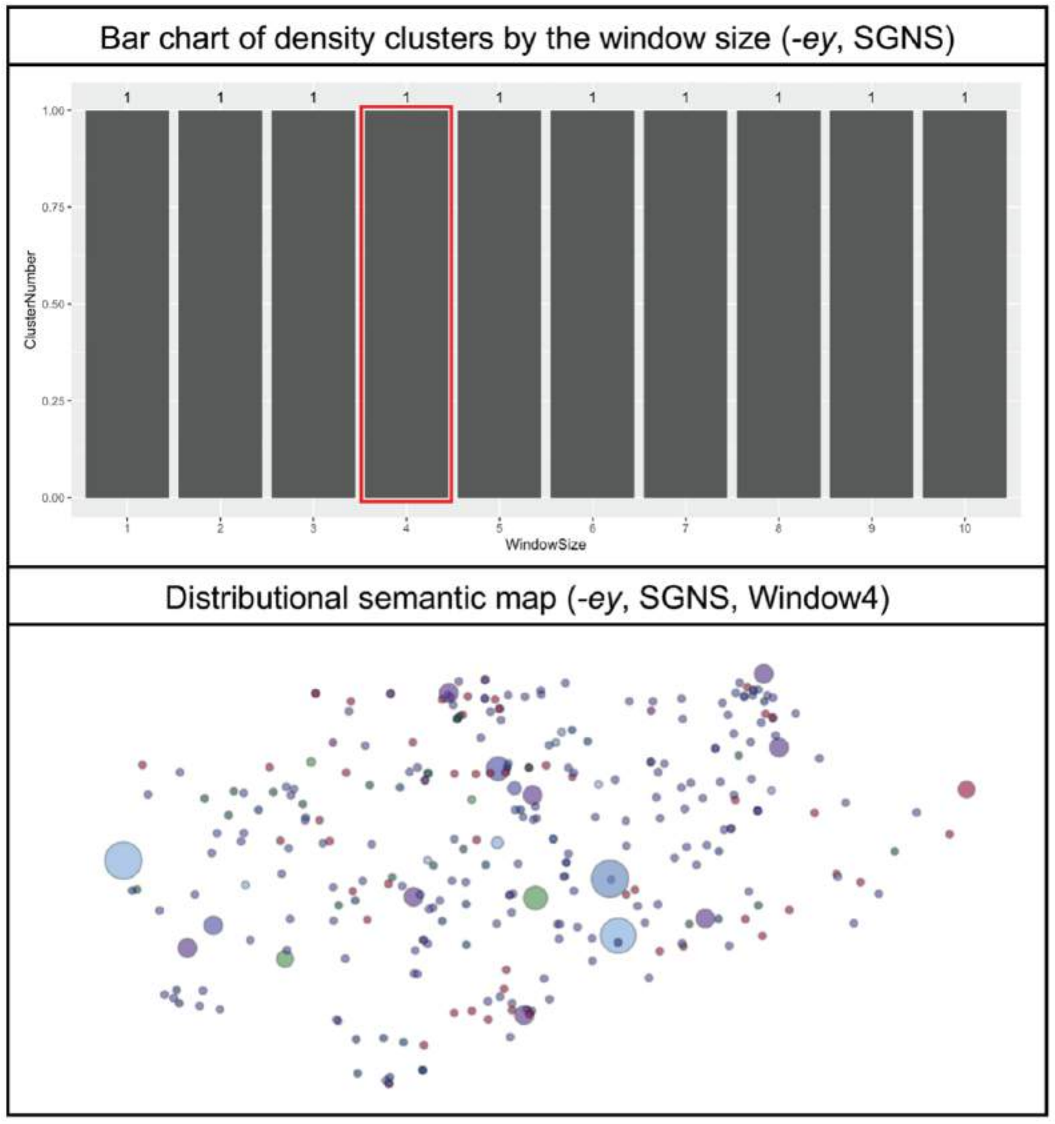

Figure 5.15: Bar chart of the density cluster result and distributional semantic map for *-ey* (SGNS). Red in graph = the size of window showing the highest accuracy.

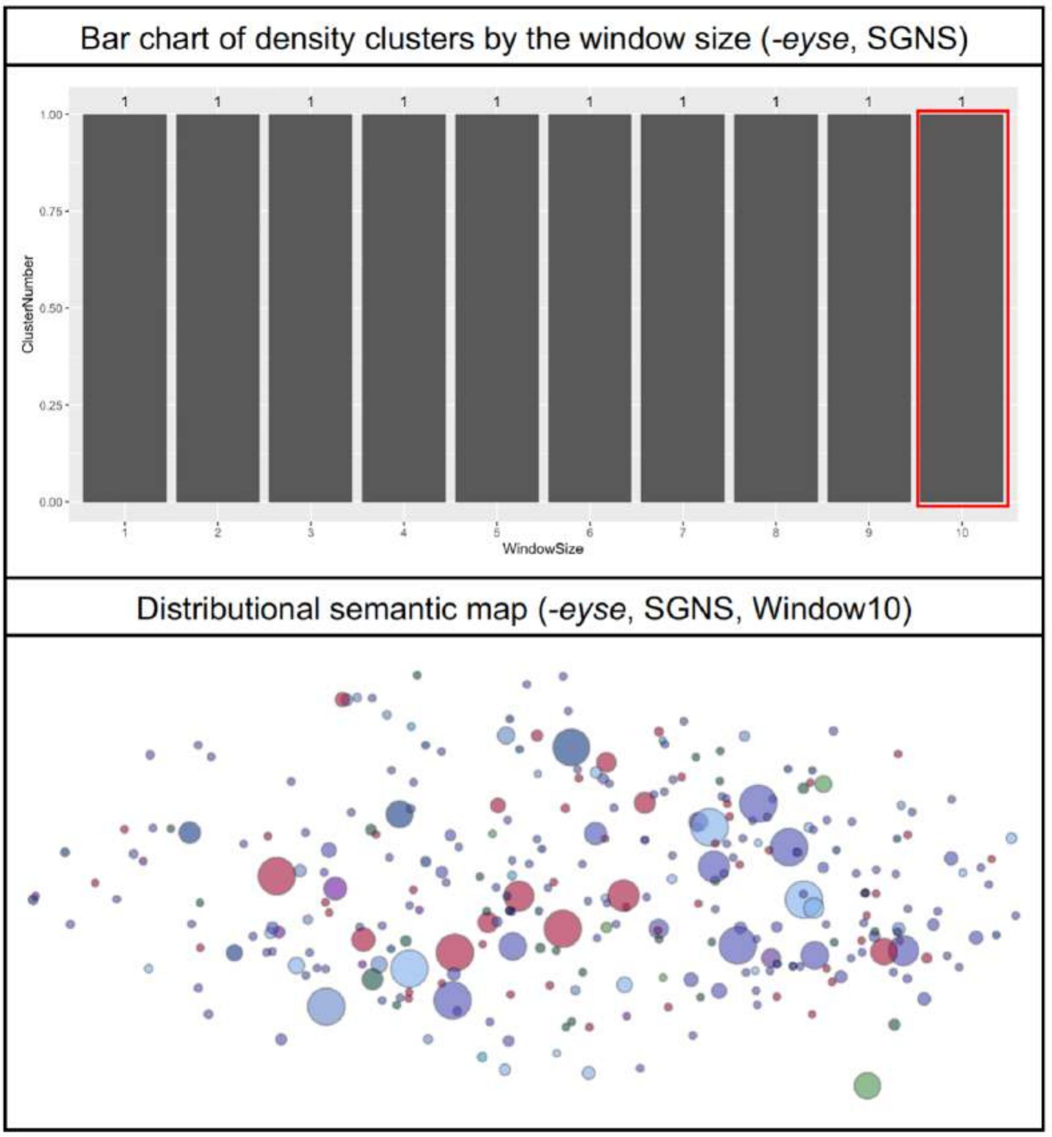

Figure 5.16: Bar chart of the density cluster result and distributional semantic map for -*eyse* (SGNS). Red in graph = the size of window showing the highest accuracy.

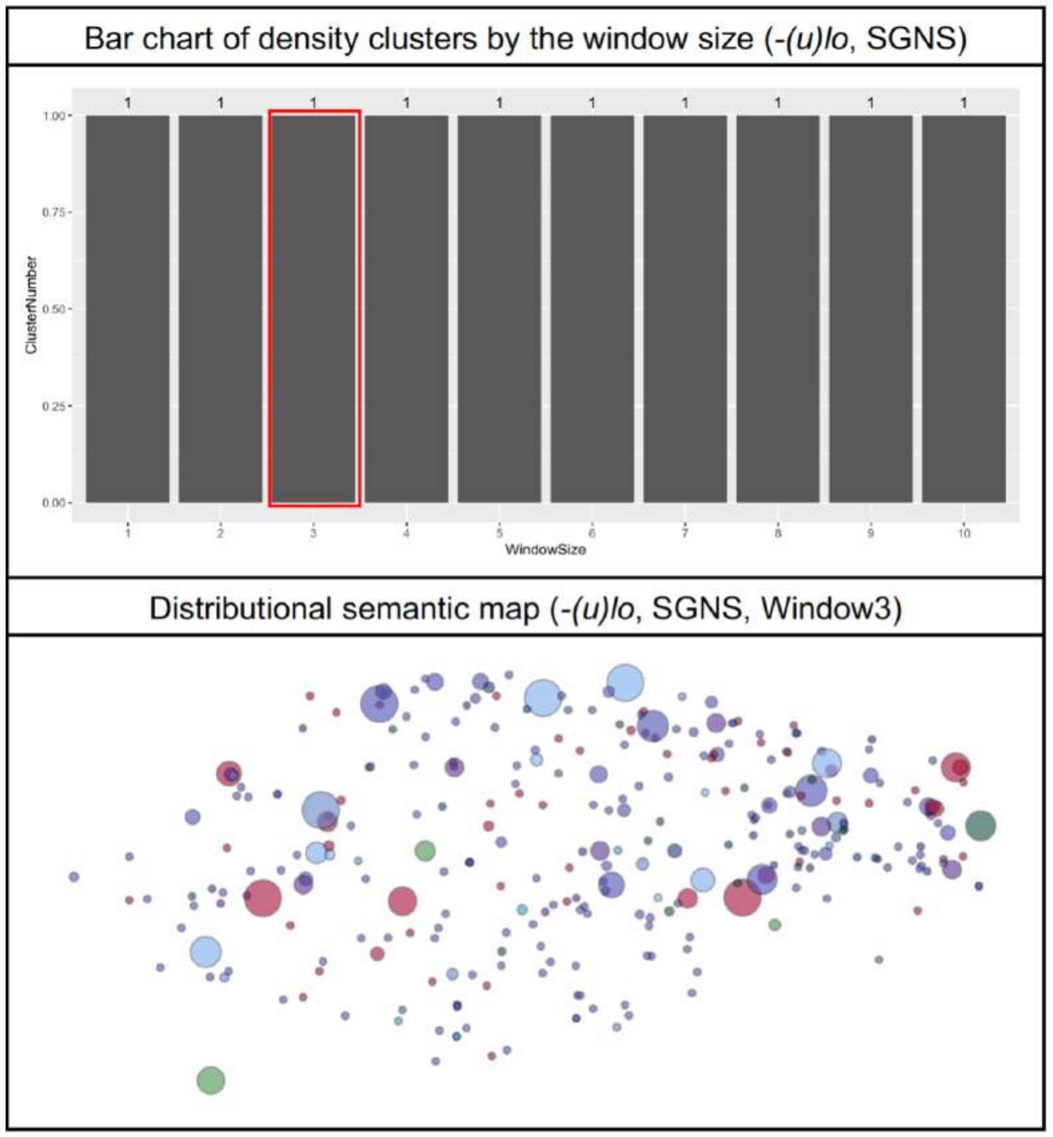

Figure 5.17: Bar chart of the density cluster result and distributional semantic map for *-(u)lo* (SGNS). Red in graph = the size of window showing the highest accuracy.

In summary, the most frequent words were placed in the center of the cluster for the PPMI-SVD model. This is because it operated on the basis of token frequency. In contrast, the SGNS model was based on type frequency, and the words were distributed across all window sizes, regardless of token frequency. However, the cluster analysis showed that the distributional semantic maps for each model were not so much different in terms of the final product of grouping (producing one or two groups for each model), indicating that the clusters created did not differ significantly from each other by environments.

5.3.2 Changes of co-occurring words by the functions of each postposition

-ey

Figure 5.18 shows the embedding results of when the highest classification accuracy performance was obtained (0.600; PPMI-SVD with the window size of nine). Similar to the other PPMI-SVD models (Section 5.3.1), words that appeared frequently in the entire corpus were at the center of the cluster.

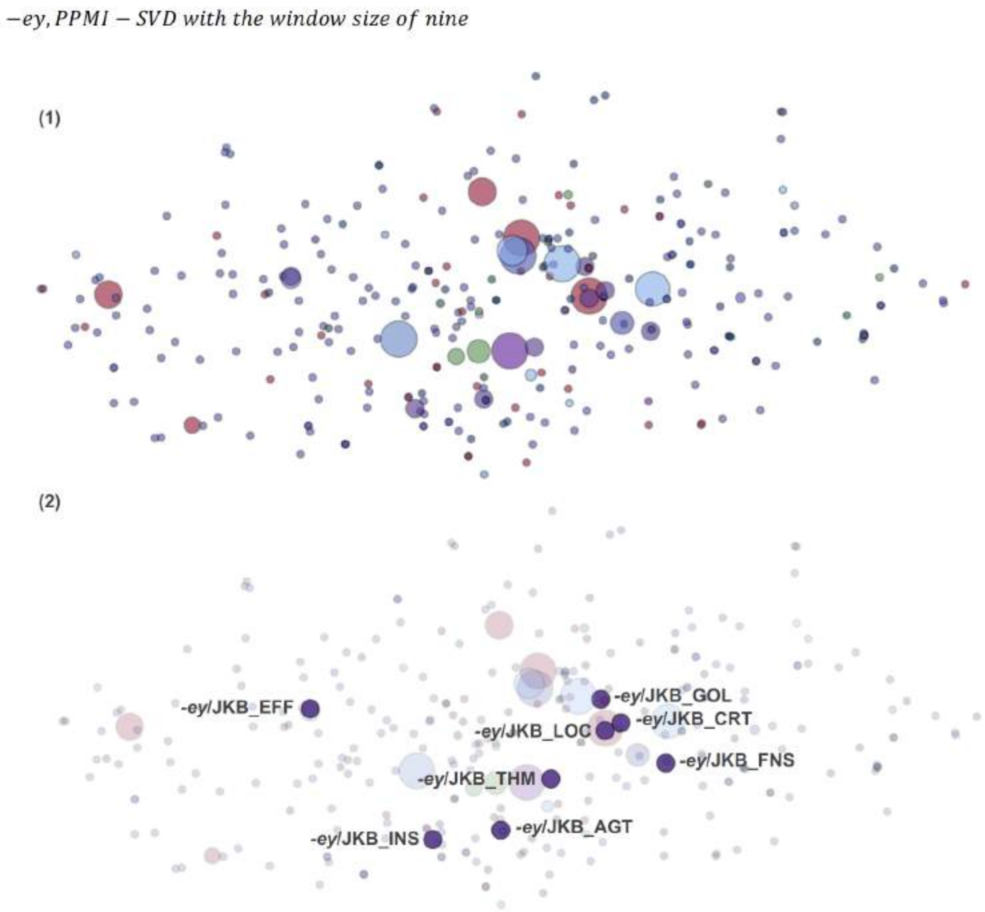

Figure 5.18: Distributional semantic map for *-ey* (PPMI-SVD; window size of nine)

Figures 5.19 and 5.20 show clusters between each function of *-ey* and co-occurring words (only the top five frequent words). The value in each circle indicates the frequency of each word, the double arrows indicate that the two words are affected by each other, and values next to the arrows show cosine similarity scores between each postposition and its co-occurring words. When *-ey* was used as LOC, the most frequent word was *iss-'to exist'/VV* (765 instances). For CRT, *elkwul-'face'/NNG* (40 instances) was frequently used. In the case of THM, the most frequently used word was *kukes-'that or it'/NP*

(47 instances). Considering that this word is often used as placeholder for theme (Choo and Kwak, 2008), the close association between *-ey* and *kukes-'that or it'/NP* is reasonable. For GOL, the verb *takase-'come close'/VV* (2 instances) related to the motion was included in the list of co-occurring words, and *chengnyen-'young boy'/NNG* was most used among the other words. The two words showed high similarity but showed a low co-occurrence frequency. This is due to two words only appeared when *-ey* was used as GOL.

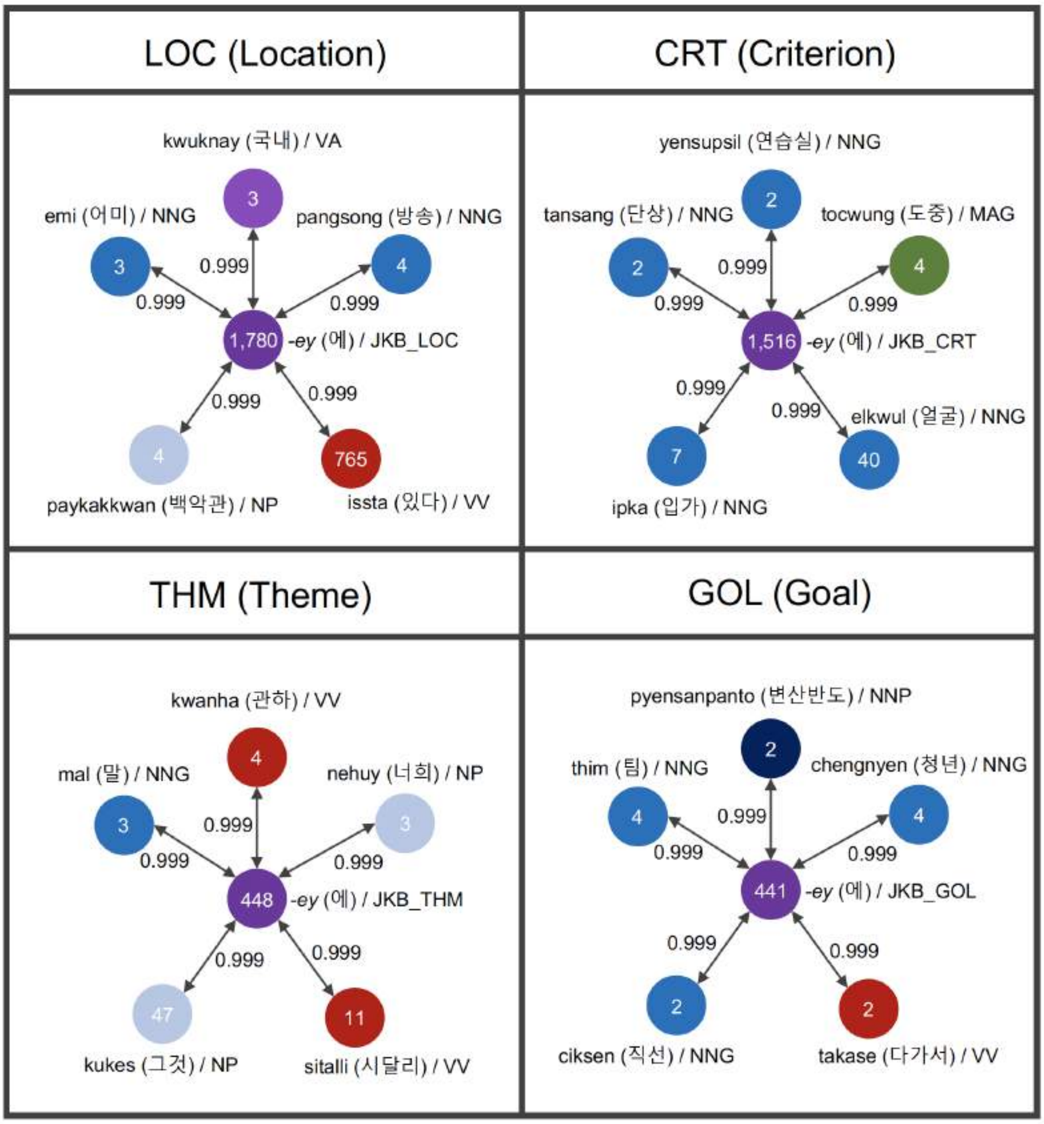

Figure 5.19: By-function co-occurring words for *-ey*: LOC, CRT, THM, and GOL

Note. Abbreviation: JKB = adverbial postposition; MAG = general adverb; NNG = common noun; NP = pronoun; VV = verb

When *-ey* was used as FNS, the most frequent word was *ipen-'this time'/NNG* (95 instances). For EFF, the noun *ttaymwun-'reason'/NNG* (80 instances), which relates to cause and effect, was ranked as the most frequent co-occurring word. In the case of INS and AGT, two conjunctive adverbs, *kuliko-'and'/MAJ* (67 instances) and *kulena-'but'/MAJ* (95 instances) appeared frequently.

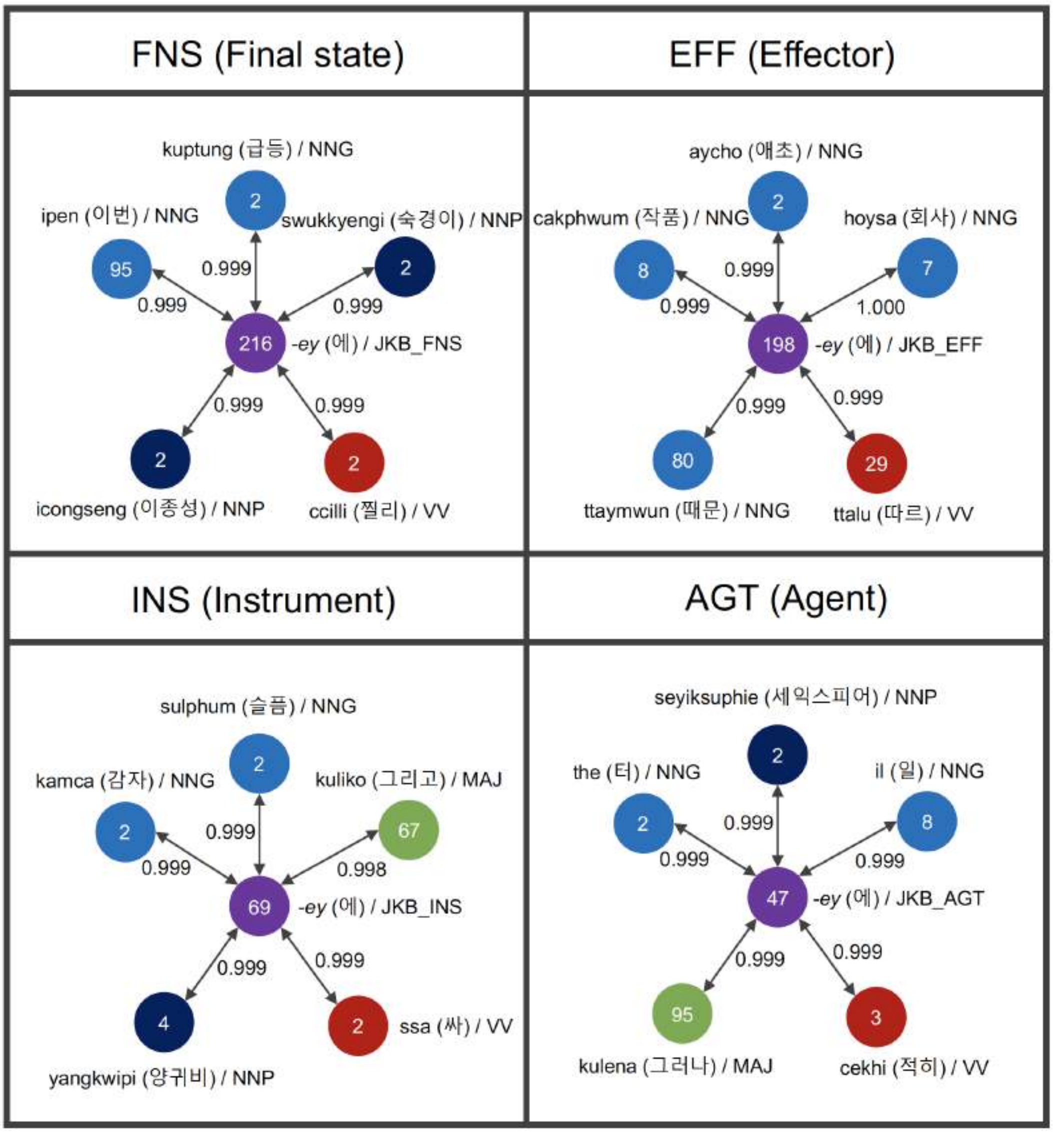

Figure 5.20: By-function co-occurring words for *-ey*: FNS, EFF, INS, and AGT

Note. Abbreviation: JKB = adverbial postposition; MAJ =conjunctive adverb; NNB = bound noun; NNG = common noun; NNP = proper noun; VV = verb

-eyse

Figure 5.21 displays the embedding outcomes of the best performance achieved (0.861; PPMI-SVD with the window size of eight). The most frequent appearing words in the corpus were at the center of the cluster, just like the other PPMI-SVD models, with both SRC and LOC located in the center.

Figure 5.21: Distributional semantic map for *-eyse* (PPMI-SVD; window size of eight)

As shown in Figure 5.22, when *-eyse* was used as LOC, the verb *na-'come'/VV* (106 instances) was included in the list of co-occurring words, and *soli-'sound'/NNG*

(121 instances) was the noun that was frequently used. In the case of SRC, the noun *wi-'up'/NNG* (50 instances) related to direction, was included in the list of co-occurring words. This shows that word semantics has a strong connection to the functions of *-eyse*.

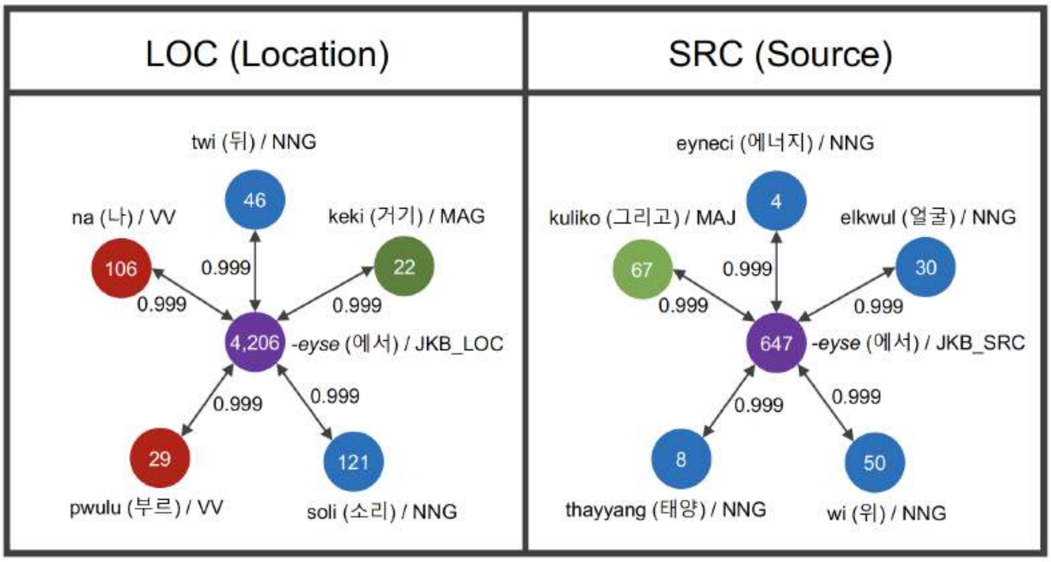

Figure 5.22: By-function co-occurring words for *-eyse*: LOC and SRC

Note. Abbreviation: JKB = adverbial postposition; MAG = general adverb; MAJ = conjunctive adverb; NNG = common noun; NP = pronoun; VV = verb

-(u)lo

For *-(u)lo*, the highest classification accuracy was obtained when the PPMI-SVD model used with the window size of nine. Figure 5.23 shows the result that the words that appear frequently in the entire corpus are at the center.

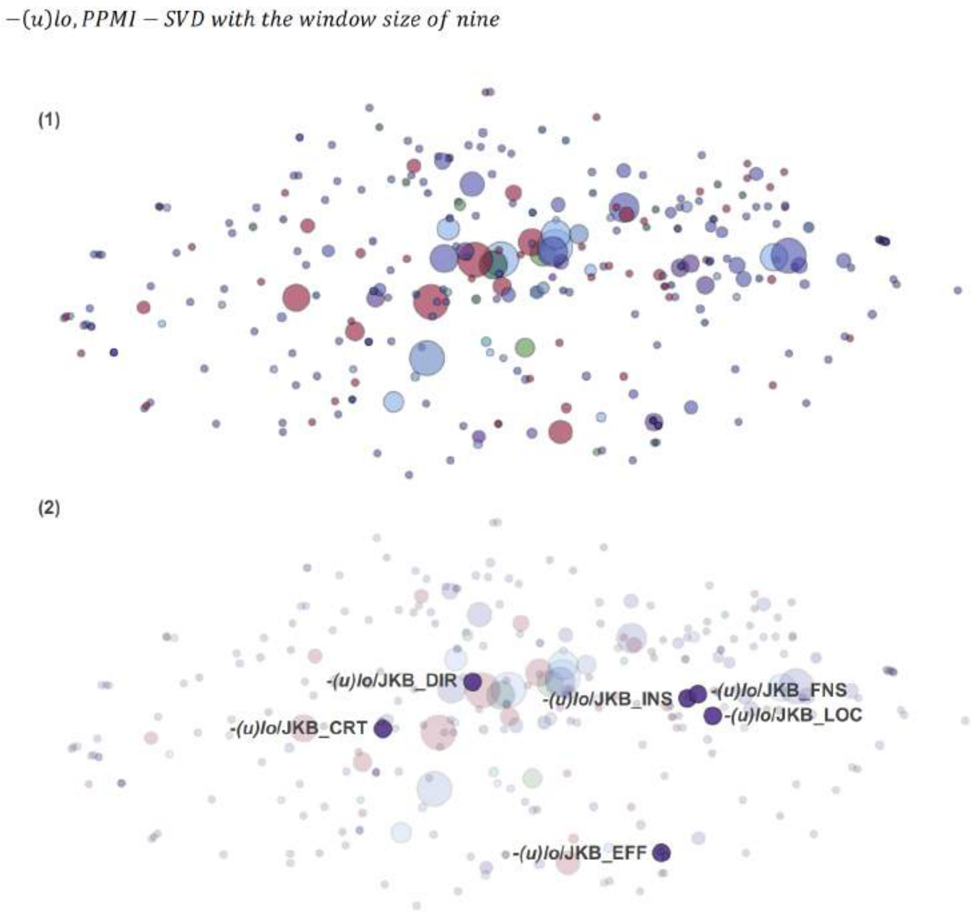

Figure 5.23: Distributional semantic map for *-(u)lo* (PPMI-SVD; window size of nine)

As shown in Figure 5.24, when *-(u)lo* was used as FNS, the most frequently used word was *tut-'listen'/VV* (13 instances). For DIR, *eti-'where'/MAJ* (96 instances) and *nao-'come out'/VV* (59 instances) were included in the list of co-occurring words, both of which are related to the target function semantically. In the case of INS, *mal-'word'/NNG* (257 instances) and *tulese-'pick up'/VV* (32 instances) occurred as its co-occurring words. For CRT, the verb *mantul-'make'/VV* (54 instances) related to criterion, was included in the list of co-occurring words, and *sik-'ways'/NNB* (95 instances) appeared frequently also.

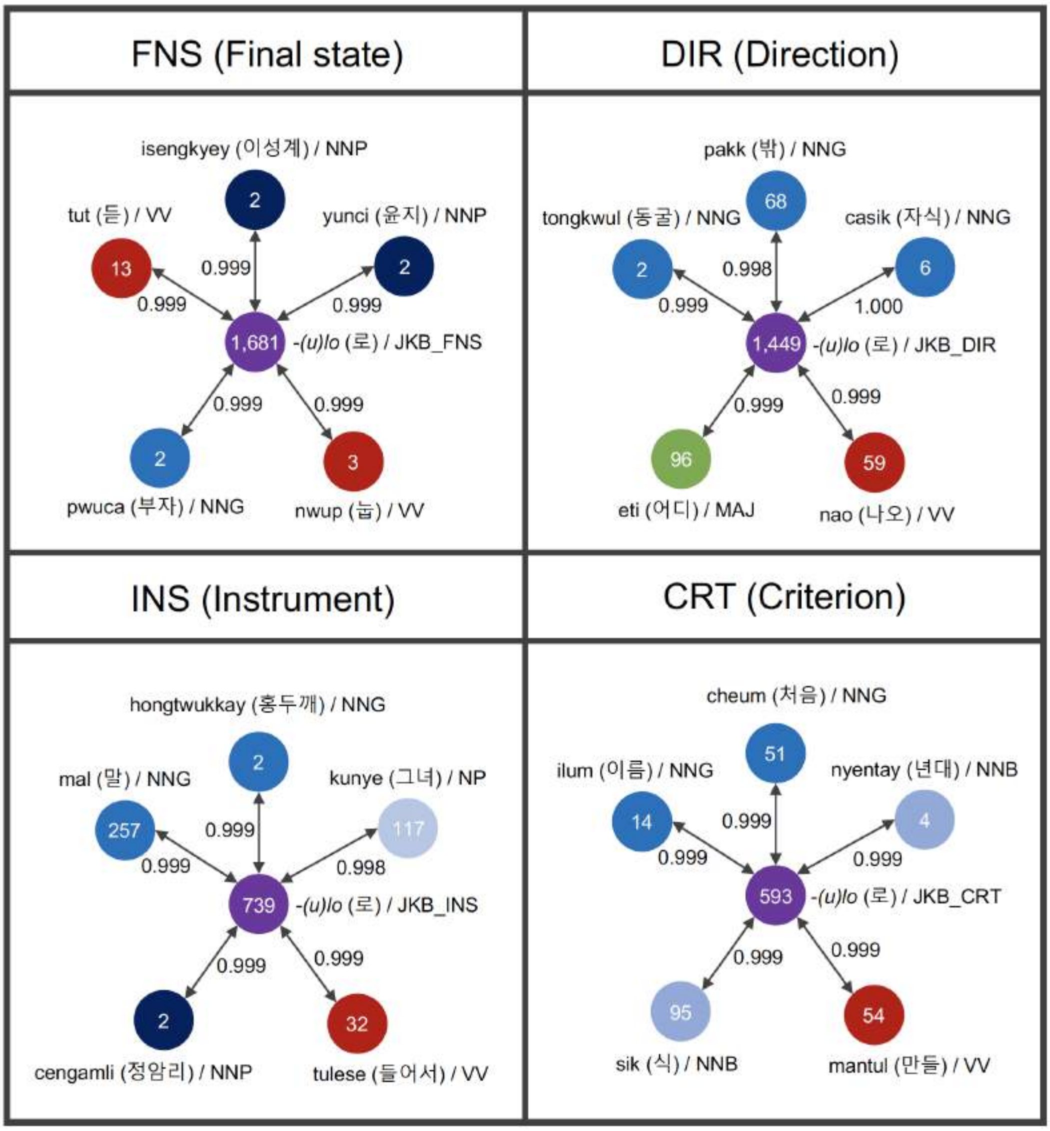

Figure 5.24: By-function co-occurring words for *-(u)lo*: FNS, EFF, INS, and CRT

Note. Abbreviation: JKB = adverbial postposition; NNB = bound noun; NNG = common noun; NNP = proper noun; NP = pronoun; VV = verb

Continuing to Figure 5.25, when *-(u)lo* was used as LOC, nouns indicating locations such as *elkwul-'face'/NNG* (64 instances) and *sewul-'seoul'/NNP* (56 instances) were included in the list of co-occurring words. In the case of

EFF, the verb *pwulu-'call'/VV* (32 instances) related to reason, was included in the list of co-occurring words.

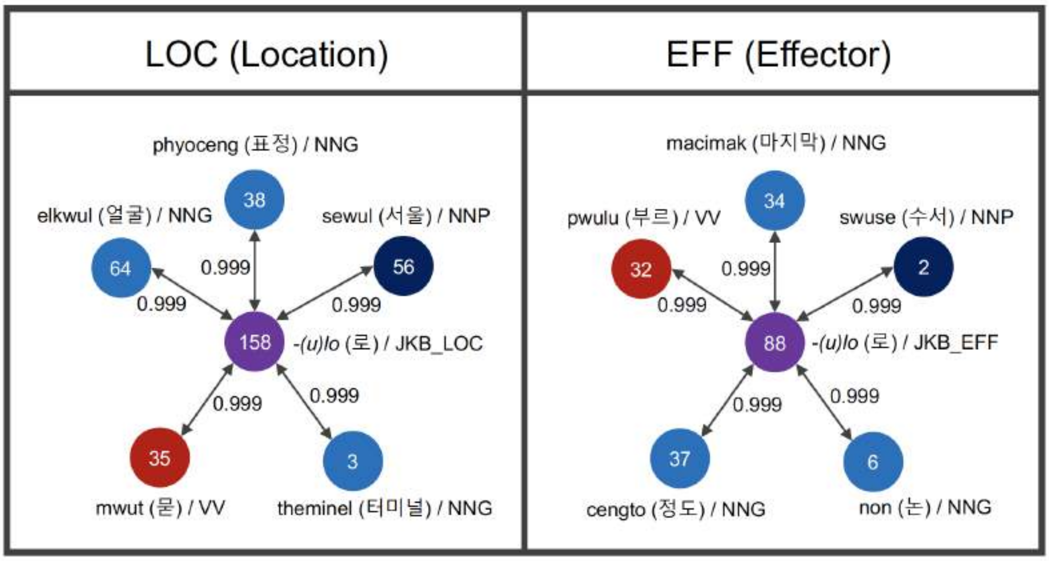

Figure 5.25: By-function co-occurring words for *-(u)lo*: LOC and EFF

Note. Abbreviation: JKB = adverbial postposition; NNG = common noun; NNP = proper noun; VV = verb

5.3.3 Interim summary of visualization results

The visualization system responded interactively to the options (e.g., model types, postposition types, window sizes) and showed the corresponding results. I expected that the visualization could answer Hypothesis 3 (see Section 5.1)—the relationships and their co-occurring words should vary depending on the environments of word-level embedding. To my surprise, the PPMI-SVD model showed that words with high frequency were located in the center of each cluster without much change, and the SGNS model showed that the words seem widely distributed without little influence of word frequency.

However, cluster analysis showed that the two models did not differ significantly from each other because the density clusters were gathered into one or two clusters in the end. Regarding the co-occurring words in each function of the postpositions, the outcomes can be divided into the following types: (i) words with high similarity but low frequency of co-occurrence, and (ii) words with high similarity and also a high frequency of co-occurrence. The first group were words that appeared only when they were used as a particular function. Conversely, the second group were words that had a strong connection in language use regardless of which functions a postposition was used.

5.4 Discussion of the Chapter

In this chapter, I reported model performance of the count-based model (PPMI-SVD) and the probability-based model (SGNS) in the classification of the functions of the postpositions *-ey*, *-eyse*, and *-(u)lo*. There were three major findings.

First, the fewer dedicated functions the postposition has, the higher the classification accuracy was. Considering that the three postpositions have different numbers of functions (e.g., two for *-eyse*, six for *-(u)lo*, and eight for *-ey*), there was an inverse relation between the classification accuracy and the number of functions in a postposition, as in Hypothesis 1 (see Section 5.1).

Second, contrary to Hypothesis 2 (see Section 5.1), the PPMI-SVD model obtained high classification accuracy at a larger window size. The SGNS model showed low classification accuracy, regardless of window size in the case of *-ey* and *-(u)lo* and high classification accuracy at a larger window size in *-eyse*. Considering that smaller context windows work better for syntactic tasks and larger context windows contribute more to semantic tasks (e.g., Jurafsky and Martin, 2019, Levy et al., 1999), our model may have performed more semantically than structurally.

Third, the cluster was not changed much by the environments of word-level embedding. In the PPMI-SVD model, the words used most frequently in the corpus were located in the center of the cluster; in the SGNS model, in contrast, the words were distributed rather evenly, regardless of word frequency. This was because the PPMI-SVD model was based on token frequency and the SGNS model was based on type frequency. However, cluster analysis showed that there was no clear difference between the two models because all of the density clusters for each distributional semantic map were gathered as one or two groups in the end. In addition, I found that there were two types of word group. First, words that appeared only when they were used as a particular function. Second, words that had a strong connection in language use regardless of which functions a postposition was used. However, they did not change much depending on the environments of word-level embedding (2 models * 3 postpositions * 10 window sizes), which is not consistent with Hypothesis 3 (see Section 5.1).

Despite these findings, the two models that I tested in this chapter have serious limitations. The model performance is unsatisfactory considering previous studies on the classification of the postpositions on which I focused (Bae et al., 2015, Kim and Ock, 2016, Shin et al., 2005). They reported a level of accuracy ranging from 0.882 (Kang and Park, 2003) to 0.623 (Bae et al., 2014). In contrast, the average level for my models was 0.550. Furthermore, the model appears to perform well only when the target functions occur very frequently in the data, which is not how I aimed to deal with polysemy resolution. It is due to the technical nature of word-level embedding, which distinguishes words occurring in the entire corpus using only the morphological information and the window size, and which uses words without considering their possible different effects on determining the meaning of a particular postposition. This is because the traditional word-level embedding models are *static*—a single vector is assigned to each word (Ethayarajh, 2019, Liu et al., 2019a).

To overcome these problems, I employed Bidirectional Encoder Representations from Transformer (BERT) (Devlin et al., 2018) for the classification of the functions of the postpositions. BERT produces contextual embeddings, and this characteristic may help us to create a better classification system for postpositions. A recent trend to handle this task is called *contextualized* word embedding, which converts all words into each vector by considering the context (e.g., position, a form of the word) in which they appear. Various models have been proposed such as Embeddings from Language Models (ELMo; Peters et al., 2018) and Generative Pre-Training (GPT; Radford et al., 2018), but BERT shows the best performance out of all of the models introduced so far. Therefore, I applied BERT to my classification model in order to improve model performance. This is outlined in detail in Chapters 6 and 7.

5.5 Summary of the Chapter

In this chapter, I provided the findings of the classification models and visual inspections, starting from my hypothesis on the research questions. First, Section 5.2.1 showed that the classification accuracy is inversely proportionate to the number of functions of a postposition. Second, Section 5.2.2 showed that high classification accuracy was not obtained in smaller window sizes, but rather in larger window sizes. This section also showed that the overall classification accuracy is similar to the curve of the functions that accounts for a large portion of the total corpus size. Finally, Section 5.3 showed that the clusters and the co-occurring words of each function of the postpositions was not very different based on the environments of word-level embedding (2 models * 3 postpositions * 10 window sizes).

However, despite these findings, two limitations have been found because of the technical nature of word-level embedding, that a single vector is assigned to each word. One was that the model performance was lower than the accuracy reported by the other studies, and the other was that it was high only if the target functions occupy a large portion of each postposition in the corpus data. To address these limitations, I decided to use BERT, which is a contextualized word embedding model with the best performance. The technical description and application of BERT will be discussed in the next chapter.

Chapter 6

BERT for polysemy resolution

The outcomes of the two word-level embedding models (PPMI-SVD and SGNS) revealed issues with model performance—accuracy seemed to be affected by the corpus size of each function of the postpositions. This is because word-level embedding converts a word into a single vector based on its morphological form. To overcome this limitation, I employ Bidirectional Encoder Representations from Transformer (BERT) (Devlin et al., 2018), a state-of-the-art technique, for the same task but is sensitive to the context in which words appear (e.g., Ethayarajh, 2019, Liu et al., 2019a). This chapter reviews BERT as a classification model and provides methodological details on how it handles the polysemy of the three adverbial postpositions in Korean.

6.1 How BERT was born

BERT was developed as a response to the downsides of previous models for NLP tasks. Recurrent Neural Network (RNN) models (e.g., Mikolov et al., 2010), for example, utilize information about prior cells in the neural network in a circular manner, both by updating the current hidden state based on the information about the prior cell and by updating the posterior cell based on the information about the current hidden state. This process is known as one strength in addressing context information that indicates the correct meaning by extracting hidden state from the sequential combination of words (e.g., Mikolov et al., 2010, 2011, Sundermeyer et al., 2013).

Suppose the following sentence involving the postposition *-(u)lo* with a function of DIR (Direction) as in (1).

(1) pang_07/NNG -(u)lo/JKB ka/VV n-ta/EF ./SF

pang-ulo ka-n-ta.
room-DIR go-PRS-DECL

'(I am) going to the room.'

An RNN model (Figure 6.1) uses the information about the *pang_07/NNG* (prior cell) to update information about *-(u)lo/JKB* (current hidden state) and *ka/VV* (posterior cell) with *-(u)lo/JKB* (current hidden state). Two weights (W_{xh}, W_{hh}) and one bias value (b) are used to calculate information about the current hidden state. First of all, the weight W_{xh} and the input of each word are multiplied by each other. When the prior cell comes in, the weight W_{hh} and the prior cell are multiplied by each other. Next, these two values and b are added to each other and are represented as the current hidden state. Second, the hyperbolic tangent (an activation function) is used to calculate the output of the current hidden state. This function is advantageous for resolving the Vanishing Gradient Problem, namely the issue that the gradient disappears in the process of backpropagation (e.g., Bengio et al., 1994). The tangent function is used to calculate the output value of the current hidden state and then it is used to update information about the next hidden state. Finally, the output value from the last hidden state is converted into a context vector of the inputted context, and its probability is calculated by using the softmax function (see Chapter 3). W_{xh}, W_{hh}, and b are modified gradually, in a way that minimizes the difference (i.e., an error) between a prediction and an outcome by backpropagation.

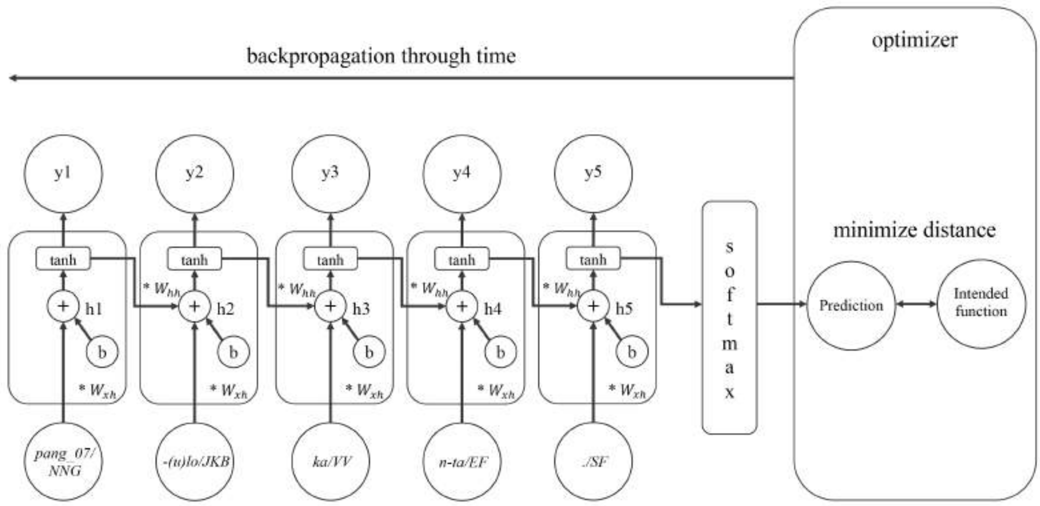

Figure 6.1: Workflow of the RNN model adapted from Heo (2018)

The RNN models, which consider information about word context, has an advantage in modeling with any size of context, such as long sentences, paragraphs, and even documents (Tang et al., 2015). This is unlike the traditional deep learning models that follow the pre-defined context size for the modeling (Chung et al., 2014). Due to this advantage, RNN models have shown better performance than other techniques such as convolutional neural networks (Collobert and Weston, 2008), recursive neural networks (Socher et al., 2011) for POS tagging (dos Santos and Zadrozny, 2014), multi-category text categorization (Chen et al., 2017), and sentiment analysis (Poria et al., 2017).

However, one major issue with the RNN model is that it has to represent all the information in a fixed-length context vector even if a word is not relevant to the target context. This can lead the RNN model to be incapable of processing sentences longer than those in the training corpus. Cho et al. (2014) revealed this aspect, by showing that the performance of the basic RNN model indeed decreases rapidly as the length of input sentences increases.

To address this issue, Bahdanau et al. (2014) proposed an attention mechanism that generates a dynamic context vector from all the hidden states. As stated above, the RNN model uses only the fixed-length output value of the last hidden state as a context vector for classification. In contrast, the attention model uses all the hidden state values and an attention weight of each hidden state when generating a context vector. This aspect leads to better performance in classification (e.g., Bahdanau et al., 2014, Parikh et al., 2016, Seo et al., 2016).

Suppose that we have the same example sentence (1) here. The workflow of extracting the context vector from it through the attention model is shown in Figure 6.2. The basic structure of the attention model is the same as the RNN model. However, instead of generating the context vector from the last hidden state, the attention model uses fully connected (FC) layers, which means that all former layers are connected with the next ones to calculate the score for each hidden state. These values are then converted to probability scores through the softmax function. The probability score of each word becomes an attention weight, indicating which words should be focused on generating the context vector. Finally, the dynamic context vector is obtained from the calculation that summed the values obtained by multiplying the hidden state of each word (y) and each attention weight (s).

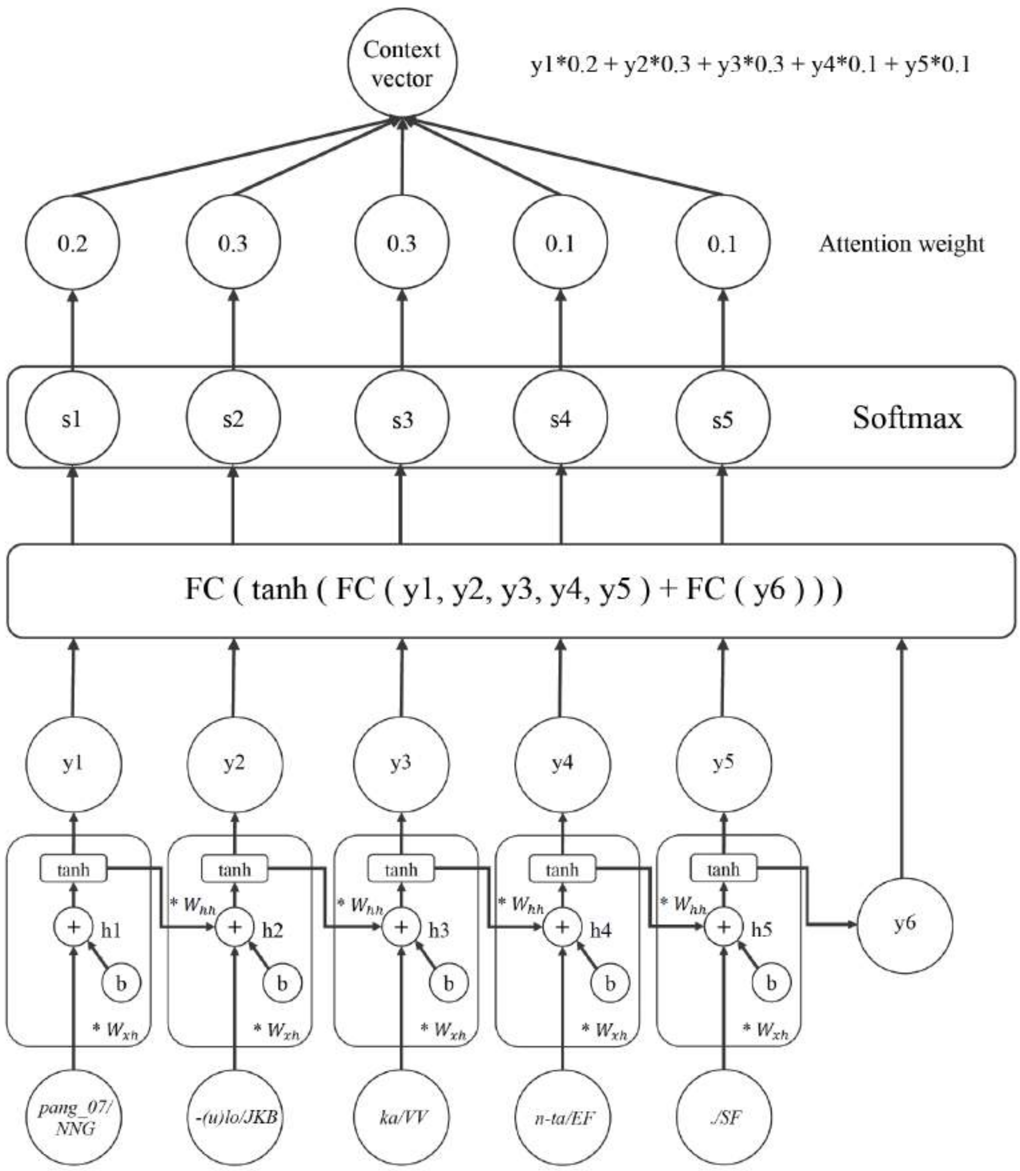

Figure 6.2: Workflow of the attention model adapted from Heo (2018)

The attention model uses information about all hidden states and focus on the core one among these through attention weight in order to generate a dynamic context vector. This characteristic allows the model to perform better than the RNN model for machine translation (Bahdanau et al., 2014), syntactical parsing (Vinyals et al., 2015), and sentiment analysis (Wang et al., 2016).

Despite its strength, the attention model has a weakness in terms of efficiency; it takes a huge amount of time to train the model. This is because it works on the basis of the RNN model, which proceeds sequentially, and so it must wait until the training of a previous phase is complete to pass to another. Parikh et al. (2016), for example, claimed that the performance of the attention model is problematic in two aspects: (i) the time complexity, that quantifies the amount of time taken by the attention model to generate context vector and (ii) the space complexity, that quantifies the amount of space or memory taken.

As a remedy for this issue, Vaswani et al. (2017) proposed a self-attention model (Transformer), eliminating RNN from the attention model and parallelizing the learning processes by computing the dot products of the query with all keys. Figure 6.3 illustrates the workflow of the self-attention model with the same example sentence (1).

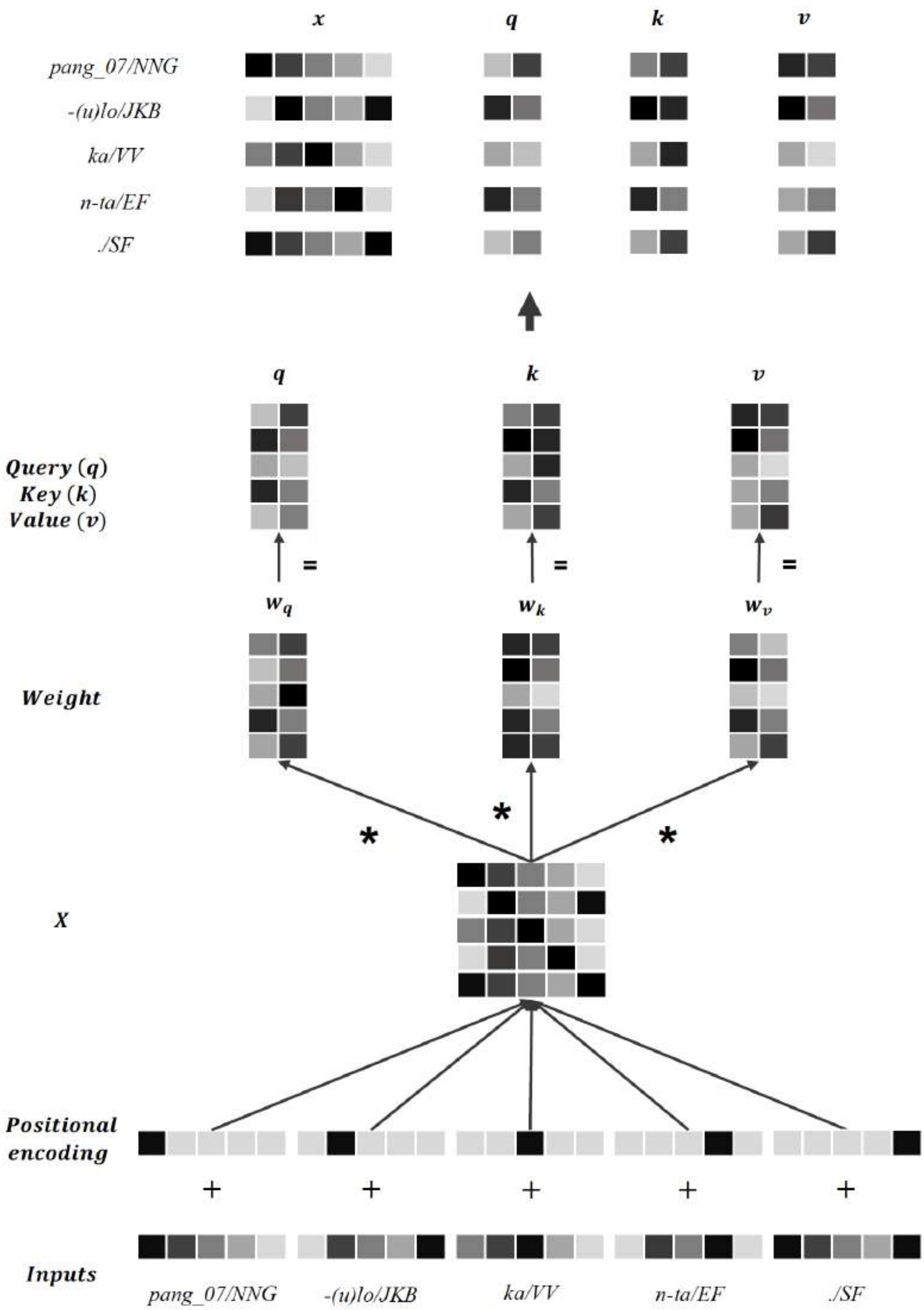

Figure 6.3: Workflow of the self-attention model

The steps of the self-attention model workflow are as follows: first, the vector of each word and the positional encoding value that indicates the position of the word in the sentence are merged into a single value. Second, one matrix is created on the basis of word inputs. Third, the weight matrices w_q, w_k, and w_v (optimized by a feed-forward process) are multiplied to generate a $query$ (the current input), a key (the other words in the current input to calculate the correlation with the $query$), and a value (the correlation value of the $query$ and the key). By using these three values, a self-attention model generates a vector that best represents the model, as shown in Figure 6.4.

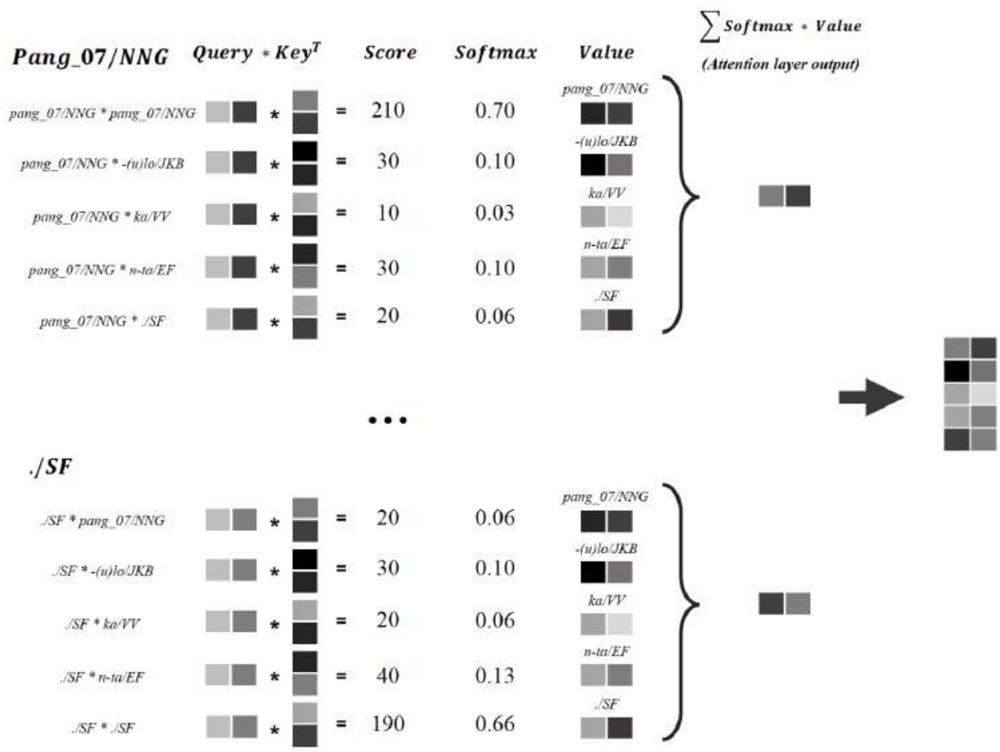

Figure 6.4: Calculation process of the self-attention model

The correlation between the $query$ and the $keys$ in the sentence is computed through their dot products. This score, called an attention score, represents the degree of relation between a word and the current input. In order to get the probability scores, the attention score is divided by the square-root of the dimension of the key to prevent the score from getting larger as the dimension of the key increases. Then each attention score is obtained through the softmax function. Each softmax probability is multiplied by the $values$. The calculated scores are then added to represent the attention layer output of one token itself in the current input. Finally, the attention layer outputs are merged as one matrix representing the current input sentence.

In order to represent the context vector more accurately, the multi-head self-attention model employs multiple models in the following way (Figure 6.5): first of all, the multi-head self-attention model generates a matrix (whose size is the same as the input matrix (a)) by merging all the matrices obtained by individual self-attention models, and then the generated matrix is multiplied by the weight (w_o) to generate matrix (b). After then, the prior input matrix (a) is added to the current matrix (b). This is called a residual connection that prevents the value of the positional encoding from being changed during the backpropagation. The vector of each word is extracted from the matrix (b) and updated in the fully connected layer. The updated word vectors are then merged to create a matrix (c). Finally, the matrix (b) is added to the matrix (c), following the same concept of the residual connection. The final matrix is normalized by using the mean and standard deviation of each vector. The matrix (c) represents the final output value of the multi-head self-attention model.

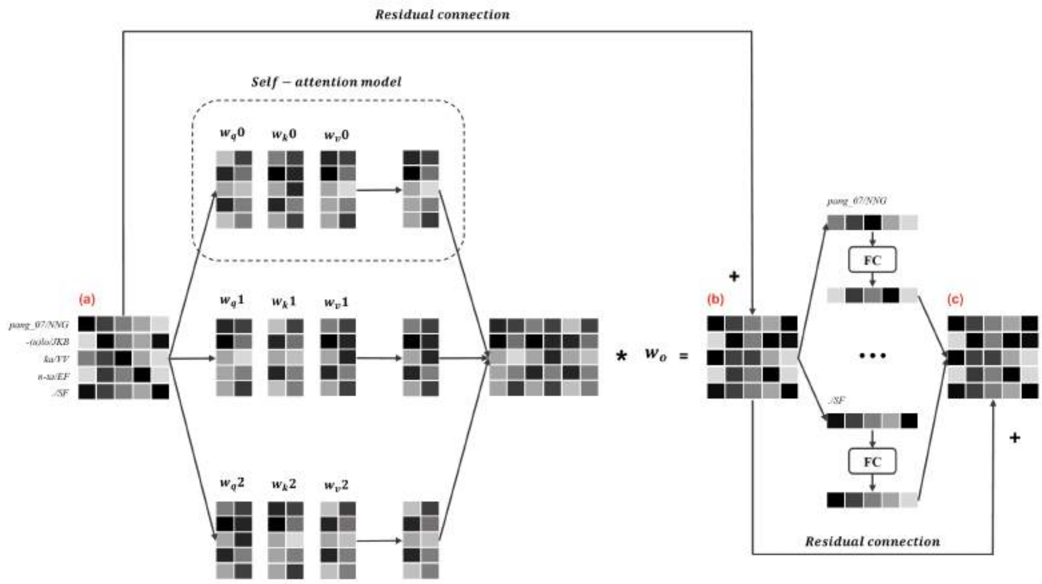

Figure 6.5: Workflow of the multi-head self-attention model (an encoder layer)

Transformer, which is a base model of BERT, consists of six multi-head self-attention models, each of which includes eight self-attention models. It generates a context vector the same way the multi-head self-attention model does, except that Transformer uses the output of the sixth from the last one as the final context vector.

6.2 Characteristics of BERT

BERT has a set of pre-trained language representations obtained by general-purpose, large-scale corpora. This applies increasingly to downstream NLP tasks such as language translation, sentence classification, and question answering (e.g., Devlin et al., 2018, Tang et al., 2019). The model architecture of BERT is based on Transformer, with some changes in the input unit and the specific tasks for model training. Just as the word-level embedding puts emphasis on vectorizing, so also does BERT, commonly through the WordPiece tokenization (Devlin et al., 2018).

6.2.1 WordPiece tokenization

The WordPiece tokenization, proposed by Schuster and Nakajima (2012), works on the basis of bi-gram pairs. Basically, the WordPiece tokenization has strength in addressing to build vocabulary including segmentation by morpheme. Suppose that the model encounters the word *walking*. Unless this word occurs frequently enough in the training corpus, the model may not learn how to deal with this word. However, the model may have words like *walked*, *walker*, and *walks* occurring a few times each. Without segmentation by morpheme, the model recognizes all these words as completely different ones. However, if these words are segmented as *walk*, *##ing*, *walk*, *##ed*, and so forth, the model is able to notice that all the original words have walk in common, which in turn occurs much more frequently while training (Schuster and Nakajima, 2012).

The workflow of the WordPiece tokenization is as follows (Sennrich et al., 2016): first, it splits a sentence into words (by using white-space-based tokenization) and these are turned into individual alphabets to be used as a word candidate list and set the number of iterations for learning. The word candidate list is used when the WordPiece tokenization combines each word into bi-gram pairs. After then, the WordPiece tokenization combines two individual alphabets as one word per learning. If the combined words have a high frequency of occurrence, segmentation by morpheme is removed from the word candidate list and the combined word is included instead. Finally, after the learning has progressed by the specified number of learning iteration, the remaining words in the word candidate list are stored in the vocabulary. This is then used for the BERT training (see more details in follows).

Whereas the WordPiece tokenization operates in the above way in English, it works differently in Korean. The algorithm extracts words by combining two syllables as one word per learning. First by splitting a sentence into words (by using white-space-based tokenization) and then the word into syllables. Second, by putting the bi-gram pairs of each syllable together. And third, by extracting word candidates which are frequently attested in the sentences into the vocabulary.

Suppose the following sentence involving the postposition *-(u)lo* with a function of FNS (Final state) as in (2).

(2) Yongho-lul pancang-ulo senchwul ha-ass-ko Chelswu-lul
Yongho-ACC class leader-FNS election do-PST-and Chelswu-ACC
pwu-pancang-ulo senchwul ha-ass-ta.
vice-class leader-FNS election do-PST-DECL
'(We) elected Yongho as the leader of the class and Chelswu as the vice- leader of the class.'

This sentence has six segments (*yongholul*, *pancangulo*, *senchwulhayssko*, *chelswulul*, *pwupancangulo*, *senchwulhayssta*). Each word segment then splits into syllables as shown in Figure 6.6. During this process, the character *##* is attached to the front of each syllable except the first syllable of the segment. This is designed to increase model performance by recognizing the same form of syllables differently.

Word segments	*Syllables*				
yongholul	yong	##ho	##lul		
pancangulo	pan	##cang	##u	##lo	
senchwulhaassko	sen	##chwul	##haass	##ko	
chelswulul	chel	##swu	##lul		
pwupancangulo	pwu	##pan	##cang	##u	##lo
senchwulhaassta	sen	##chwul	##haass	##ta	

Figure 6.6: Segmentation: Splitting word segments into syllables

Next, in order to get word candidates for the vocabulary to be used in BERT training, the WordPiece tokenization algorithm finds frequently attested bi-gram pairs, and utilizes these as a word candidate instead of as syllables, as shown in Figure 6.7. If the iteration number is 0, the WordPiece tokenization splits all word segments into syllables to be used as a word candidate list. When the iteration progresses, the bi-gram pairs that are frequently used in the previous iteration remain as one single combination in the word candidate list whereas the rest of the morphemes are removed. As shown in Figure 6.7 (b), if there are many bi-gram pairs which have the same frequency values, the WordPiece tokenization only considers the first bi-gram pair in the word candidate list. Finally, the final word candidate list is fed into the vocabulary for the BERT training.

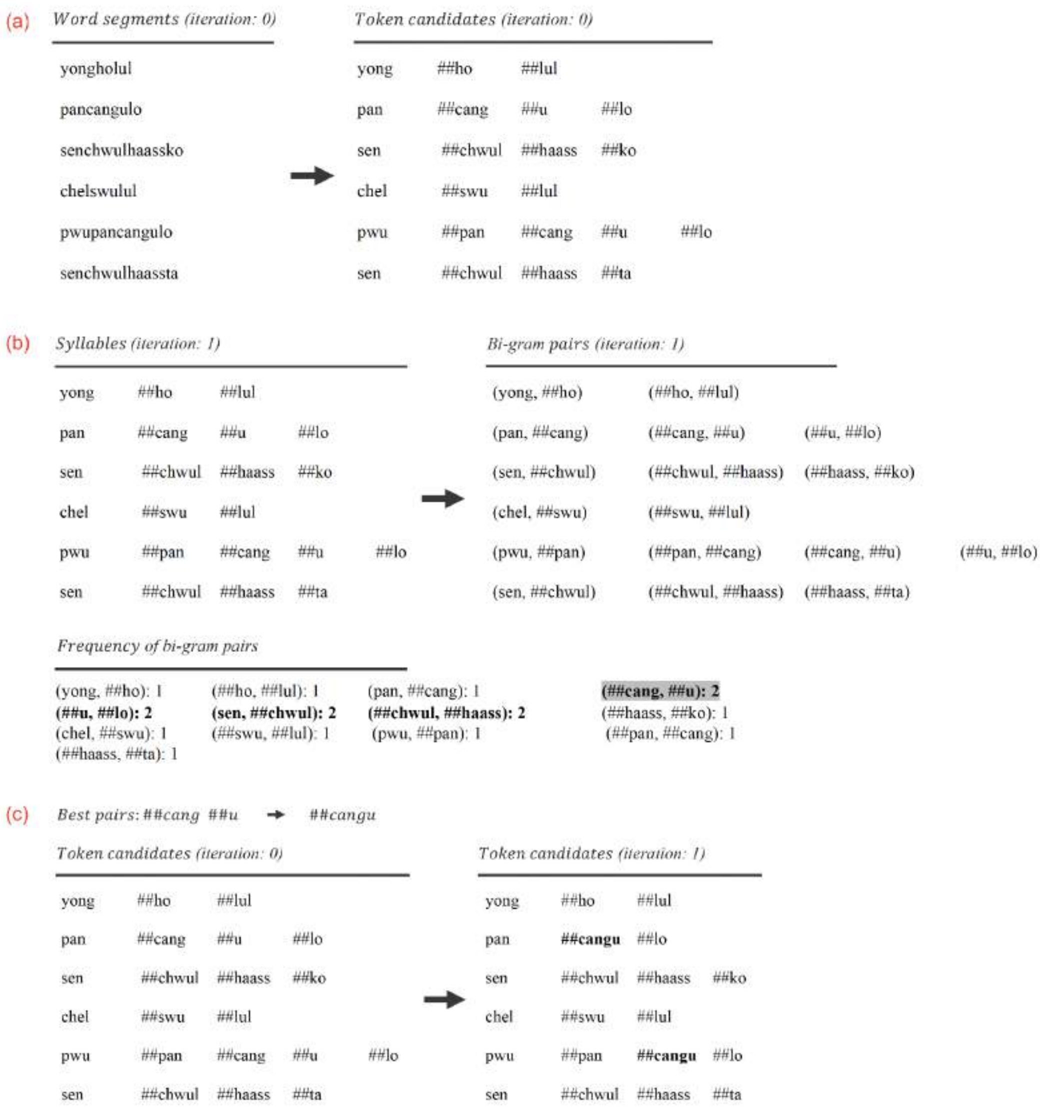

Figure 6.7: Workflow of the WordPiece tokenization

6.2.2 BERT

Transformer combines word information in a sentence with the positional information of each word and uses this combinatorial information as input for model training. However, BERT uses a combination of token embeddings, segment embeddings, and position embeddings as input (Devlin et al., 2018).

Suppose the sentence involving the postposition *-(u)lo* with a function of DIR (Direction) as in (3) and the sentence involving the postposition *-ey* with a function of LOC (Location) as in (4).

(3) pang_07/NNG -(u)lo/JKB ka/VV n-ta/EF ./SF

pang-ulo ka-n-ta.
room-DIR go-PRS-DECL

'(I am) going to the room.'

(4) kuliko/MAJ chimtay_02/NNG -ey/JKB nwup_01/VV nun-ta/EF ./SF

Kuliko chimtay-ey nwup-nun-ta.
And bed-LOC lie-PRS-DECL

'And (I) lay down on the bed.'

Figure 6.8 illustrates the three embedding types of BERT from the sentences ((3), (4)). For the input, it uses the two sentences as a single input. The sentences are split by marking [CLS] ('classification' indicating the start of a sentence) at the beginning of the first and [SEP] ('separation' indicating the end of a sentence) at the end of each sentence. Next, BERT makes tokens for input by using WordPiece tokenization, which extracts frequently attested bi-gram pairs, and utilizes these pairs as tokens. For the token embedding, each token extracted through the WordPiece tokenization is represented as an embedding value of each token. For the segment embedding, the tokens in the first sentence are expressed as A and the ones in the second sentence as B. For the position embedding, BERT indicates the position number of each token in the input.

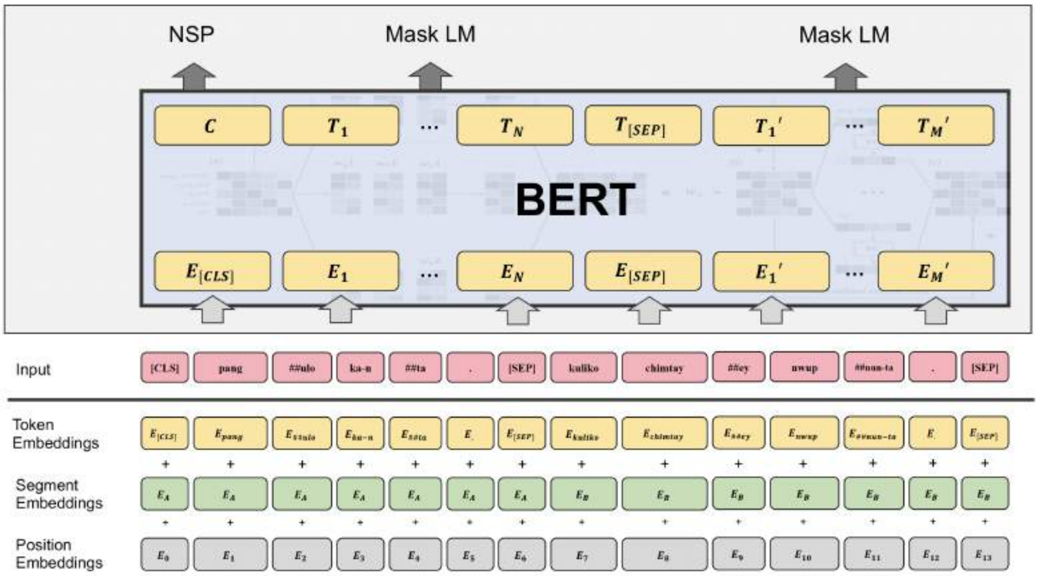

Figure 6.8: Three embedding types for BERT adapted from Devlin et al. (2018)

These three embeddings are used for model training, which is based on how the Transformer model is trained (see Section 6.1). One difference is that BERT employs not only Masked Language Model (MLM) but also Next Sentence Prediction (NSP) as the training method. As shown in Figure 6.9, in order to conduct the MLM training, sentences in the whole corpus are transformed into three different types: (i) 80% of the corpus include the [MASK] token which replaces only one word in the particular sentences (e.g., pang [Mask] *ka-n ##ta*.), (ii) 10% include a random word which replaces only one word in the particular sentences (e.g., *pang ##eyse ka-n ##ta*.), and (iii) the remaining 10% are unchanged. The aim of the MLM task is to predict the masked words correctly by using the given unmasked words.

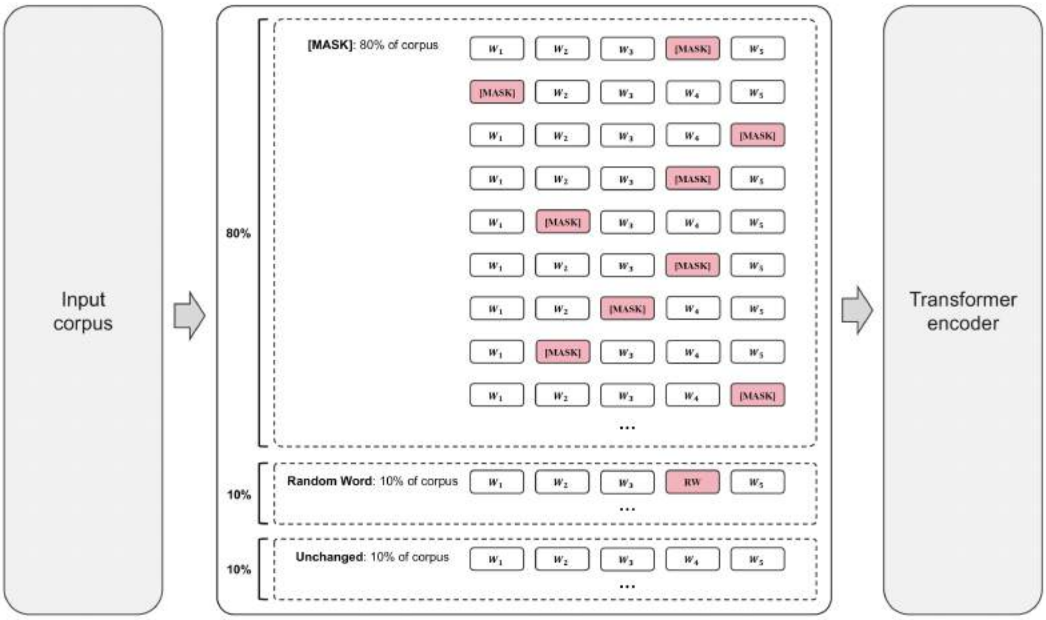

Figure 6.9: Workflow of the Masked Language Model (MLM)

The other training method is NSP, with the aim of predicting whether or not a sentence that follows is the correct one in the original document. This method assumes that an acontextual sentence will be disconnected from the sentence of interest. As shown in Figure 6.10, in order to conduct the NSP training, the second sentence of a sentence pair changes such that a half of the second becomes a random one (i.e., NotNext), and the other half is intact (i.e., IsNext). BERT then proceeds to the training by classifying whether or not the next sentence is an actual sentence.

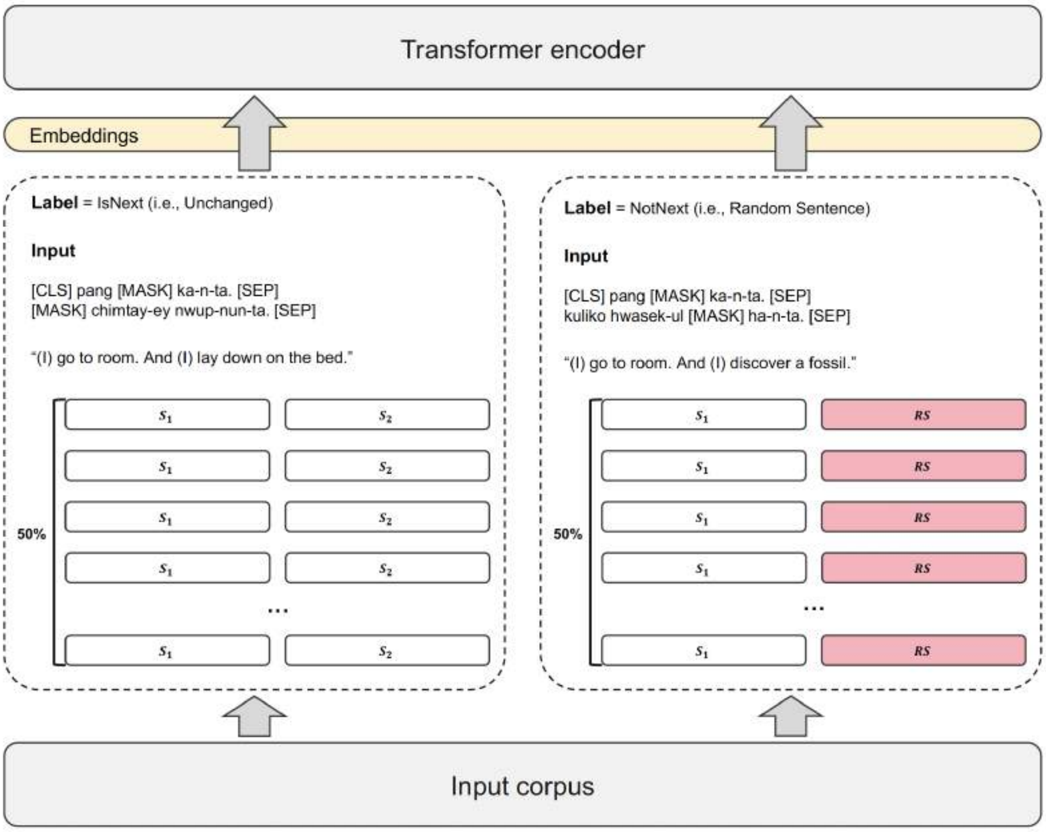

Figure 6.10: Workflow of the Next Sentence Prediction (NSP)

BERT conducts NSP for the input sentence pairs and MLM after dividing pairs of sentences into individual ones. The performance of BERT is improved by using both training methods that continuously optimize the loss between the predicted outcome and the actual one.

6.3 Effectiveness of BERT

The recent deep-learning, neural-network models such as Embeddings from Language Models (ELMo; Peters et al., 2018), Generative Pre-Training (GPT; Radford et al., 2018), and BERT (Devlin et al., 2018) have successfully created contextualized word embeddings, allowing the same word types to be represented differently in a given context (e.g., Liu et al., 2019a, Loureiro and Jorge, 2019, Wiedemann et al., 2019). Replacing the static embeddings produced by PPMI-SVD (Turney and Pantel, 2010) and SGNS (Mikolov et al., 2013a) with the contextualized representations has improved performance in NLP tasks significantly (e.g., Clark et al., 2019, Lin et al., 2019, Liu et al., 2019a, Loureiro and Jorge, 2019, Wiedemann et al., 2019).

Some studies compared the performance of various models on the basis of contextualized word embedding (e.g., Devlin et al., 2018, Sanh et al., 2019, Tang et al., 2019). For instance, Devlin et al. (2018) introduced BERT for the first time and reported the comparison of BERT, ELMo, and GPT. They found that the BERT outperformed the other models on eleven NLP tasks (GLUE score to 80.5% (7.7% point absolute improvement)) and thus, more effective. Moreover, Tang et al. (2019) conducted experiments on the General Language Understanding Evaluation (GLUE; Wang et al., 2018) benchmark, a collection of six Natural Language Understanding tasks that are classified into three categories: (i) Stanford Sentiment Treebank 2 (SST-2; Socher et al., 2013) that classifies a single sentence according to intended sentiment (positive or negative), (ii) Multi-genre Natural Language Inference (MNLI; Williams et al., 2017) that classifies a pair of sentences considering whether the next sentence is contextually correct or not, and (iii) Quora Question Pairs (QQP; Iyer et al., 2017) that generates an answer to a given question. They found that BERT showed an accuracy of 0.949 in SST-2, an accuracy of 0.893 in QQP, and an accuracy of 0.867 in MNLI, indicating that it shows the best performance out of all the models introduced so far. Inspired by these studies, employing BERT for the downstream NLP tasks became a recent trend in contextualized word embedding research.

6.4 Summary of the Chapter

The performance of the two word-level embedding models (PPMI-SVD and SGNS) showed an unsatisfactory level of performance in polysemy resolution. This is due to the technical nature of these models; they are static in that a single vector is assigned to each word (Ethayarajh, 2019, Liu et al., 2019a). As a remedy to this issue, I employ Bidirectional Encoder Representations from Transformer (BERT) (Devlin et al., 2018), which considers neighborhood information about a polysemous word on the basis of the context in which they appear. BERT was developed as a response to improving the downside of previous language models such as the recurrent neural network model, attention model, self-attention model, multi-head self-attention model, and Transformer.

There are many natural-network models such as ELMo (Peters et al., 2018), GPT (Radford et al., 2018), and BERT (Devlin et al., 2018). However, BERT showed the best performance out of all the models introduced so far (e.g., Clark et al., 2019, Lin et al., 2019, Liu et al., 2019a, Loureiro and Jorge, 2019, Wiedemann et al., 2019). Inspired by these results, I decided to use BERT for the classification of the functions of the postpositions in this dissertation.

Chapter 7

Methodological set-up: BERT

The previous chapter has shown that the performance of the two word-level embedding models (PPMI-SVD and SGNS) was modulated by the size of training corpora containing specific functions of the Korean adverbial postpositions (see Chapter5). In addition, previous models showed unsatisfactory classification accuracy compared to the previous studies. To handle these issues, I use BERT (Devlin et al., 2018) to classify the functions of these postpositions. Unlike word embedding models that assign a single vector to each word type, BERT considers not only word form but also its context information—word vectors that are sensitive to the context in which they appear (e.g., Ethayarajh, 2019, Liu et al., 2019a).

This chapter introduces methodological details of BERT, with the three specific research questions as follows:

- Research question 1: How does the number of functions involving a postposition affect the model performance of BERT?
- Research question 2: How the asymmetric proportions of the functions in each postposition affect the model performance?

- Research question 3: How does the BERT model classify sentences for each postposition based on function as the epoch proceeds?

7.1 Corpus

I use the same hand-coded corpus that was used for the word-level embedding models (Section 4.1.4), with some changes in the data considering how BERT works. First, BERT uses raw sentences to indicate the beginning and end of a sentence with [CLS] ('classification'; indicating the start of a sentence) before a sentence and [SEP] ('separation'; indicating the end of a sentence) after a sentence. Second, BERT expresses the function of the postpositions used in the sentence in a separate column, which is different from word-level embedding models that used lemmatized and POS-tagged sentences for training. Figure 7.1 illustrates the format of the data frame used for the BERT training and testing.

Index	Label	Sentence
1,862	1	[CLS] 한참 만에 오반장이 침묵을 깼다. [SEP]
1,863	1	[CLS] 정말 오랫만에 먹어보는 고기였다. [SEP]
1,864	1	[CLS] 옛날 구한말에 유명한 얘기가 있었죠? [SEP]
1,865	1	[CLS] 한밤중에 신나게 한바탕했지요. [SEP]
1,866	1	[CLS] 그런데 몇 시에 왔어? [SEP]
1,867	1	[CLS] 겨울에 꽃이라니요. [SEP]
1,868	1	[CLS] 아침에 엄마한테 돈을 달랬어요. [SEP]
1,869	1	[CLS] 결혼은 반드시 적령기에 해야 한다. [SEP]
1,870	1	[CLS] 한 달에 얼마씩은 정확하게 들어오니까. [SEP]
1,871	1	[CLS] 그럼 일 주일 후에 뵙겠습니다. [SEP]

Figure 7.1: Example sentences used in the BERT training (-*ey*, CRT)

The rows of the data frame are composed of the total number of the sentence (4,715 sentences for -*ey*, 4,853 sentences for -*eyse*, and 4,708 sentences for -*(u)lo*). The columns are the index, label (-*ey*: 8 labels, -*eyse*: 2 labels, -*(u)lo*: 6 labels), and sentence. To use this data in the BERT training, the data for training and testing should be independent. I thus split the corpus into two sub-sets, one with 90 percent of the corpus for the training and the remaining 10 percent for the testing.

7.2 Model training

For sentence-level embedding for BERT to recognize the polysemy involving Korean adverbial postpositions in each epoch (i.e., step), I devised a BERT model through Python programming, by adapting functions provided by *keras* (Chollet, 2015), *pytorch* (Paszke et al., 2019), *scikit-learn* (Pedregosa et al., 2012), *tensorflow* (Abadi et al., 2016), and *transformers* (Wolf et al., 2019). For the training, I used GPUs and TPUs provided by Google Collab for the environment with a view for faster processing, because BERT consumes a considerable amount of memory (Jeon et al., 2019). To avoid excessive memory consumption, I also used an iterator through the function *DataLoader* from *pytorch* (Paszke et al., 2019) with the *batch* size of 32 in random sampling of the data per epoch (i.e., step). I then employed a pre-trained language model in order to obtain high accuracy of outcomes. For this, I used a Korean BERT (KoBERT), which was developed by Jeon et al. (2019).

7.2.1 KoBERT: pre-trained BERT model for Korean

KoBERT is a BERT model pre-trained with a corpus of 5 million sentences extracted from Korean Wikipedia. It consists of 768 hidden units, 12 attention heads, and the same 12 encoder Layers as the *BERT base cased*, which was previously distributed by Google. Jeon et al. (2019) explained that the existing *BERT base multilingual cased* showed unsatisfactory performance for Korean, so they conducted this work to release the Korean version of the BERT model. To evaluate the performance, they performed an evaluation that classifies movie review sentences as positive or negative using the existing *BERT base multilingual cased*, *KoGPT2*, and KoBERT. The results showed that *BERT base multilingual cased* showed an accuracy of 0.875, *KoGPT2* showed an accuracy of 0.899, and KoBERT showed the highest performance among the three with an accuracy of 0.901. Based on this result, I used KoBERT as pre-trained model for the BERT fine-tuning.

7.2.2 BERT fine-tuning by using *BertForSequenceClassification*

In order to start the training, two steps were necessary before working on the main training algorithm (i.e., input embedding and parameter setting). First, the input data are transformed into three embedding types: token, segment, and position (Devlin et al., 2018). Suppose the sentence involving the postposition -*(u)lo* with a function of DIR (Direction) as in (1).

(1) 방으로 간다.
pang-ulo ka-n-ta.
room-DIR go-PRS-DECL
'(I am) going to the room.'

Figure 7.2 illustrates the three embedding types of BERT from the sentence (1). At the first step of input embedding, I set the maximum number of tokens in one sentence to 128 for the optimal and efficient model training process. For the *token embedding*, *KoBertTokenizer* is used to tokenize the sentences in the data. For the *position embedding*, the tokens generated through the *KoBertTokenizer* are converted into numeric values that indicate a unique index of the tokens in the vocabulary of KoBERT. In this process, the maximum number of tokens in one sentence was designated as 128. For the *segment embedding*, the number of tokens of each sentence is converted into 128 numeric values using 0 (i.e., did not exist) and 1 (i.e., existed). If the number of tokens in the sentence were more than 128, the rest are automatically eliminated from the sentence in this process. In addition, to use BERT as classification model, I extracted the labels of the data separately. The information including three types of embeddings and labels extracted from the data is transformed as tensors, which reduces data size and thus, makes BERT-related data processing faster.

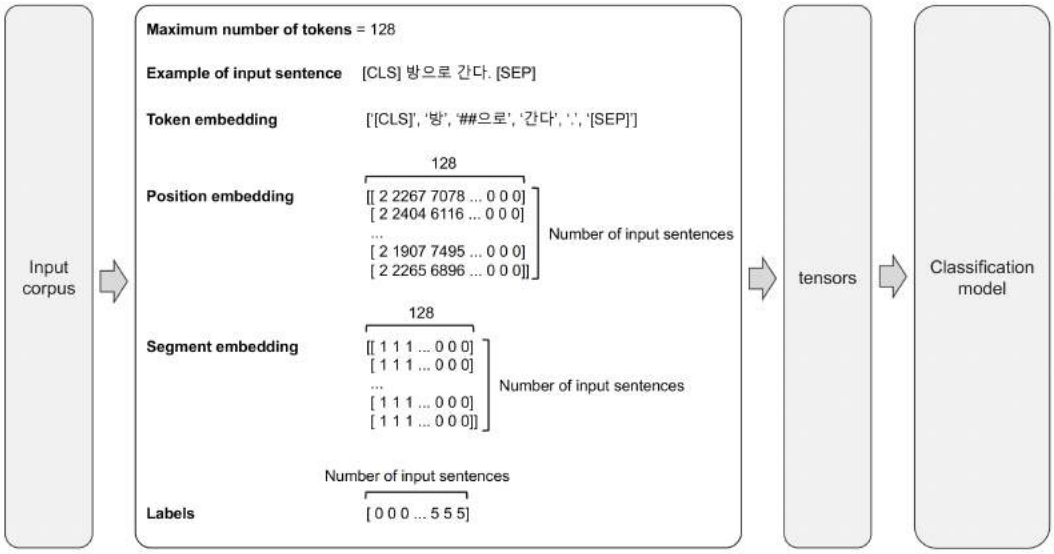

Figure 7.2: Input embeddings for the BERT classification model

The other required step for the BERT training was parameter setting. First, I set the value of the *seed* as 42, which is the initial value that enables the BERT model to start (Guggisberg, 2020). Second, I set the *optimizer*, which includes two parameters. One is *epsilon* (*eps*), which is a very small number, to prevent any division by zero in the calculation (Brownlee, 2020). I set the *epsilon* (*eps*) as 0.00000008 for the initial value. The other was the *learning rate* (*lr*), which is updated according to the outcomes of each epoch. I set the initial value of the *learning rate* (*lr*) as 0.00002. For the setting of these hyperparameters for the BERT model, I followed the recommendations of McCormick (2019).

After finishing all the necessary steps taken, the algorithm for the BERT training proceeded in the following ways (see Appendix A.3 for the entire algorithm)[1]. First of all, KoBERT (i.e., pre-trained model) is loaded through the function *BertForSequenceClassification* from *transformers* (Wolf et al., 2019) by each postposition. Second, I fine-tuned the pre-trained BERT model by using the training set. In this process, BERT reduces the loss of the model and updates the *learning rate*. Third, the testing set is loaded to evaluate whether the fine-tuned model recognizes the intended function of postpositions in each sentence. The accuracy rate of each function and the total accuracy rate was measured by comparing the intended function of attested postposition in each testing instance with the classified function through the fine-tuned BERT model. As a result of training in each epoch, I obtained two types of outcome, one composing a set of arrays (the number of sentences in each postposition; -*ey*: 4,715, -*eyse*: 4,853, and -*(u)lo*: 4,708) and the other composing the total of sets (the number of sentences in each postposition) of arrays (the number of functions in each postposition; -*ey*: 8, -*eyse*: 2, and -*(u)lo*: 6). In this dissertation, I used the first type of outcome. Finally, I employed the t-distributed Stochastic Neighbor Embedding (t-SNE; Maaten and Hinton, 2008) for dimension reduction of classification embeddings from the postposition by each epoch (see more details of t-SNE in Section 4.3.1).

The entire model training/testing is conducted 1,600 times (32 batches * 50 epochs), starting from the initial model with the zero value of gradients to an optimal model with updated values involving the model through forward- and back-propagation (cf., Xu et al., 2020). Loss values, which are the difference between outcomes from a BERT model in a particular epoch and real data, decreased as the values consisting of the model kept updating. I ob-

[1]The entire code for BERT training that I developed is available at: https://github.com/seongmin-mun/PhD_dissertation/tree/main/Python/BERT

tained the two-dimension distribution data in each epoch as the outcome of the BERT training. I used these in the visualization to see how BERT recognizes the polysemy involving Korean adverbial postpositions in each epoch.

7.3 Visualization: PostBERT

BERT is known to achieve a state-of-the-art accuracy when it is fine-tuned for supervised tasks (e.g., Dai and Le, 2015, Peters et al., 2018, Radford et al., 2018). However, it is not fully understood why this is so (e.g., Clark et al., 2019, Coenen et al., 2019). Clark et al. (2019) investigated this matter by analyzing the attention mechanisms of the pre-trained BERT model. In their study, they used attention maps to see how the BERT's attention heads exhibit patterns by such changes as delimiter tokens, positional offsets. They found that these attention heads attended to the direct objects of verbs, determiners of nouns, and objects of prepositions with remarkably high accuracy. Coenen et al. (2019) addressed this matter by visualizing the sentence-level representations of the BERT model. They investigated how BERT recognizes word meanings through the visualization of the sentence-level embeddings (Figure 7.3), and found that it could distinguish the different meanings of the word *'die'* in several contexts. This means that BERT recognized the exact intended meaning of *'die'* in each context.

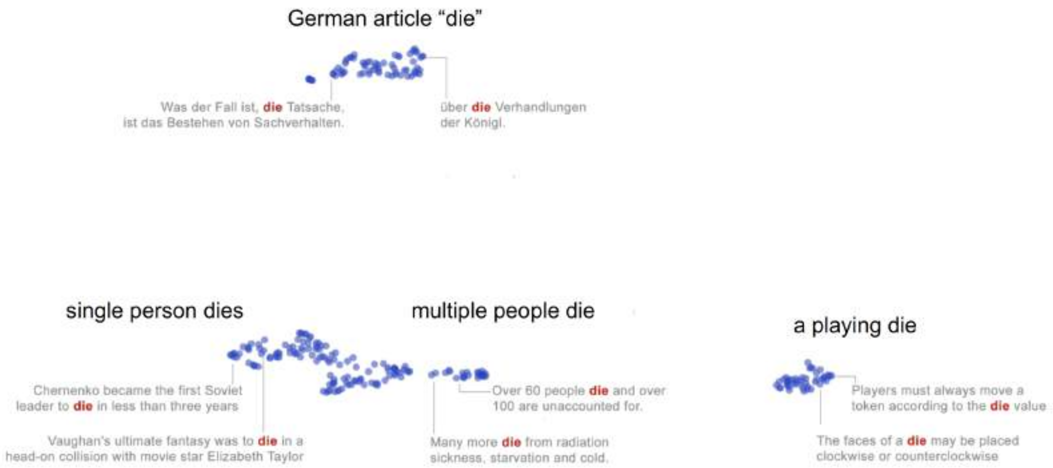

Figure 7.3: The visualization of the sentence-level embeddings for the word *'die'* in different contexts (adapted from Coenen et al. (2019))

7.3.1 Tasks and design objectives

With the advantages of visualization (see Section 4.3) and inspiration from the work by Coenen et al. (2019), I designed a BERT-based visualization system by specifying tasks and objectives as follows:

Task 1: Visually represent different clusters by the epochs (i.e., learning step) of the postposition types.

Design Objective: Design options for users to select each postposition on the left side of the visualization system. Also create a play button and slider at the bottom of the main visualization view in order to see the changes of clusters by epoch.

Task 2: Identify the details of each sentence in the cluster (e.g., the index number of a sentence, the intended function of postposition in the sentence, the raw sentence, the POS-tagged sentence).

Design Objective: Add pop-up views on the upper side of the main visualization view in order to provide the details of each sentence when the user moves the cursor over the circle (i.e., each sentence).

Task 3: Represent the various information about the model performance, such as overall accuracy, by-function accuracy, and loss rates.

Design Objective: Create two multi-line bar charts on the right side of visualization to see the change of the aforementioned model performance.

Task 4: Express the result of density clusters in each epoch, such as plots of density cluster and number of clusters.

Design Objective: Provide a bar chart at the bottom right side of the visualization system in order to present the number of clusters produced in each epoch. Also create a density cluster view at the bottom left of the system to present the clustering results according to the selected epoch.

7.3.2 System development

Considering the tasks and design objectives, I designed a visualization system (available at: PostBERT) to see how my BERT model classifies the functions of these postpositions in each epoch[2]. The visualization system was developed in Java, JavaScript, HTML, and CSS environments. The process of

[2]More details of PostBERT is available at: https://github.com/seongmin-mun/VisualSystem/tree/master/Major/PostBERT

development was divided into three parts: (i) data processing, (ii) front-end, and (iii) back-end. Each part of the process is similar to the previous visualization system (see Section 4.3.3), but the data processing part is different.

In data processing, I created four types of data using Java programming. The first data contains t-SNE outcomes that I obtained from the BERT classification. This data is connected with the distributional map for sentence-level embeddings to show the clusters between sentences. The second data includes raw sentences of the test set that represents each sentence in the distributional map. This data is merged with the first data to show details of each sentence such as an index of the selected sentence, the intended function used in the sentence, and the original sentence. The third data contains various information about the model performance: overall accuracy, by-function accuracy, and loss rates in the classification task by epoch. This data is used in the multi-line charts of the visualization system. The final data includes the results from the density-based cluster in order to show the number of clusters produced by the BERT model. This data is connected with the bar chart for density cluster. After the data processing, these JSON data were stored in the database.

For the front-end and back-end part of the visualization, I used several frameworks such as *Bootstrap*, *Media queries*, *D3.js*, and *jQuery* in order to make visualization system more interactivly (see more details of the front-end and back-end part in Section 4.3.3).

7.3.3 Interface of visualization system

For the interface of the visualization system, I propose three views to efficiently explore how BERT recognizes the word-level polysemy of the Korean adverbial postpositions. Each view presents the different outcomes related to BERT: sentence-level embedding, accuracy loss with respect to its performance, and results of density cluster (see Section 8.3).

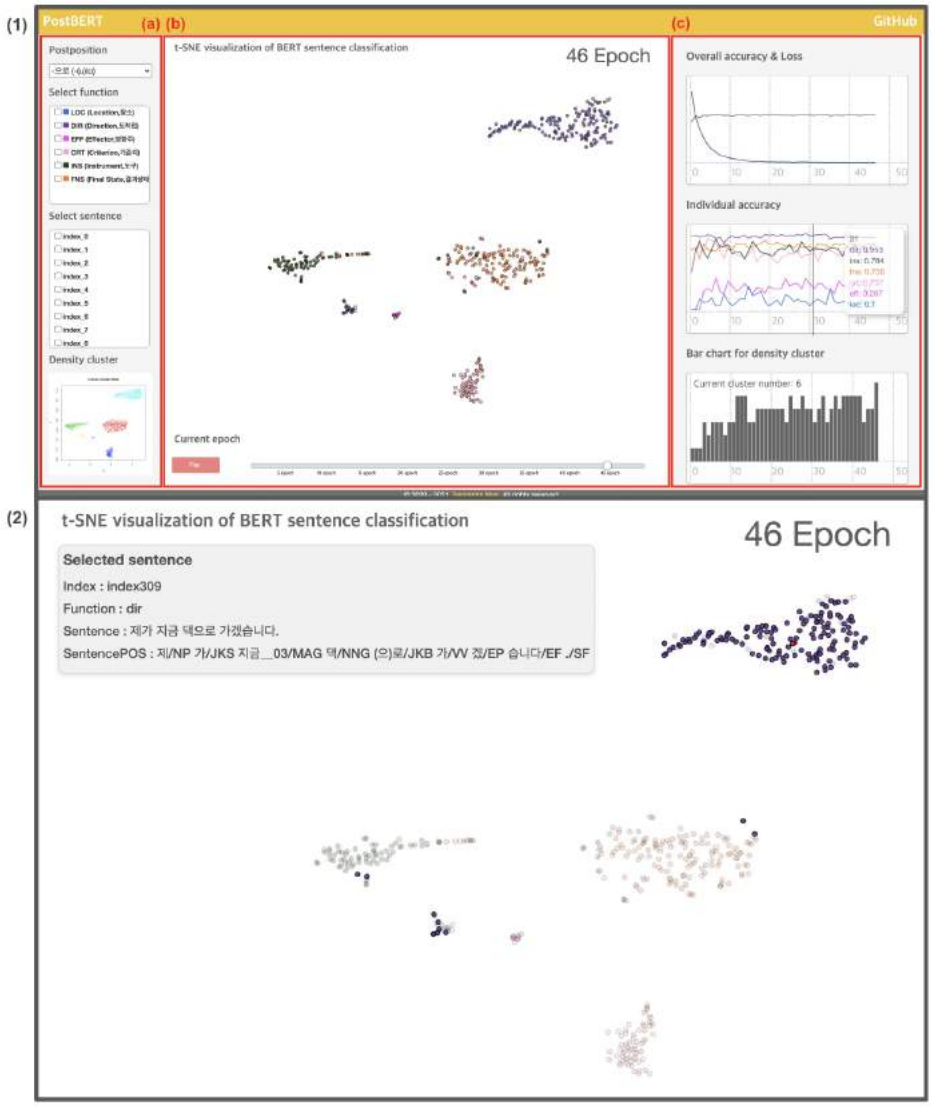

Figure 7.4: The visualization system: the overall interface (1) and the main view (2)

Figure 7.4 (1) shows the interface of the developed visualization system. (a) provides options to select the postpositions and checkboxes to highlight and tracking interesting sentences according to the index number or the function of these postpositions. (b) shows a distributional map of the sentence-level embeddings reduced to two dimensions using t-SNE. It also allows users to see the details of each sentence (represented as points) when the users hover their cursor over the circle. This allows the user to check the information such as an index number of the selected sentence, the intended function of the postpositions used in the sentence, and the raw sentence. At the bottom of (b), there is a play button to see the changes of the BERT outcome in each epoch. (c) shows two different types of BERT: (i) multi-line charts for its performance and (ii) a bar chart for density cluster. The multi-line charts on the right side of the visualization system (Figure 7.4 (c)) allow users to see the BERT performance such as overall accuracy, by-function accuracy, and loss in relation to the classification task by epoch (i.e., learning). This view also provides a hovering function to see the detailed score of each line in each epoch. The bar chart at the bottom of the right side of the visualization system (Figure 7.4 (c)) is to present the number of clusters indicating how BERT classified the sentences by their function in each epoch. This bar chart also provides a hovering function to see the actual number of clusters in each epoch.

7.4 Summary of the Chapter

The performance of the two word-level embedding models (PPMI-SVD and SGNS) resulted in an issue that the accuracy rate was modulated by the size of training corpora containing specific functions of the postpositions. As a remedy for this, I employed Bidirectional Encoder Representations from Transformer (BERT) (Devlin et al., 2018) to classify the functions of the postpositions.

Due to how the BERT model works, to develop a classification model, this study made an algorithm for training by using a hand-coded corpus in a slightly different manner. After training the model, I developed a visualization system to see how BERT classifies the function of the postpositions in each epoch and how the accuracy rate varies by each function by using 2-dimensional distribution data of the testing set.

The visualization system has several options to use and can identify each sentence-level embedding by using a reduced two-dimensional t-SNE plot. It also shows the user more information about the model performance (i.e., overall accuracy, by-function accuracy, and loss rates) and the results of density clustering.

In conclusion, I developed a classification model by using BERT. I then developed a visualization system to see how the BERT model classifies the functions of the postpositions in each epoch. The following chapter will explain the findings of the BERT classification model and BERT-based visualization system.

Chapter 8

Results: sentence-level embedding

This chapter reports the BERT model performance of classifying the functions of postpositions, starting from my hypotheses on the research questions, to by-postposition accuracy, and the results of the BERT-based visualization system.

- Research question 1: How does the number of functions involving a postposition affect the model performance of BERT?
- Research question 2: How the asymmetric proportions of the functions in each postposition affect the model performance?
- Research question 3: How does the BERT model classify sentences for each postposition based on function as the epoch proceeds?

8.1 Hypotheses

Hypotheses were made with respect to the three research questions about my classification models and the visualization results that showed how BERT model understand word-level polysemy of the three Korean adverbial postpositions (*-ey*, *-eyse*, and *-(u)lo*).

- Hypothesis 1: The accuracy of the classification should be inversely proportionate to the number of functions of a postposition.

The word-level embedding models that I investigated (see Chapter 5) showed that there was an inverse relationship between the classification accuracy and the number of functions. Given this finding, I predicted that the classification models will be influenced by the number of functions that a postposition has.

- Hypothesis 2: The accuracy of the classification should vary depending on the corpus size of each function.

The previous results of word-level embedding models showed that the classification accuracy is affected by the corpus size of the functions that account for a larger portion of the total corpus size. I thus predicted that this should also influence the accuracy of BERT.

- Hypothesis 3: The accuracy of the classification should be higher in larger epochs.

The previous studies that investigate various inquiries on language by using BERT recommended setting the epoch size small (e.g., Reimers and Gurevych, 2019, Reimers et al., 2019, Sun et al., 2019, Warstadt and Bowman, 2020). However, they did not explain clearly explain why the epoch should be set as a small size. Contrarily, I predicted that the classification accuracy will improve as the epoch increases, considering that epoch is the number of learning steps.

8.2 Model performance: Classification

8.2.1 Overall accuracy by the BERT model

Figure 8.1 shows the classification accuracy of the BERT model by epoch and by postposition.

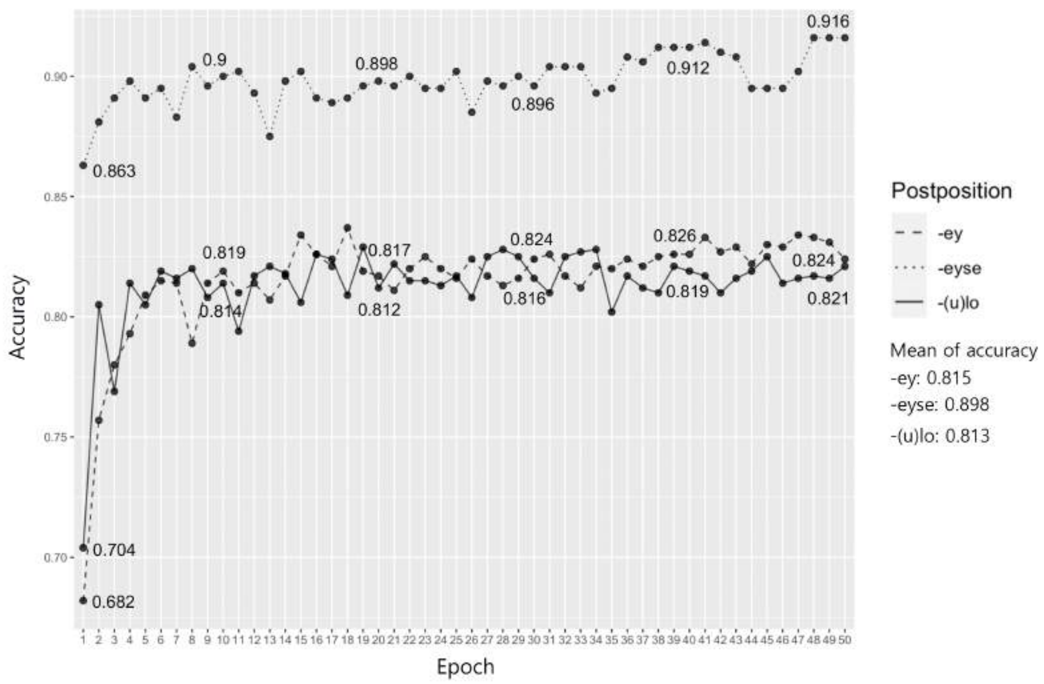

Figure 8.1: Classification accuracy by epoch and by postposition

The result showed that the BERT model performed better for *-eyse*, which has only two functions (SRC and LOC), than for the other two postpositions (*-ey* and *-(u)lo*). The average classification accuracy for *-ey*, *-eyse* and *-(u)lo* were around 0.815, 0.898 and 0.813, respectively. Statistical analysis of pair-wise comparisons (Table 8.1) further showed that the model performance in *-eyse* was significantly better than in the other two postpositions.

Table 8.1: Statistical comparison of each postposition: Two-sample *t*-test

Comparison	$\|t\|$	p
-ey vs. *-eyse*	22.588	$<.001$***
-ey vs. *-(u)lo*	0.533	.594
-eyse vs. *-(u)lo*	28.301	$<.001$***

Note. *** $<.001$

8.2.2 Overall accuracy by postpositions: *-ey*, *-eyse*, and *-(u)lo*

-ey

Figure 8.2 shows the classification accuracy in the BERT model for *-ey*. It was 0.682 in epoch one and increased to 0.824 in epoch 50, indicating that it increased as the epoch progressed. The highest accuracy was recorded in epoch 18 (0.837) and the lowest in epoch one (0.682).

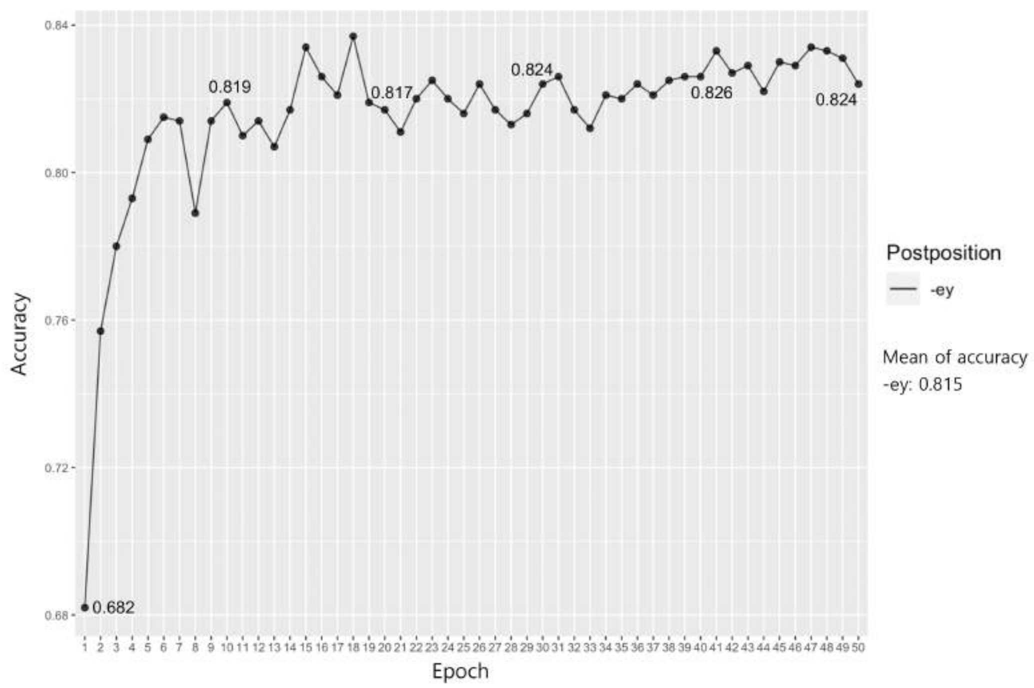

Figure 8.2: By-epoch accuracy for the BERT model: *-ey*

The performance of the BERT classification model for *-ey* varied depending on its function, as shown in Figure 8.3 and Table 8.2. The average classification accuracy was the highest in LOC (0.947) and the lowest in AGT (0.041); the other functions yielded accuracy ranging from 0.911 to 0.076. The results revealed three trends. First, the functions CRT and LOC maintained high accuracy epoch after epoch. Second, four functions, GOL, EFF, FNS, and THM, showed an increase in accuracy as the epoch proceeded. The degree of increase was the largest in FNS (71%), followed by THM (69%), then EFF (62%), and finally GOL (45%). Surprisingly, among these four functions, FNS showed an accuracy of 0 in epoch one but increased to 0.711 in epoch 27. Third, INS and AGT achieved low accuracy without improvement (around 0.2).

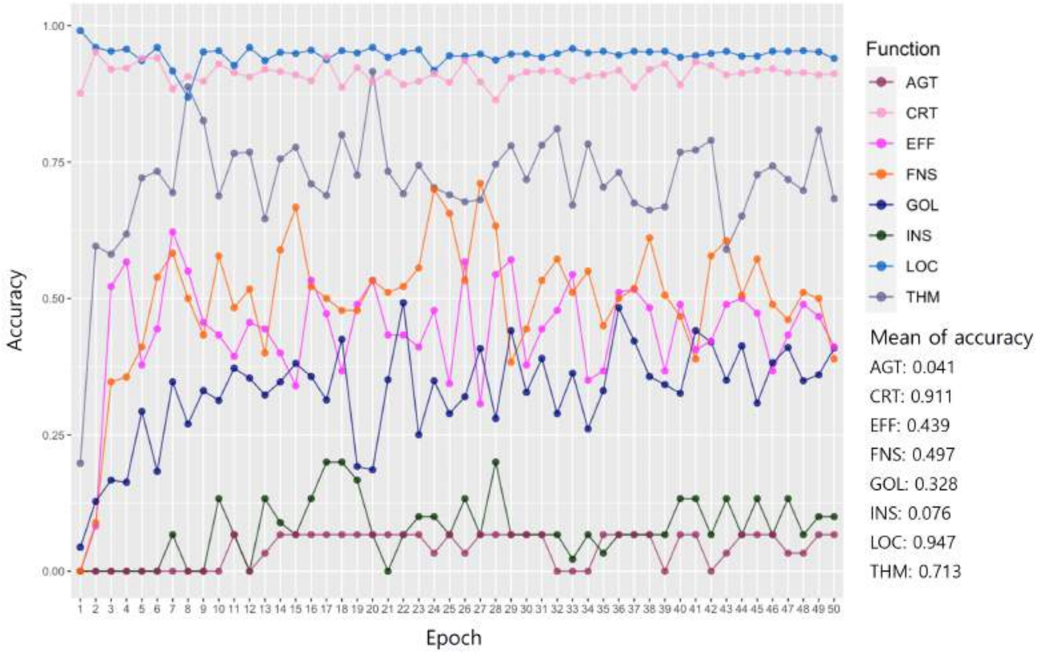

Figure 8.3: By-function accuracy curve for the BERT model: *-ey*

Note. Abbreviation: AGT = agent; CRT = criterion; EFF = effector; FNS = final state; GOL = goal; INS = instrument; LOC = location; THM = theme

Table 8.2: By-function accuracy for the BERT model: *-ey*

Epoch	Classification accuracy							
	AGT	*CRT*	*EFF*	*FNS*	*GOL*	*INS*	*LOC*	*THM*
1	0	0.876	0	0	0.044	0	0.911	0.198
10	0	0.930	0.433	0.578	0.313	0.133	0.954	0.688
20	0.067	0.897	0.533	0.533	0.186	0.067	0.960	0.916
30	0.067	0.915	0.378	0.444	0.328	0.067	0.948	0.718
40	0.067	0.892	0.489	0.467	0.326	0.133	0.942	0.768
50	0.067	0.912	0.411	0.389	0.409	0.1	0.940	0.683

-eyse

Figure 8.4 shows the classification accuracy in the BERT model for *-eyse*. It was 0.863 in epoch one and increased to 0.916 in epoch 50, indicating that it increased as the epoch progressed.

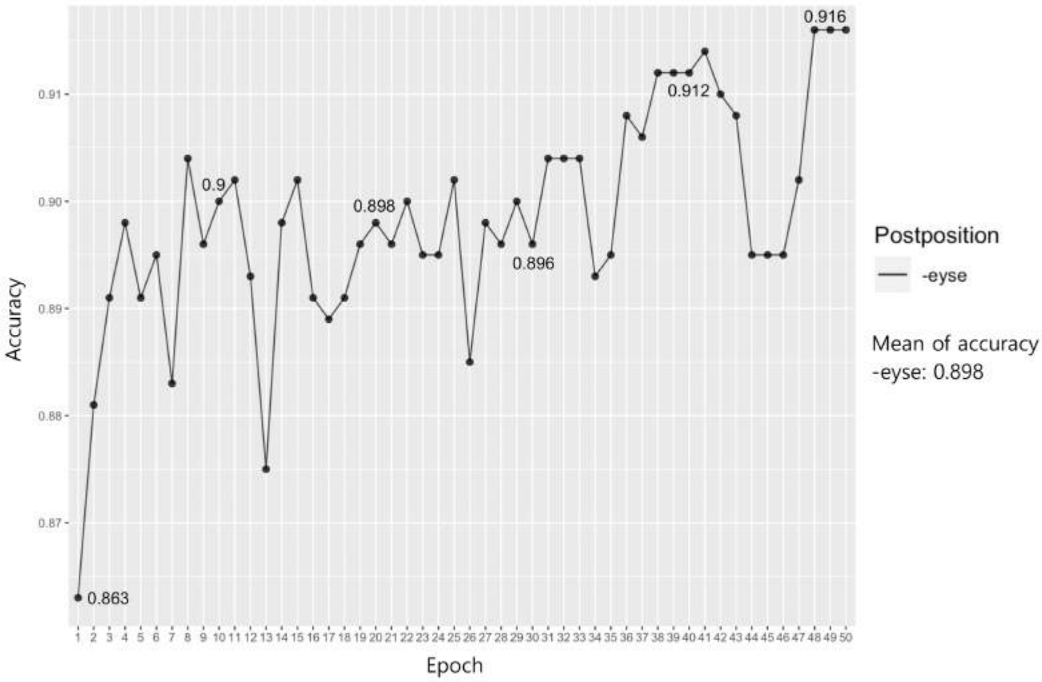

Figure 8.4: By-epoch accuracy for the BERT model: *-eyse*

The performance of the BERT classification model for *-eyse* varied depending on its function, as shown in Figure 8.5 and Table 8.3. The average classification accuracy was the highest in LOC (0.948) and the lowest in SRC (0.535). LOC maintained a high classification accuracy from epoch one. It showed an accuracy range from 0.916 to 0.98 without much change even when the epoch progressed. In contrast, the classification accuracy of SRC increased as the epoch proceeded. It showed a low classification accuracy of 0.174 in epoch one but increased to 0.725 in epoch 41.

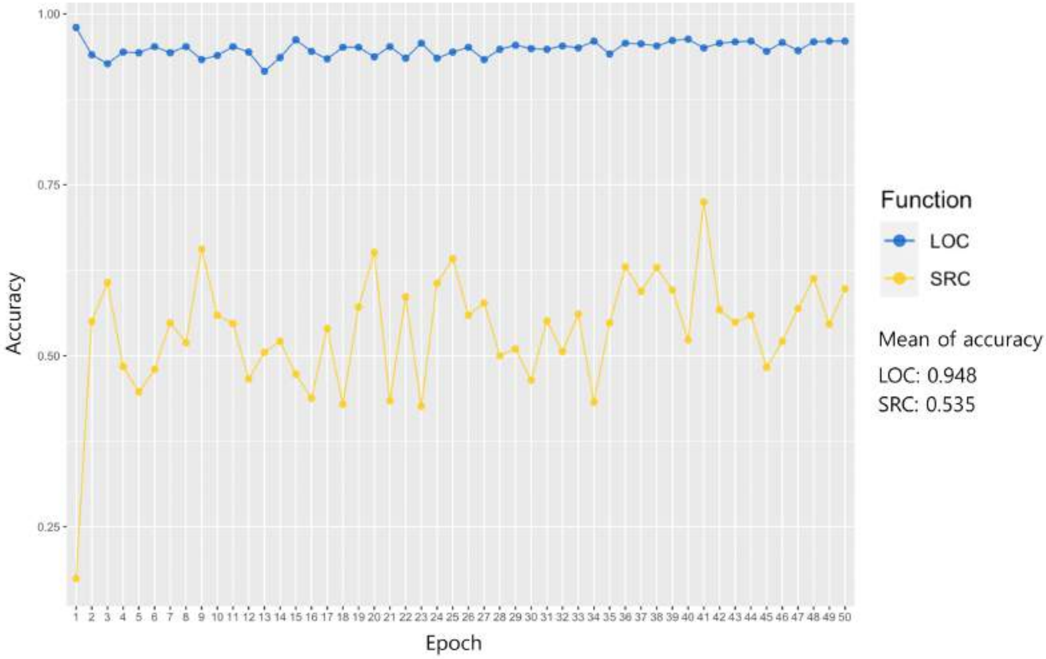

Figure 8.5: By-function accuracy curve for the BERT model: *-eyse*
Note. Abbreviation: LOC = location; SRC = source

Table 8.3: By-function accuracy for the BERT model: *-eyse*

Epoch	Classification accuracy	
	LOC	*SRC*
1	0.980	0.174
10	0.939	0.559
20	0.937	0.651
30	0.949	0.464
40	0.963	0.523
50	0.960	0.598

-(u)lo

Figure 8.6 shows the classification accuracy in the BERT model for *-(u)lo*. It was 0.704 in epoch one and increased to 0.821 in epoch 50, indicating that it increased as the epoch progressed. The accuracy was the highest in epoch 19 (0.829) and the lowest in epoch one (0.704).

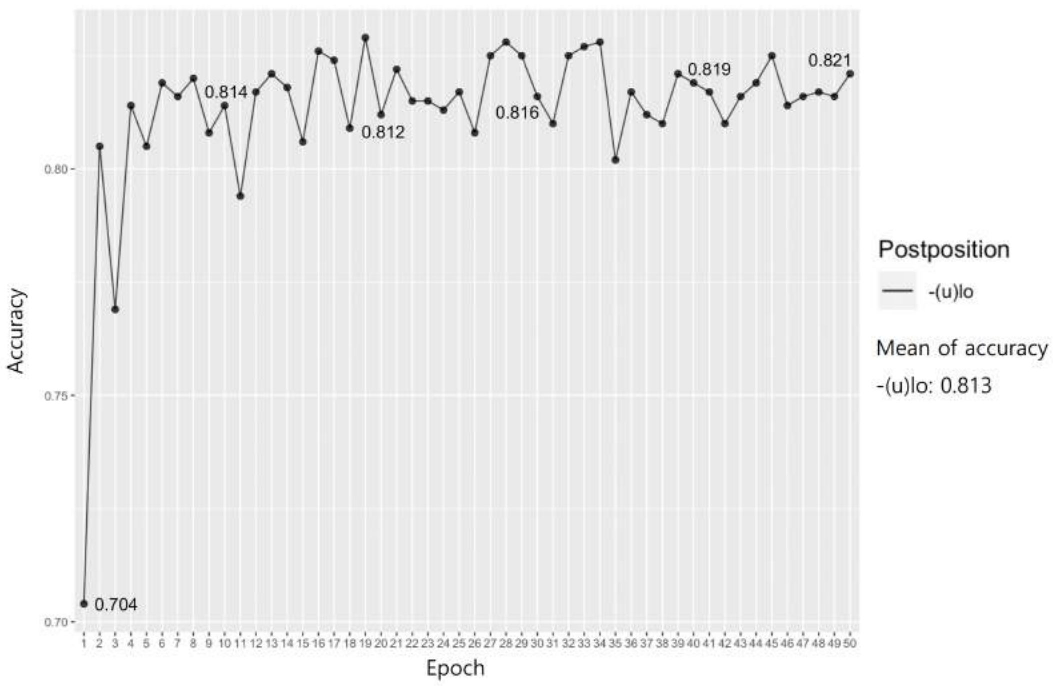

Figure 8.6: By-epoch accuracy for the BERT model: *-(u)lo*

The performance of the BERT classification model for *-(u)lo* varied depending on its function, as presented in Figure 8.7 and Table 8.4. The average accuracy was the highest in DIR (0.938) and the lowest in LOC (0.106); the other functions yielded accuracy ranging from 0.815 to 0.278. The by-function classification accuracy of this postposition is categorized into two types: one group (LOC and EFF) increased gradually from zero, and the other group (CRT, DIR, FNS, and INS) started above zero. Considering that LOC and EFF are the functions that account for a smaller portion of the total corpus size, this result may be interpreted that BERT could recognize the less occurring functions as the epoch (i.e., learning) progressed.

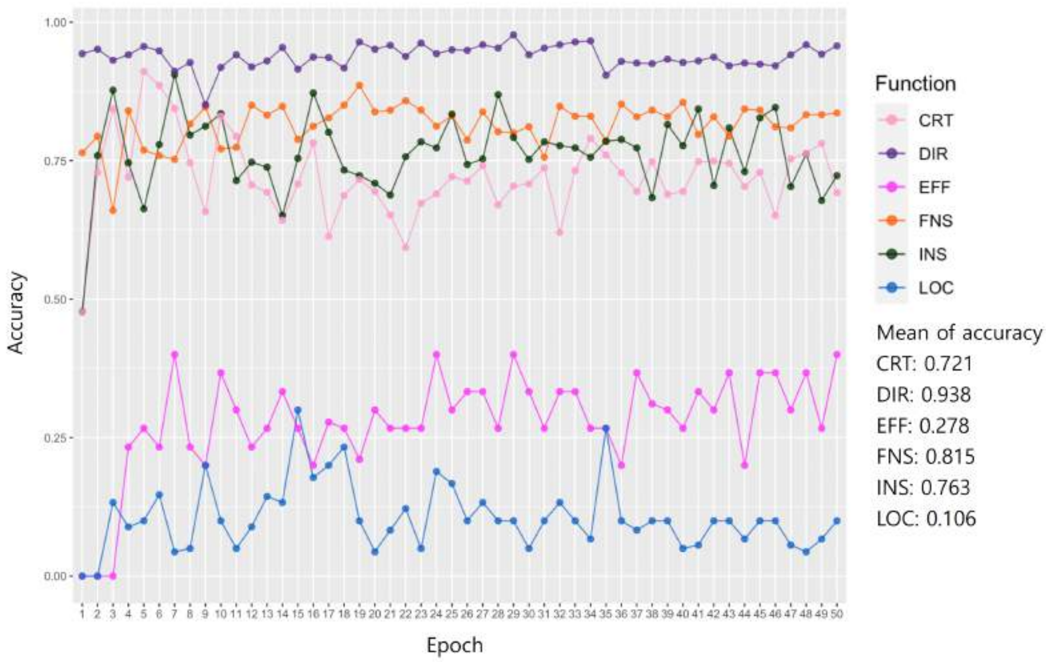

Figure 8.7: By-function accuracy curve for the BERT model: *-(u)lo*

Note. Abbreviation: CRT = criterion; DIR = direction; EFF = effector; FNS = final state; INS = instrument; LOC = location

Table 8.4: By-function accuracy for the BERT model: *-(u)lo*

Epoch	Classification accuracy					
	CRT	*DIR*	*EFF*	*FNS*	*INS*	*LOC*
1	0.476	0.943	0	0.764	0.477	0
10	0.83	0.918	0.367	0.771	0.835	0.1
20	0.694	0.951	0.3	0.838	0.709	0.044
30	0.708	0.941	0.333	0.811	0.752	0.05
40	0.694	0.927	0.267	0.855	0.777	0.05
50	0.692	0.957	0.4	0.836	0.723	0.1

8.2.3 Correlation between corpus size and classification accuracy

The word-level embedding models (PPMI-SVD and SGNS) have shown that the classification accuracy of the functions that account for a larger portion of the total corpus size affects the accuracy of the model for each postposition (Section 5.2.3). Hence, I conducted the same correlation analysis of postposition to see if the same phenomenon also occurred with BERT. For this task, the Pearson Correlation was used to calculate the correlation score between the mean accuracy of the BERT model and of each function for these postpositions.

-ey

Among the eight functions of *-ey*, LOC and CRT occur most frequently in the corpus data. However, as shown in Table 8.5, in the BERT model, the mean accuracy of the model and that of each function had no correlation for these functions. I also found a high correlation with the functions that accounted for a smaller portion of the total corpus size. These results are contrary to those in the word-level embedding model and can be interpreted that the classification accuracy of the BERT model was affected less by the corpus size of each function of *-ey*.

Table 8.5: Correlation between the accuracy of the BERT model and of each function for *-ey* by epoch

Function	Corpus size	Correlation
LOC	1,780	-0.218
CRT	1,516	0.115
THM	448	0.708
GOL	441	0.691
FNS	216	0.733
EFF	198	0.553
INS	69	0.469
AGT	47	0.429

Note. Abbreviation: AGT = agent; CRT = criterion; EFF = effector; FNS = final state; GOL = goal; INS = instrument; LOC = location; THM = theme

-eyse

LOC accounts for more than 80% of the occurrences in the total corpus. However, as shown in Table 8.6, the overall accuracy has more correlation with SRC than LOC. This indicates that the BERT model was not strongly affected by the corpus size.

Table 8.6: Correlation between the accuracy of the BERT model and of each function for *-eyse* by epoch

Function	Corpus size	Correlation
LOC	4,206	0.283
SRC	647	0.593

Note. Abbreviation: LOC = location; SRC = source

-(u)lo

Of the six functions of *-(u)lo*, FNS and DIR account for the largest portion of occurrence in the total corpus. However, as presented in Table 8.7, the correlation score was the highest in EFF (0.581), which has the smallest portion, and the lowest in DIR (0.142) which has the second largest portion. This result is consistent with the results shown by *-ey* and *-eyse*. This further indicates that the BERT model was not strongly affected by the corpus size.

Table 8.7: Correlation between the accuracy of the BERT model and of each function for *-(u)lo* by epoch

Function	Corpus size	Correlation
FNS	1,681	0.505
DIR	1,449	0.142
INS	739	0.499
CRT	593	0.255
LOC	158	0.151
EFF	88	0.581

Note. Abbreviation: CRT = criterion; DIR = direction; EFF = effector; FNS = final state; INS = instrument; LOC = location

Contrary to the results of the word-level embedding models, the BERT model was not particularly affected by the corpus size. This is because the BERT model assigns each word to a vector that is sensitive to the context in which it appears. This is a major difference from the traditional word-level embedding models. In addition, the BERT model operates on the basis of the pre-trained model, which means that it already has enough information on the target language.

8.3 Visualization system: clusters of sentence-level embeddings

The visualization system aimed to identify the sentence-level embeddings interactively in order to see how BERT classified the functions of the postpositions in each epoch. To carry this out, I selected two-dimensional t-SNE data of testing (the number of sentences of each postposition; *-ey*: 467, *-eyse*: 484, and *-(u)lo*: 467).

To statistically confirm changes of the sentences containing different functions of each postposition according to each epoch, I performed a cluster analysis that allows the identification of groups that share common characteristics and the relationship between data (Romersburg, 1984). Comparing the advantages of clustering (see Section 5.3.1) and of density-based clustering (Sander et al., 1998), I chose to use density-based clustering through R (R version 3.6.2; R Core Team, 2019), by adapting *dbscan* package (Hahsler et al., 2019). I recommend seeing the following findings while demonstrating the visualization system together [1].

[1]PostBERT, the second visualization system is available at: https://seongmin-mun.github.io/VisualSystem/Major/PostBERT/index.html

8.3.1 *-ey*

Figure 8.8 shows how many clusters were generated as the epoch progressed. When the epoch was one, all of the sentences were divided into two groups. However, as the epoch progressed, the sentences were divided into three in epoch seven, four in epoch 12, and five in epoch 15. The details of the sentence-level embedding outcomes for *-ey* of these epochs are shown in the following Figures (Figure 8.9 to 8.12).

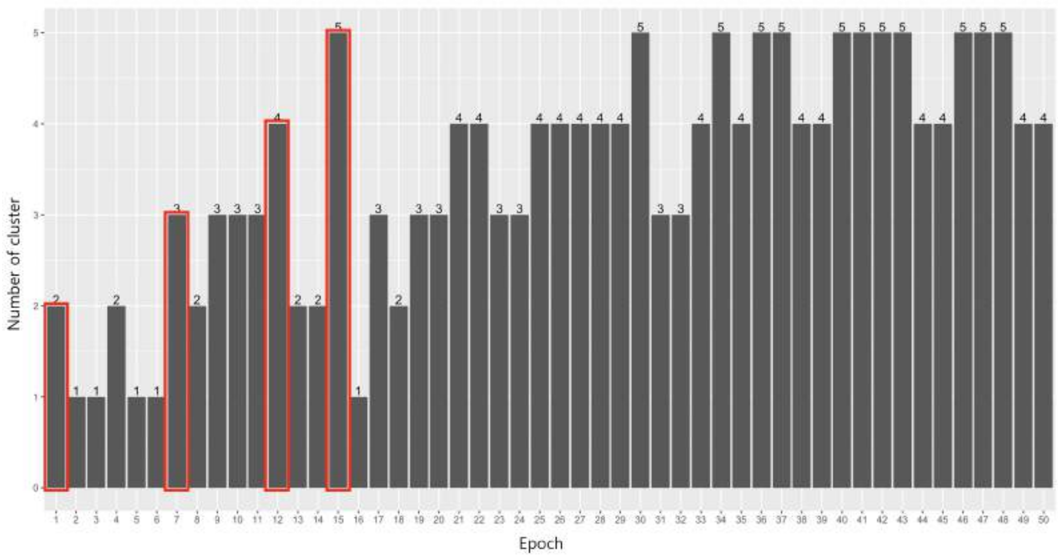

Figure 8.8: Number of density clusters in each epoch: *-ey*

Figure 8.9 shows the distributional map for *-ey* in epoch one. BERT classified the sentences into two groups, CRT and LOC, which are the functions that have a larger portion of the total corpus size. This means that it recognized LOC and CRT well in the early step of the epoch (i.e., learning). This can a reason why the BERT model showed a high classification accuracy only for LOC (0.911) and CRT (0.876) at epoch one, as shown in Figure 8.3.

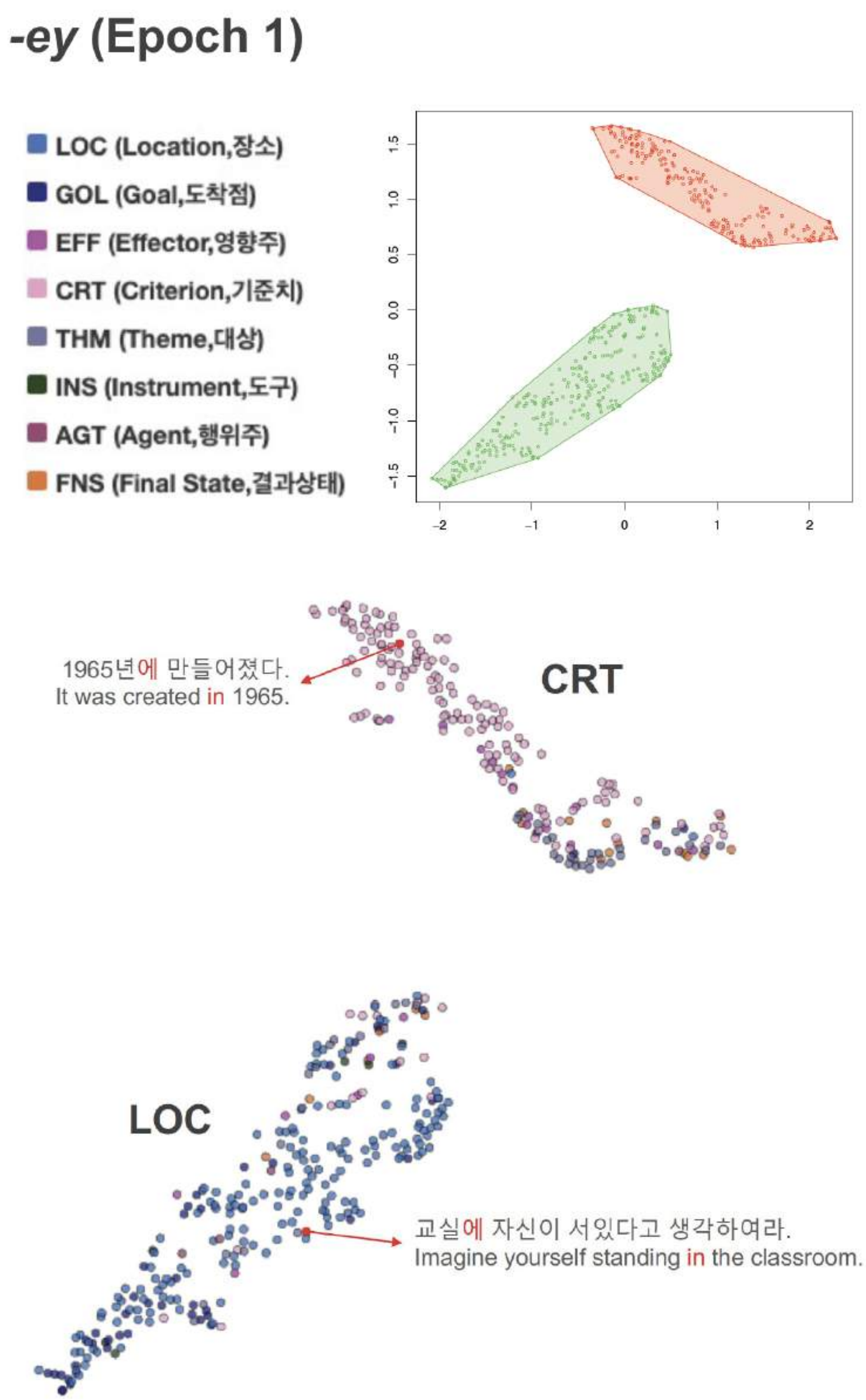

Figure 8.9: The distributional map for *-ey* in epoch one

Figure 8.10 shows the results of *-ey* when the epoch progressed to seven. BERT classified the sentences into three groups. The first was a group of sentences gathered around the LOC. Most of the functions for *-ey* contained in the sentences were LOC, however, at the bottom of this group, there were a number of sentences that functioned as GOL. The second was a group of sentences including THM, FNS, and EFF. In a density cluster, the three functions are shown to be one group, but in visualization, each is divided into an individual group. The final was a group of sentences gathered around CRT, which was recognized as its own group since the epoch was one.

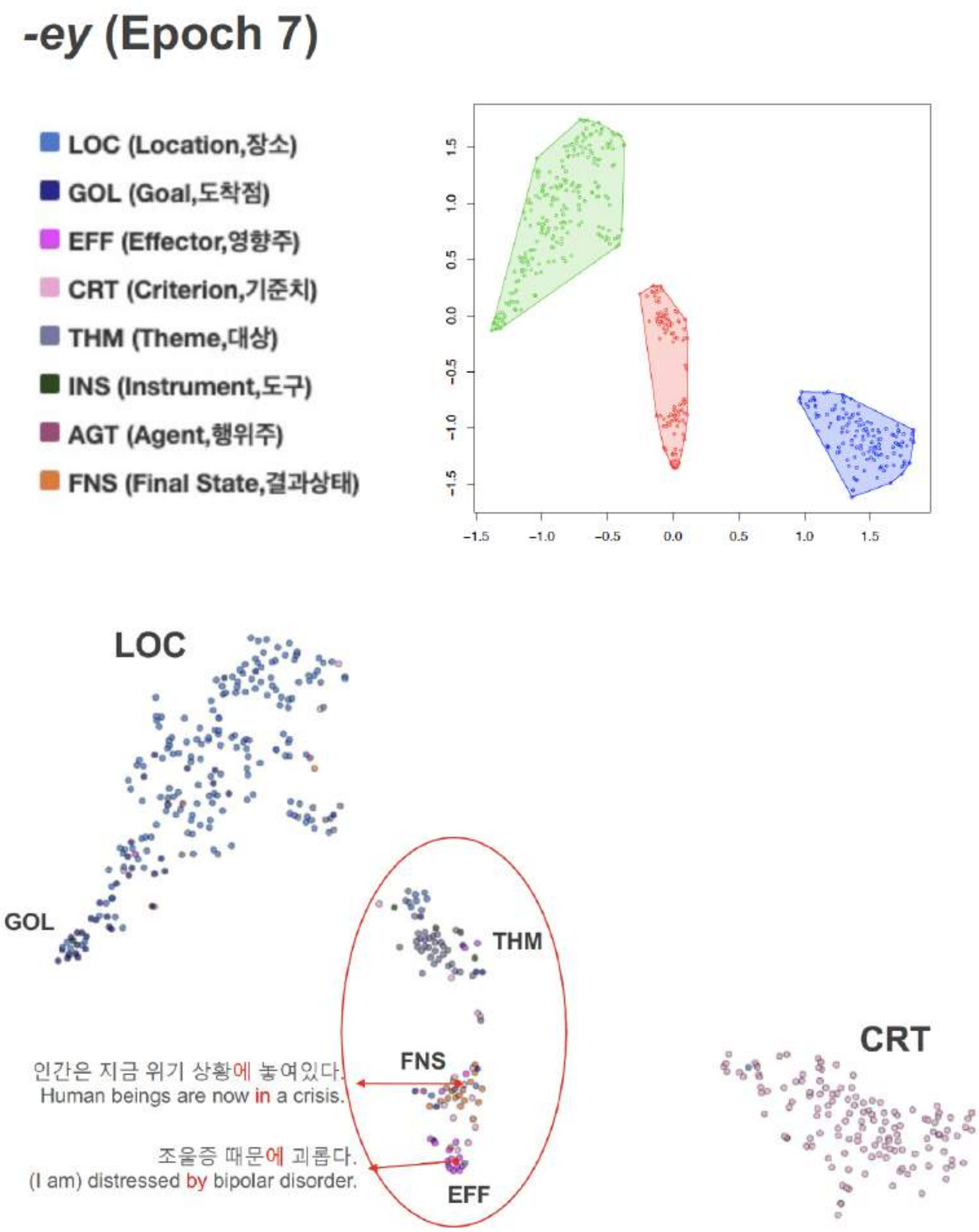

Figure 8.10: The distributional map for *-ey* in epoch seven

Figure 8.11 shows the results of *-ey* when the epoch progressed to 12. BERT classified the sentences into four groups. Particularly, THM was divided from EFF and FNS and made a separate group.

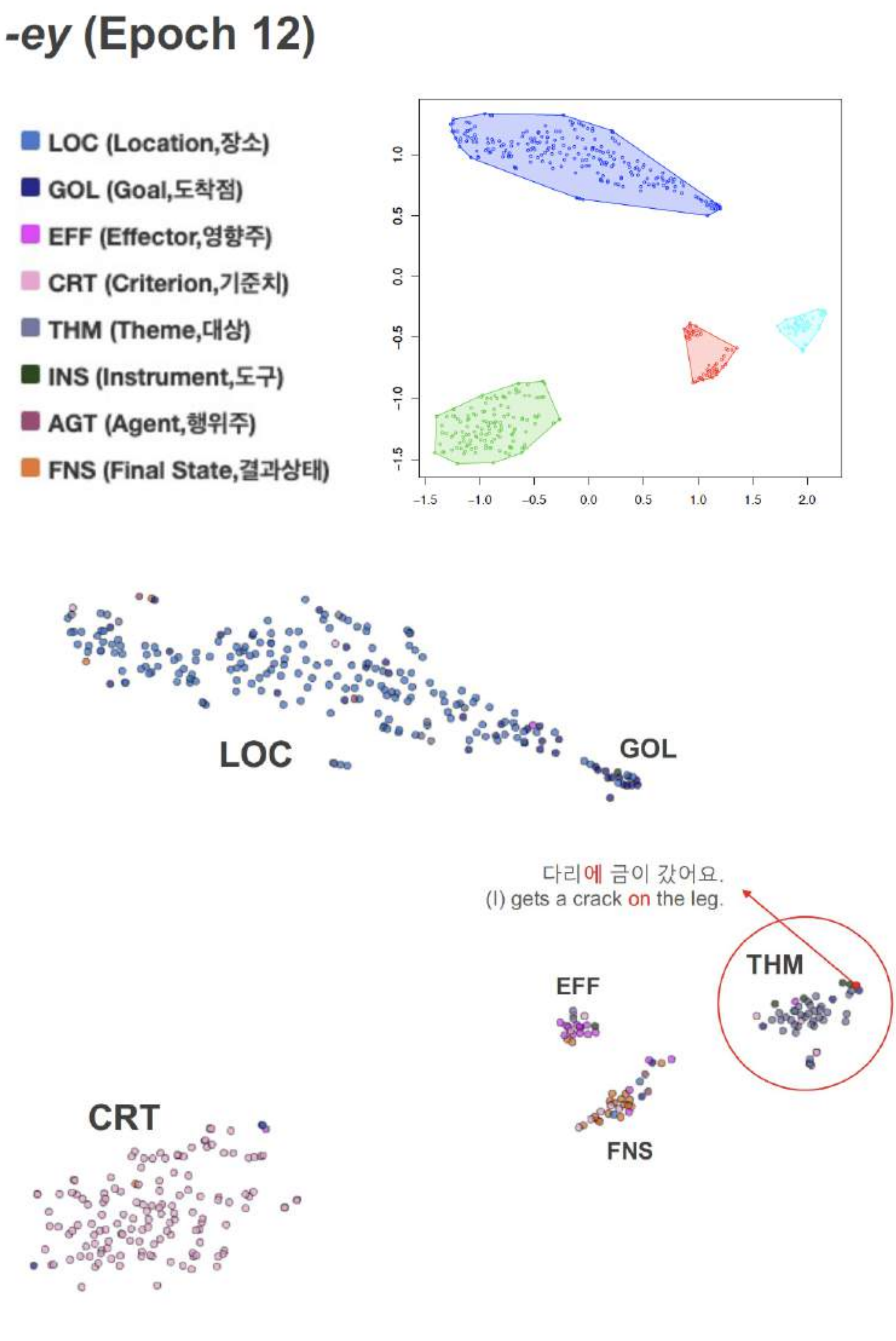

Figure 8.11: The distributional map for *-ey* in epoch 12

Figure 8.12 shows the results of *-ey* when the epoch increased to 15. BERT classified the sentences into five groups. GOL was divided from the LOC group and created a separate group. However, AGT and INS, which account for a smaller portion of the total corpus size, did make an individual group. This indicates that AGT and INS are very hard to be understood as distinguishable functions of *-ey*, even for BERT.

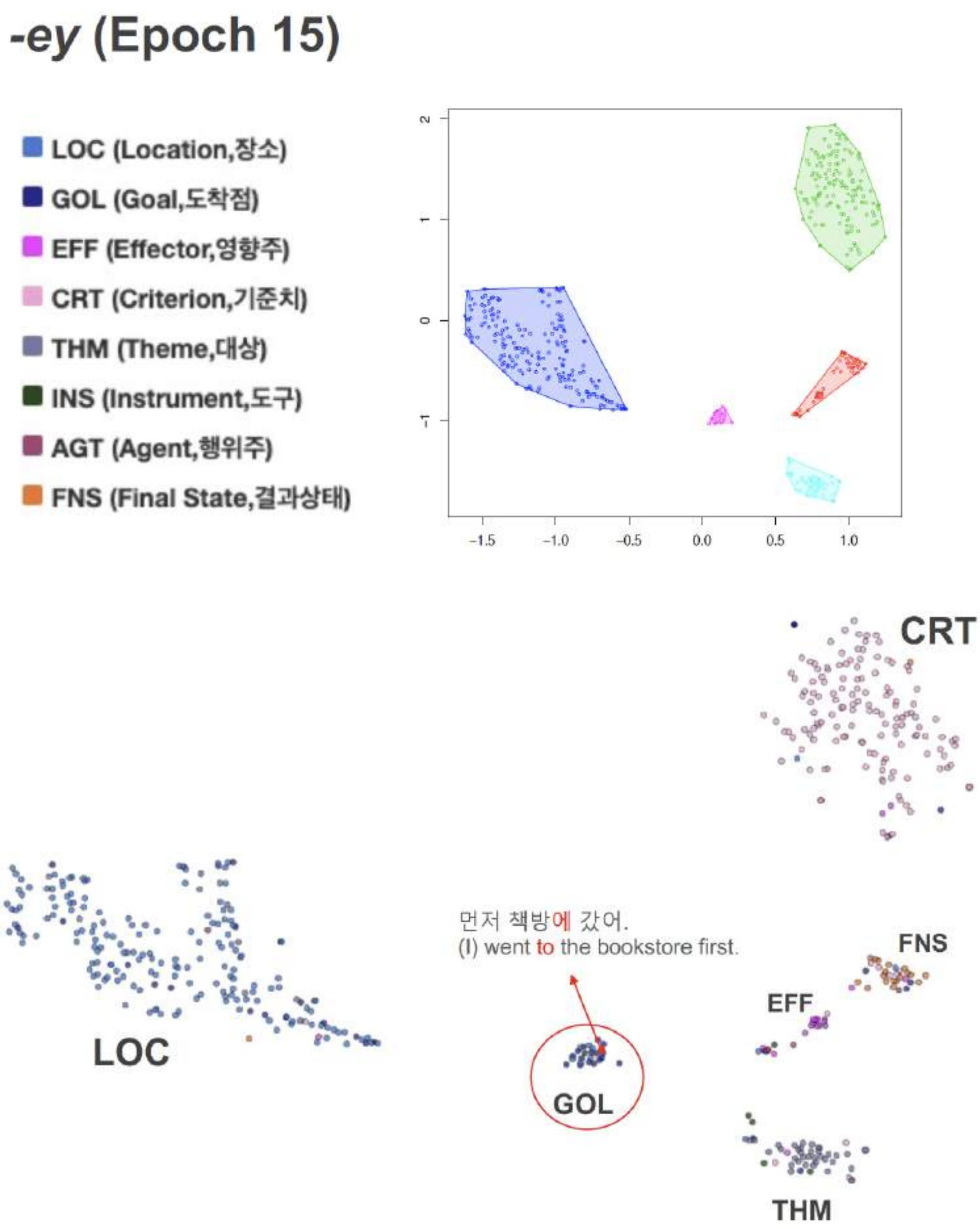

Figure 8.12: The distributional map for *-ey* in epoch 15

8.3.2 *-eyse*

Figure 8.13 shows how many clusters were created as the epochs progressed. When the epoch was one, the number of clusters was one. However, when the epoch was nine, there were two clusters. The details of the sentence-level embedding outcomes at these epochs are shown in the following Figures (Figure 8.14 and 8.15).

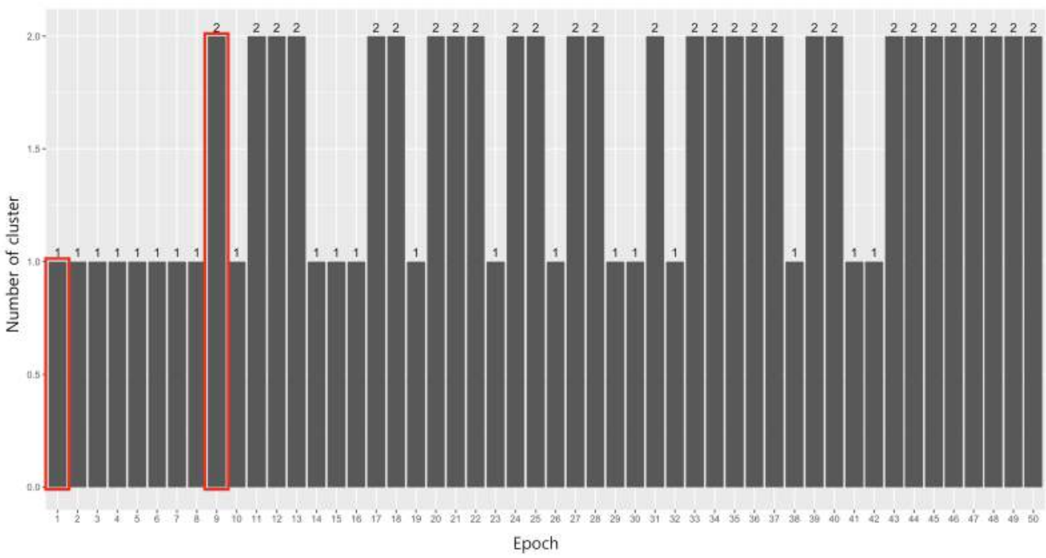

Figure 8.13: Number of density clusters in each epoch: *-eyse*

Figure 8.14 shows the distributional map for *-eyse* in epoch one. BERT recognized all of the sentences as one group, which means that it did not understand the differences between the functions. However, LOC was located at the top of the group, and SRC at the bottom.

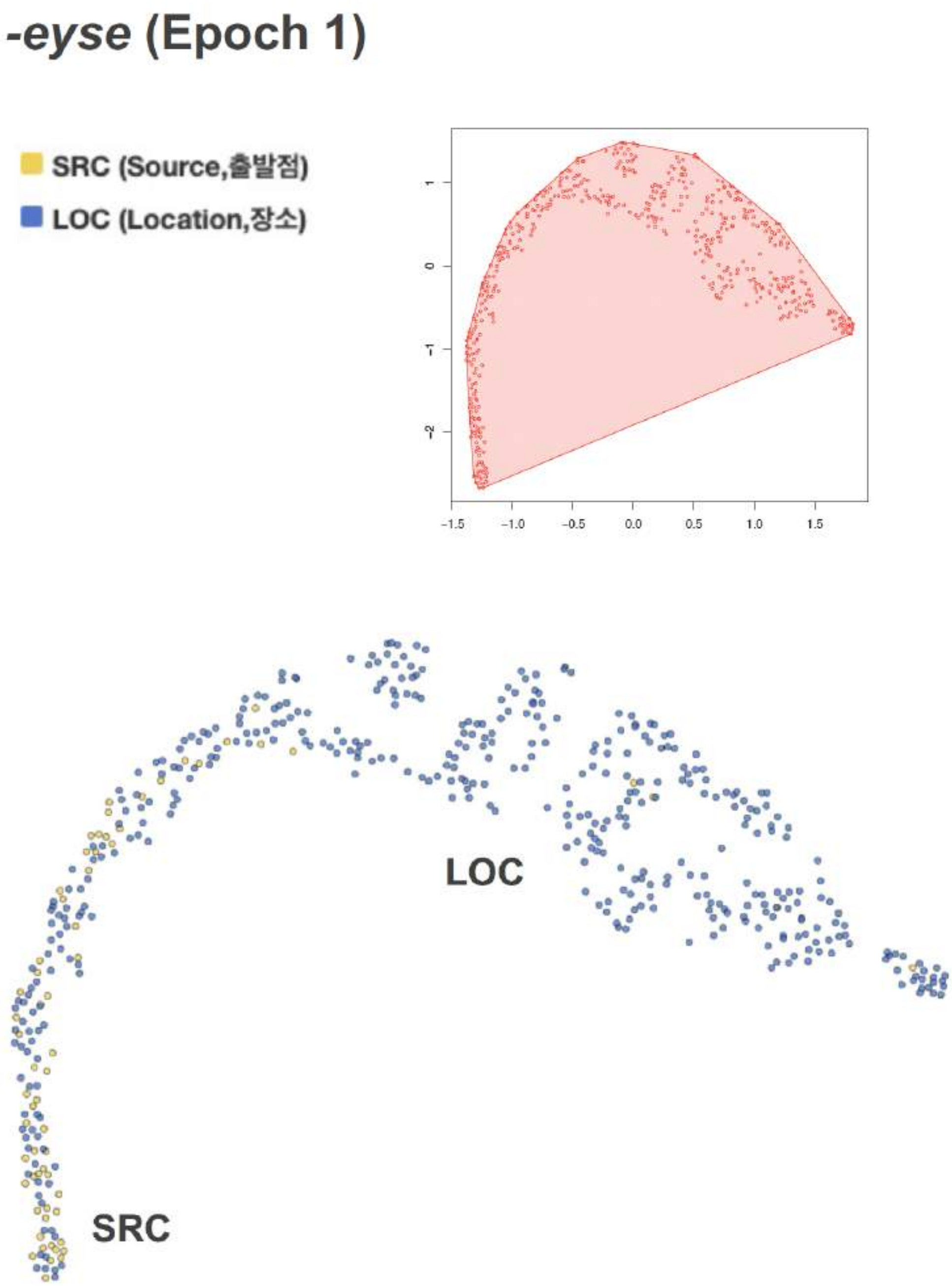

Figure 8.14: The distributional map for *-eyse* in epoch one

Figure 8.15 shows the results of *-eyse* when the epoch progressed to nine. BERT classified the sentences into two groups (LOC and SRC). From this epoch onward, BERT often showed two groups.

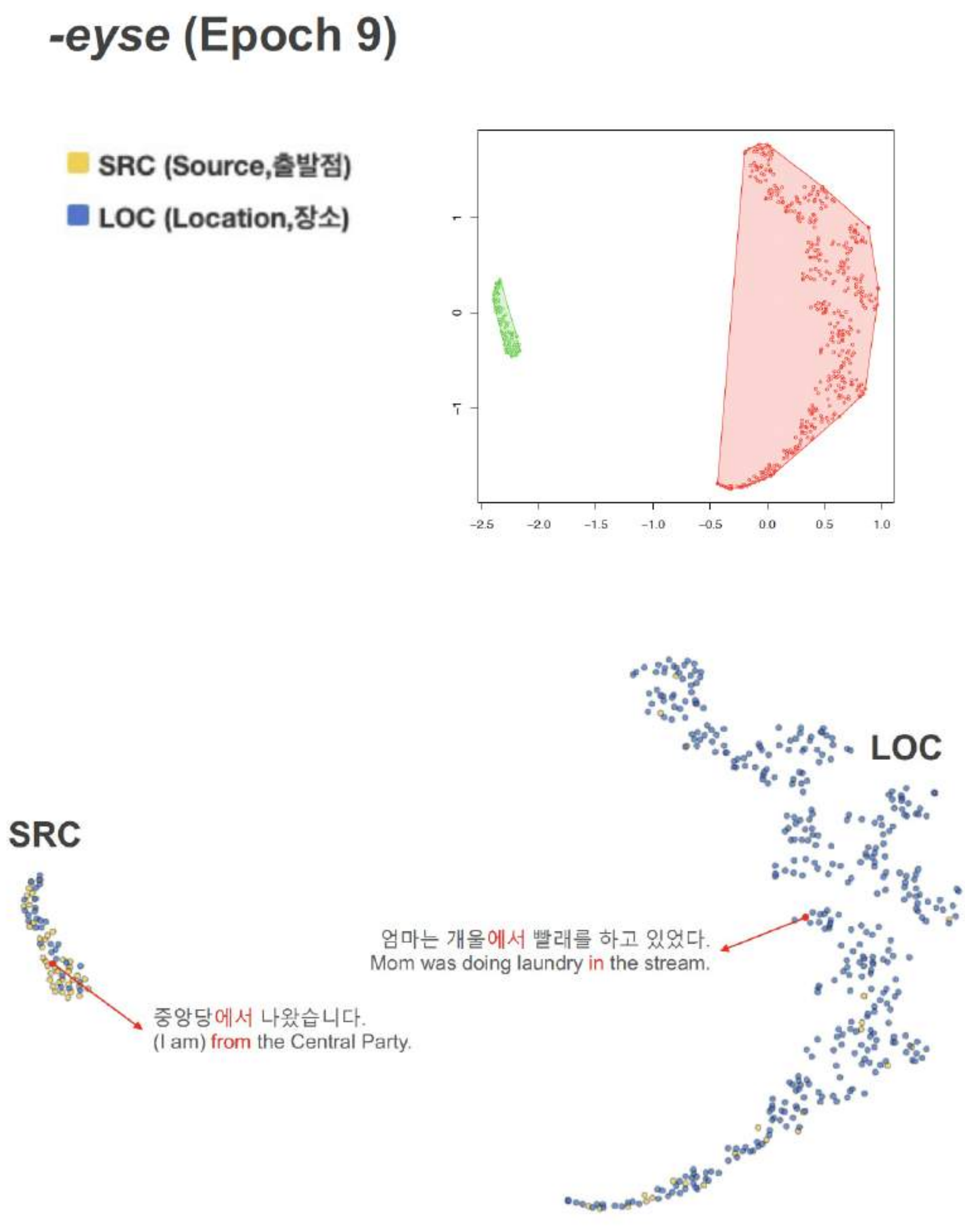

Figure 8.15: The distributional map for *-eyse* in epoch nine

8.3.3 *-(u)lo*

Figure 8.16 shows how many clusters were generated for *-(u)lo*. When the epoch was one, all of the sentences were grouped into one. However, as the epoch progressed, the sentences were divided into three in epoch four, five in epoch 12, and six in epoch 46. The distributional maps for *-(u)lo* of these epochs are shown in the following Figures (Figure 8.17 and 8.20).

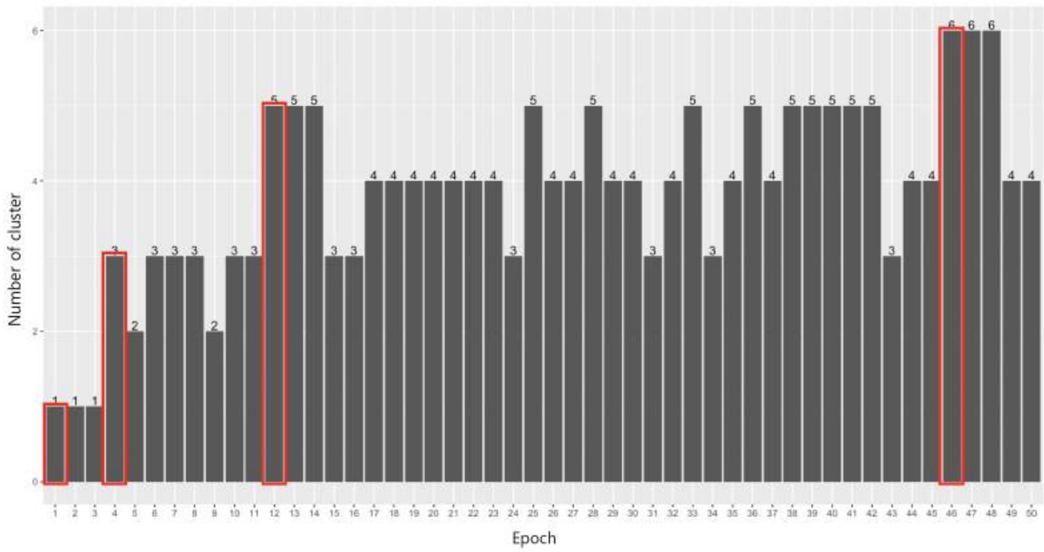

Figure 8.16: Number of density clusters in each epoch: *-(u)lo*

Figure 8.17 shows the distributional map for *-(u)lo* when the epoch was one. BERT classified all of the sentences into one group. However, DIR was located at the bottom right of cluster, while FNS at the top.

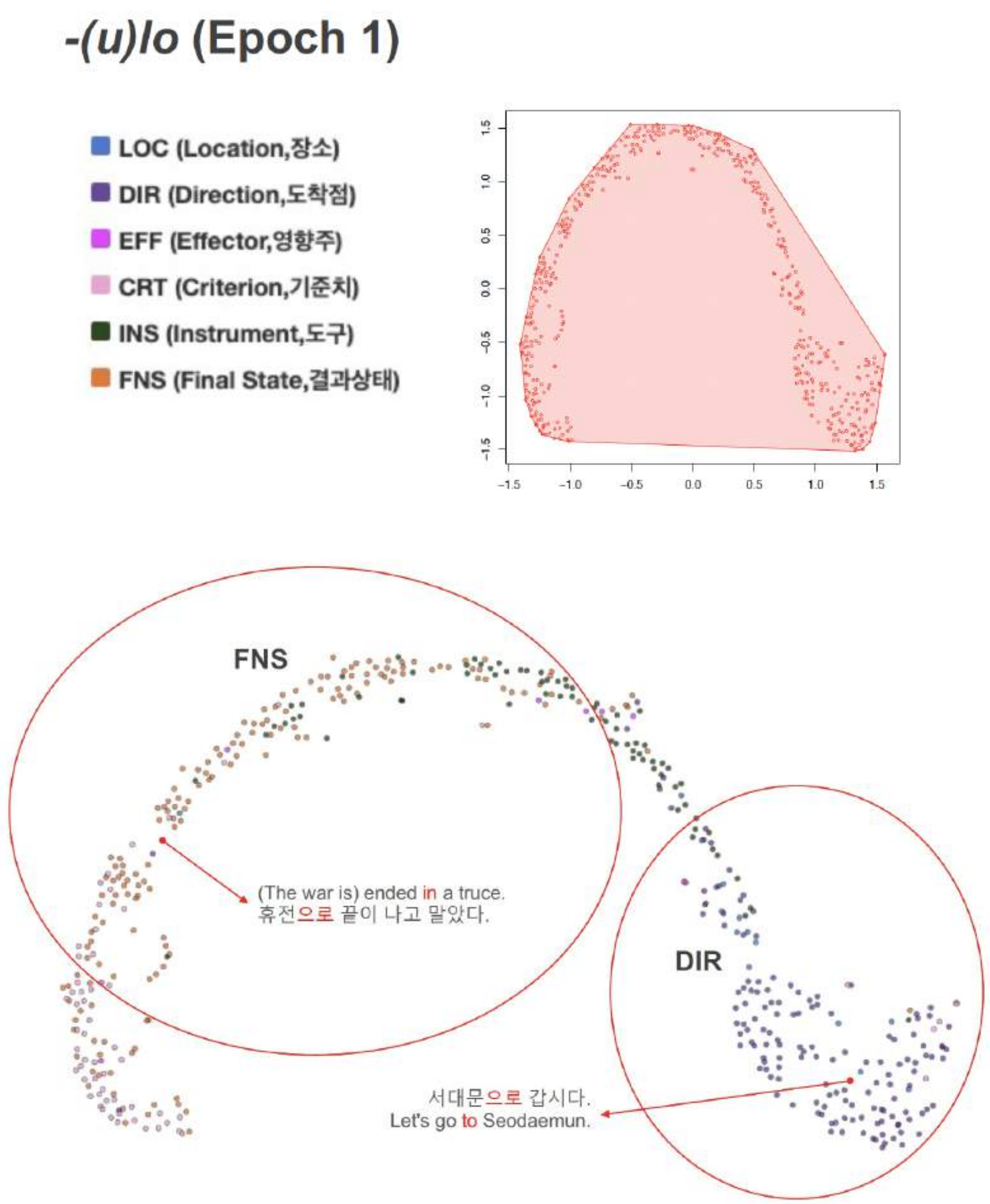

Figure 8.17: The distributional map for *-(u)lo* in epoch one

Figure 8.18 shows the results of *-(u)lo* when the epoch increased to four. BERT classified the sentences into three groups. The first was a group of sentences gathered around DIR. The second was a group of sentences including FNS, INS, and EFF. The final was a group of sentences gathered around CRT.

In this case, CRT was distinguished from the other functions since epoch four. This can be the reason to explain why the classification accuracy of CRT increased gradually from epoch one (0.476) to four (0.72).

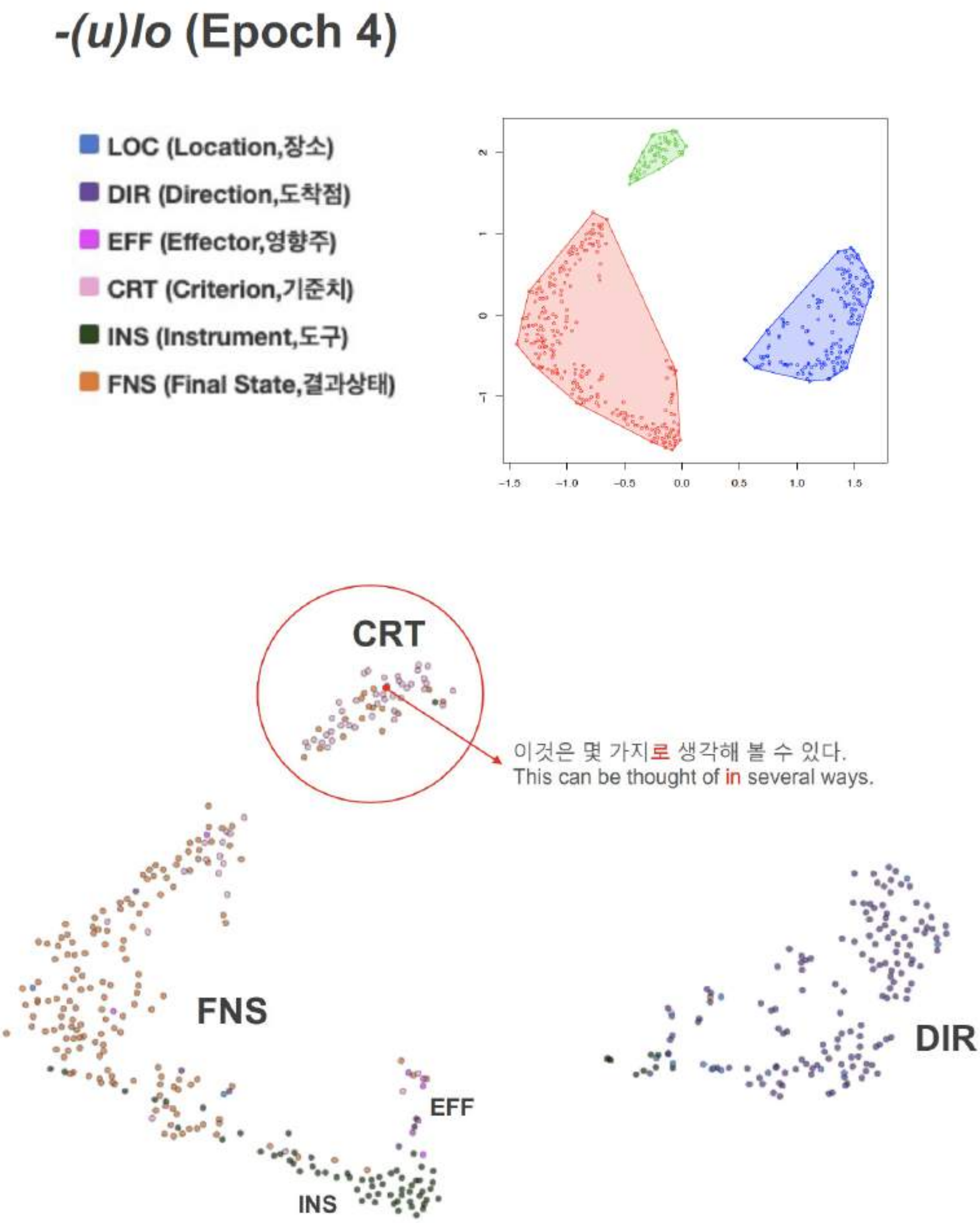

Figure 8.18: The distributional map for -*(u)lo* in epoch four

Figure 8.19 show the results of *-(u)lo* when the epoch increased to 12. BERT classified the sentences into five groups. EFF and INS were divided from the FNS group and each created a separate group. Considering that EFF accounts for a smaller portion of the total corpus size, it can be interpreted that BERT can recognize the functions as the epoch progressed, even for the less occurring functions.

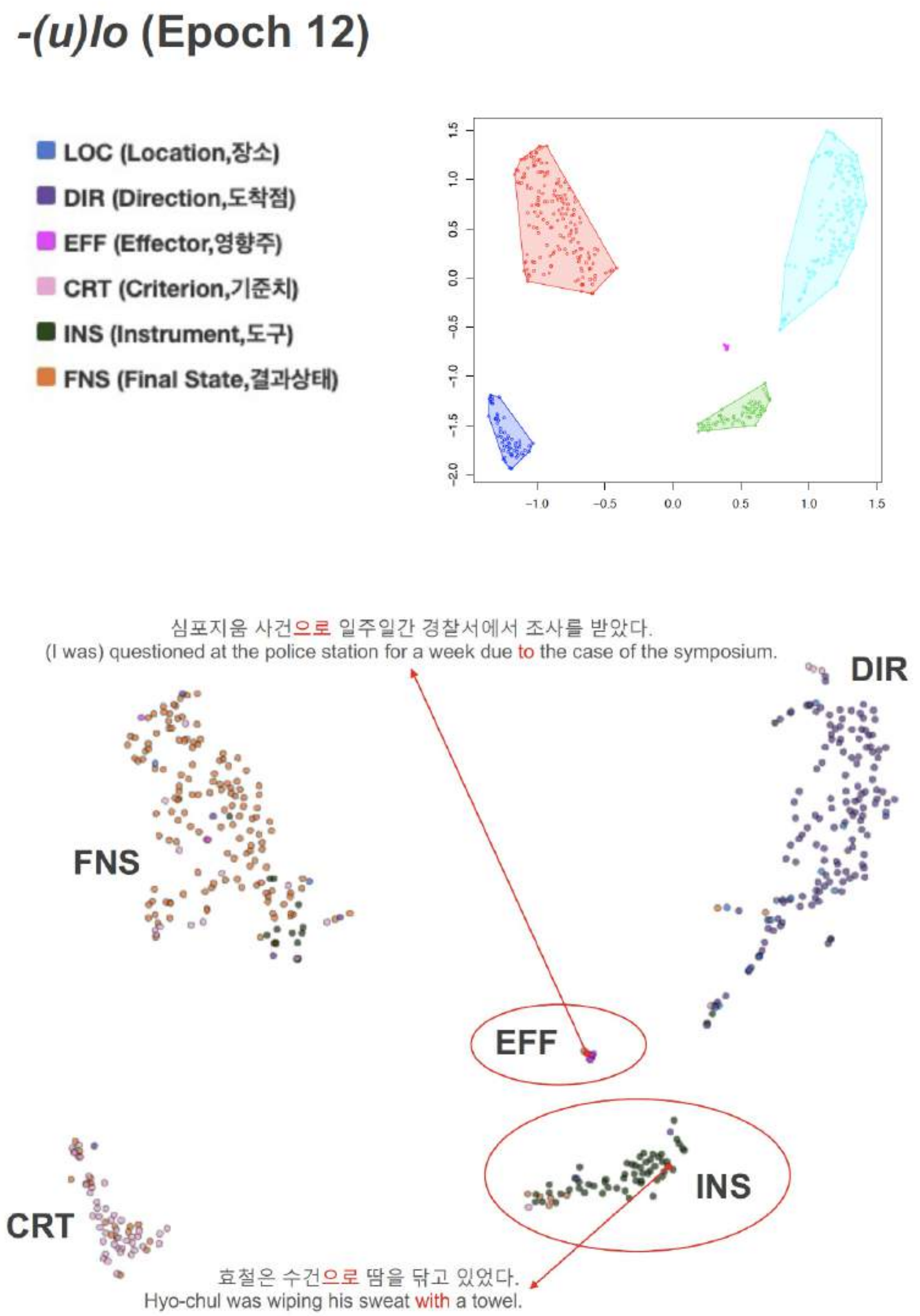

Figure 8.19: The distributional map for -*(u)lo* in epoch 12

Finally, Figure 8.20 shows the results of *-(u)lo* when the epoch progressed to 46. BERT classified the sentences into six groups, which is the same number of functions that *-(u)lo* has. However, the newly generated group, LOC, was created by collecting a few sentences from all the different functions. As shown in Figure 8.20 (2), most of the sentences (11 out of 15) belonged to the DIR group.

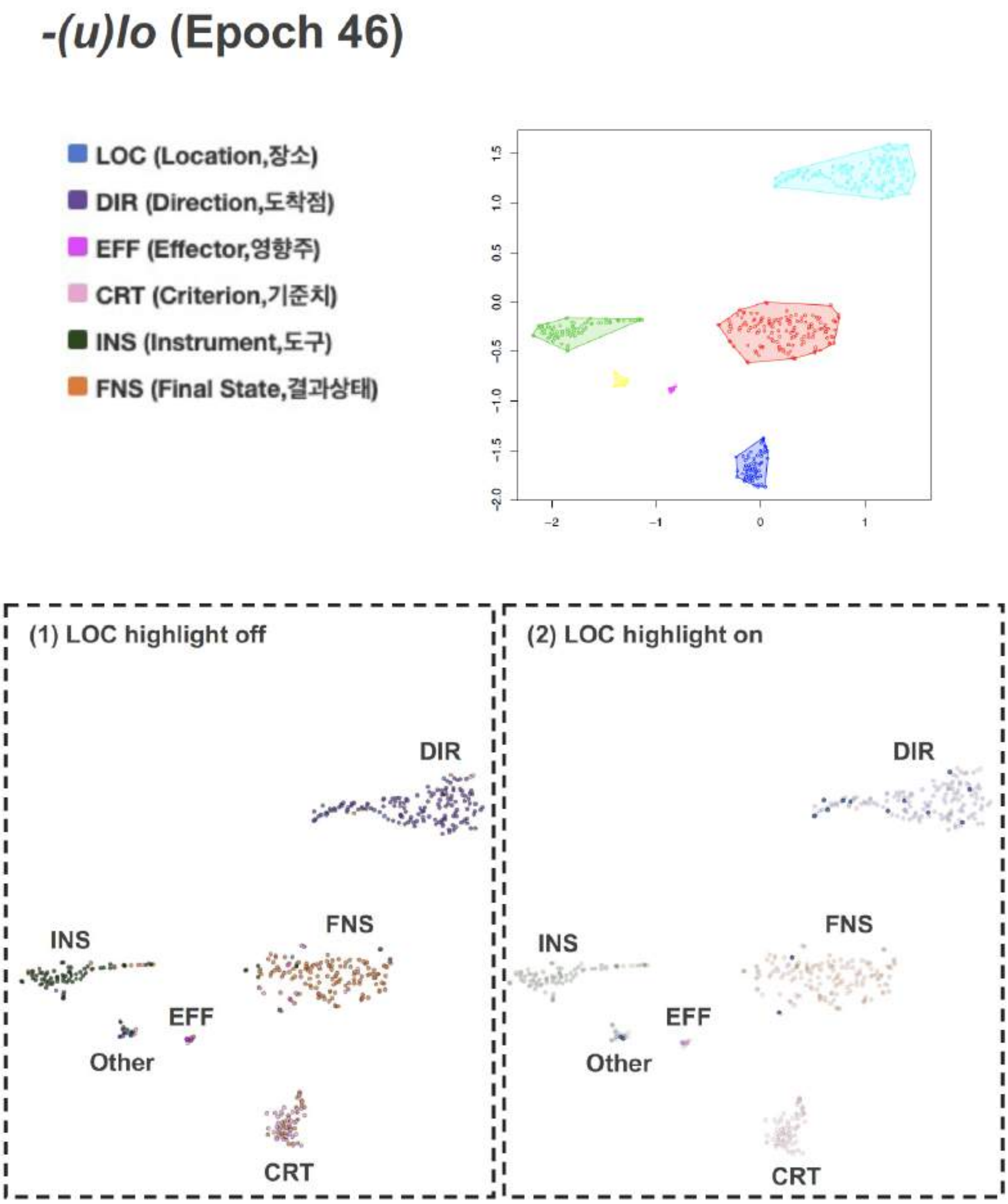

Figure 8.20: The distributional map for *-(u)lo* in epoch 46

8.3.4 Interim summary of visualization results

The visualization system interactively showed the results by the options (e.g., postposition types, epochs) and showed the relation between sentences. Using this system, I investigated the third research question concerning how the BERT model recognizes the polysemy involving Korean adverbial postpositions in each epoch. Overall, the result showed that the model tended to demonstrate more distinctive clustering as the epoch progressed, with a high level of coherence for specific function. For instance, in epoch 12 for *-(u)lo* (Figure 8.19), a cluster of EFF (the function with low-frequency of occurrences in the data) emerged. This finding further supports the idea that by using sufficient epochs, the BERT model can identify functions at a satisfactory level, even though they are relatively infrequent. In addition, as shown in Figure 8.20, LOC could not form a designated cluster in the end. Many of the instances (11 out of 15) belonged to the DIR group. This is because of the low frequency of LOC in the data and the semantic closeness between DIR and LOC—they are both related to a location and are often difficult to distinguish one from the other.

8.4 Discussion of the Chapter

In this chapter, I described the model performance in the classification of the functions of the postpositions *-ey*, *-eyse*, and *-(u)lo*. Below are the three major findings that could answer the research questions.

First, the higher classification accuracy was obtained when the postposition has a fewer number of functions. The previous word-level embedding models have shown that the different numbers of functions (e.g., two for *-eyse*, six for *-(u)lo*, and eight for *-ey*) affect the classification accuracy. Similar to these models, BERT also showed that the classification accuracy is affected by the number of functions that the postposition has. The average accuracy was 0.815 in *-ey*, 0.898 in *-eyse*, and 0.813 in *-(u)lo*. Nevertheless, considering that the word-level embedding models have shown large differences of model performance between each postposition (e.g., PPMI-SVD: *-ey* (0.534), *-eyse* (0.773), *-(u)lo* (0.567); SGNS: *-ey* (0.204), *-eyse* (0.693), *-(u)lo* (0.368)), it can be interpreted that the BERT model is less affected by the number of functions that the postposition has.

Second, the BERT model was not influenced by the corpus size of each function that a postposition has, which is the opposite of the results shown by word-level embedding models and the second Hypothesis (see Section 8.1). Considering that the BERT model assigned each word to a vector based on the context information and operated on the basis of the pre-trained model, it had much more information on the attested language than the word-level embedding models. For this reason, it was able to recognize the functions of each postposition with less influence of corpus size on model performance. In addition, it considers much more contextualized information (i.e., *token embeddings*, *segment embeddings*, and *position embeddings*) than word-level embedding models, which use only the morphological information of the word. This can also be a reason why BERT was less affected by the corpus size.

Third, as the epoch (i.e., learning) progressed, BERT could recognize the functions of each postposition, even when the functions account for a smaller portion of the entire corpus. This finding is contrary to the results shown by word-level embedding models but is consistent with Hypothesis 3 (see Section 8.1). One crucial issue of word-level embedding models was that the accuracy was low for the classification of the functions that account for a smaller portion of the total corpus size. However, when the epoch was progressed, BERT could recognize the differences between the functions. This finding further supports the idea that by using sufficient epochs, the BERT model can identify functions at a satisfactory level, even though they occur relatively infrequently. However, despite this advantage with regards to the data size, the BERT model still seems to be subject to the extremely low-frequent items and/or the semantic closeness between the items, limiting its performance in the given task to some extent.

In addition to the result of BERT, the model also showed high classification accuracy. The average classification accuracy for *-ey*, *-eyse* and *-(u)lo* were around 0.815, 0.898, and 0.813, respectively. This is a very high classification accuracy, considering that for the same tasks, previous studies reported the average accuracy ranging from 0.882 (Kang and Park, 2003) to 0.623 (Bae et al., 2014) and the word-level embedding model used in this dissertation showed the average classification accuracy of 0.550. Overall, the BERT model solved the problems shown by the word-level embedding models in a task to identify the functions of each postposition. Furthermore, I found that the BERT model was more suitable for the task of classifying the functions of the postposition resulting in a very high classification accuracy.

8.5 Summary of the Chapter

In this chapter, I reported the findings of the classification models and visual inspections, starting from my hypotheses on the research questions.

From the results of model performance and visualization, I found three major findings. First, the BERT model is affected by the number of functions that the postposition has. However, the gaps of model performance between each postposition are smaller than word-level embedding models. Second, the classification accuracy of the BERT model was less affected by the corpus size, which is different from the performance of the word-level embedding models. Third, when the epoch progressed, the BERT model could recognize more functions of the postposition, including the one that account for a smaller portion of the corpus size. Moreover, the BERT model showed higher classification accuracy than previous studies including the word-level embedding models used for the same task in this dissertation.

In the following chapter, I will discuss the interpretations of the three word embedding models with regards to the research questions (see Chapter 4 and Chapter 7).

Chapter 9

Discussion

This chapter discusses the interpretations of the findings of the word-level embedding models (see Chapter 5), and sentence-level embedding model (see Chapter 8), in relation to the research questions (see Chapter 4 and Chapter 7). In addition, it also discusses the advantages and limitations of each model for resolving word-level polysemy of Korean adverbial postpositions.

9.1 Interpretations of word-level embedding models: PPMI-SVD and SGNS

The research questions in Chapter 4 are re-stated as follows:

- Research question 1: How does the number of functions a postposition has, affect classification performance for each word-level embedding model?
- Research question 2: What is the role of the context window in the classification performance of each word-level embedding model?

- Research question 3: How does the cluster of postpositions and their co-occurring words change as the environments of word-level embedding change?

9.1.1 The number of functions in each postposition

For first research question, I made a hypothesis with respect to the number of functions in each postposition as below:

- Hypothesis: The accuracy of the classification should be inversely proportionate to the number of functions of a postposition.

In previous studies focusing on the same three adverbial postpositions, it was reported that the multiple functions of one postposition delivers recognition and ambiguous usage (e.g., Choo and Kwak, 2008, Sohn, 1999). As stated in Chapter 2, the three postpositions have different numbers of functions (e.g., two for *-eyse*, six for *-(u)lo*, and eight for *-ey*). Based on this fact, I predicted that if the postposition has more functions, the classification models would produce lower accuracy. This prediction was investigated in Chapter 5 by exploring the classification performance of word-level embedding models with each postposition.

The results proved the prediction to be true as there was an inverse relation between the classification accuracy and the number of functions of each postposition. For instance, the PPMI-SVD model showed that the classification accuracy was the highest in *-eyse* (0.773) and the lowest in *-ey* (0.534); *-(u)lo* showed classification accuracy of 0.567. Similar to the PPMI-SVD model, *-eyse* outperformed the other postpositions (e.g., *-ey* (0.204), *-eyse* (0.693), *-(u)lo* (0.368)) in the SGNS model, which is consistent with the

Hypothesis.

I further explored the relationships between corpus size and classification accuracy by conducting a correlation analysis by postposition. This is because a phenomenon was found, that the average model performance was similar to the accuracy patterns of the functions that occupy a larger portion of the total corpus size. As a result of my investigation, I found that this was true for two word-level embedding models. For instance, as shown in Table 5.3 (i.e., correlation between two models and each function for *-ey*), the mean accuracy of the two models were highly correlated to the mean accuracy of the two most frequent functions, LOC (e.g., PPMI-SVD: 0.983; SGNS: 0.797) and CRT (e.g., PPMI-SVD: 0.907; SGNS: 0.854). With regard to the relationship between the corpus size and model performance, previous research has reported that the PPMI-SVD model used the word-word matrix in the process of converting words to vectors, and therefore, was sensitive to the token frequency (Jurafsky and Martin, 2019). Furthermore, the SGNS model used one-hot encoding for the same process, and therefore relied on the type frequency (Mikolov et al., 2013a). Considering that both the token frequency and the type frequency are sensitive to the corpus size (e.g., Jurafsky and Martin, 2019, Mikolov et al., 2013a), it can be implied that both word-level embedding models are affected by corpus size.

9.1.2 The role of context window size

The second research question was represented as a hypothesis regarding the role of the context window size as below:

- Hypothesis: The accuracy of the classification should be higher in smaller window sizes.

The context window size is a range of words surrounding a target word, which affects the determination of the characteristics of the word (Lison and Kutuzov, 2017). Previous studies have shown that the smaller windows work better for syntactic representation and the larger windows contribute more to semantic representation (e.g., Jurafsky and Martin, 2019, Levy et al., 1999). Moreover, they have shown the advantage of smaller window size in addressing word-level polysemy (e.g., Bullinaria and Levy, 2012, Levy and Goldberg, 2014). I thus predicted that the two word-level embedding models will perform better in smaller context window sizes. This prediction was investigated in Chapter 5 by manipulating the context window size of word-level embedding models.

I found that the two word-level embedding models had performances that varied from each other. For instance, the PPMI-SVD model obtained high classification accuracy at larger window sizes. However, the SGNS model obtained low classification accuracy, regardless of the window size in the case of *-ey* and *-(u)lo*, but high classification accuracy for *-eyse* at larger window sizes. These results were contrary my hypothesis. Considering that the larger windows contribute more to semantic representation (e.g., Jurafsky and Martin, 2019, Levy et al., 1999), two word-level embedding models may perform more semantically than syntactically.

9.1.3 The changes in the relationship between postposition and their co-occurring words

Based on the third research question, I made a hypothesis about the changes in the relationship between the postpositions and their co-occurring words as follow:

- Hypothesis (on hyperparameters): The clusters and their co-occurring words should vary depending on the environments of word-level embedding (2 models * 3 postpositions * 10 window sizes).

Previous studies have shown different embedding results depending on the models, window sizes, or corpus used in the study based on their purpose (e.g., Bullinaria and Levy, 2007, 2012, Hilpert, 2016, Levy and Goldberg, 2014, Turney and Pantel, 2010). Considering this fact, I assumed that the two word-level embedding models would show different embedding results based on the environments (2 models * 3 postpositions * 10 window sizes). This assumption was explored in Chapter 5 by developing a visualization system to see the changes in the relationship between the postpositions and their co-occurring words by the environments of the word-level embedding models.

To statistically investigate the changes, I conducted a series of cluster analysis by using density-based clustering (Sander et al., 1998). As a result of this exploration, I found that the cluster was not changed much by the environments of word-level embedding models. The density clusters of the two models were gathered into one or two clusters in the end. However, there were a few different points between the two models. For example, in the PPMI-SVD model, the words that were used most frequently in the corpus were located in the center of the cluster. On the other hand, in the

SGNS model, the words were distributed rather evenly, regardless of word frequency. This was because the PPMI-SVD model worked based on the token frequency and the SGNS model worked based on type frequency (e.g., Jurafsky and Martin, 2019, Mikolov et al., 2013a).

In addition, I investigated the relationships between each postposition and their surrounding words by using a visualization system. Through this, I found that there were two different types of relationships between the particular postposition and its co-occurring words. First, there was a word group that appeared only when the postposition was used as a specific function. The words in this group did not appear when this postposition was used as another function. For example, when the postposition *-ey* was used with the function THM (i.e., theme), it was found in Figure 5.19 that there was a strong relationship between it and *kwanha/VV* based on the cosine similarity of 0.999. To see more detail about this finding, I further explored the raw corpus. I found that *kwanha/VV* appeared four times in total across the corpus. Moreover, *kwanha/VV* only appeared when *-ey* was used with the function THM (i.e., theme), as shown in Figure 9.1.

Index	Function	Sentence
128	THM	굴절_01/NNG 이상_05/NNG 시력_01/NNG 에/JKB_THM 관하/VV ㄴ/ETM 기능_03/NNG 적/XSN 이상_12/NNG
135	THM	국조오례의/NNP 조선/NNP 시대_02/NNG 의/JKG 오례/NNG 에/JKB_THM 관하/VV ㄴ/ETM 책_01/NNG
177	THM	첫째/NR 한_05/NNG 에/JKB_THM 관하/VV ㄴ/ETM 것/NNB
427	THM	스넬/NNP 의/JKG 법칙/NNG 빛/NNG 의/JKG 굴절_01/NNG 에/JKB_THM 관하/VV ㄴ/ETM 법칙/NNG

Figure 9.1: Example of *kwanha/VV* in the raw corpus

Note. Abbreviation: ETM = Adnominal Changing Ending; JKB = adverbial postposition; JKG = Genitive Case Marker; MAG = general adverb; NNG = common noun; NNP = Proper Noun; NP = pronoun; NR = Numeral; THM = Theme; VV = verb; XSN = A Noun Derivational Suffix

Second, there was a word group that had a strong connection in language use. These words appeared frequently regardless of which functions a postposition was used. For instance, when -*ey* was used as the function of THM (i.e., theme), there was a strong relationship between it and *kukes/NP* based on the cosine similarity of 0.999 as shown in Figure 5.19. *kukes/NP* appeared 47 times in the total corpus. Unlike *kwanha/VV*, *kukes/NP* appeared not only when -*ey* was used with the function THM (i.e., theme). *kukes/NP* appeared when -*ey* was used as different functions (e.g., AGT: 3; CRT: 16; EFF: 3; FNS: 2; GOL: 2; INS: 4; LOC: 7; THM: 10).

9.1.4 Overall discussion of two word-level embedding models: PPMI-SVD and SGNS

In the above sections, I described the interpretations of the findings with respect to the classification models and visualization systems, starting from the hypotheses on three research questions. Through the investigation of these hypotheses, I found three major findings.

First, the fewer functions the postposition had, the higher the classification accuracy was obtained (see Section 9.1.1). Second, the PPMI-SVD model obtained high classification accuracy at a larger window size (see Section 9.1.2). Third, the cluster was not changed much by the environments of word-level embedding (see Section 9.1.3).

Despite the implications of previous word-level embedding models, the two models have remained limited and have showed unsatisfactory classification performances (e.g., PPMI-SVD: *-ey* (0.534), *-eyse* (0.773), *-(u)lo* (0.567); SGNS: *-ey* (0.204), *-eyse* (0.693), *-(u)lo* (0.368)). Moreover, they performed well only when the target functions occurred very frequently in the data. This is because of the technical nature of word-level embedding, which converts a word into a single vector by using only its morphological information (e.g., Ethayarajh, 2019, Liu et al., 2019a).

To overcome these problems, I employed the Bidirectional Encoder Representations from Transformer (BERT) (Devlin et al., 2018) to classify the functions of the postpositions, which converts all of the words into different vectors considering contextual information. In the following sections, I present some discussion with respect to the findings of the BERT model.

9.2 Interpretations of sentence-level embedding model: BERT

The research questions with respect to the BERT model in Chapter 7 are restated as follows:

- Research question 1: How does the number of functions involving a postposition affect the model performance of BERT?
- Research question 2: How the asymmetric proportions of the functions in each postposition affect the model performance?
- Research question 3: How does the BERT model classify sentences for each postposition based on function as the epoch proceeds?

9.2.1 The number of functions in each postposition

Regarding the first research question, I made a hypothesis with respect to the number of functions as below:

- Hypothesis: The accuracy of the classification should be inversely proportionate to the number of functions of a postposition.

As described in section 9.1.1, the word-level embedding models showed that there was an inverse relation between the classification accuracy and the number of functions. In addition, I found that the average model performance was similar to the accuracy patterns of the functions that occupy a larger proportion of the total corpus size. Considering this, I predicted that the classification accuracy of the BERT model would be influenced by the number of functions of a postposition. This prediction was investigated in Chapter 8 by exploring the classification performance of the BERT model by each postposition.

I found that there was an inverse relationship between the classification accuracy and the number of functions that each postposition has. For instance, the average classification accuracy was 0.815 for *-ey*, 0.898 for *-eyse*, and 0.813 for *-(u)lo*. Considering the different number of functions (e.g., two for *-eyse*, six for *-(u)lo*, and eight for *-ey*), it can be interpreted that the number of functions affected the classification performance of the BERT model. However, unlike the word-level embedding models that showed large gaps of model performance between each postposition (e.g., PPMI-SVD: *-ey* (0.534), *-eyse* (0.773), *-(u)lo* (0.567); SGNS: *-ey* (0.204), *-eyse* (0.693), *-(u)lo* (0.368)), the BERT model was less affected.

9.2.2 The relationship between corpus size of each function and model performance

The second research question was expressed as a hypothesis regarding the relationship between the corpus size of each function and model performance as below:

- Hypothesis: The accuracy of the classification should vary depending on the corpus size of each function.

The results of two models showed that the classification accuracy is affected by the corpus size of the functions that account for a larger proportion of the total corpus size. I thus assumed that the model performance of BERT would be similar to the accuracy patterns of these functions. This assumption was investigated in Chapter 8 by conducting a correlation between the mean accuracy of the BERT model and of each function of the three postpositions.

I found that the BERT model was not particularly affected by the corpus size, which is contrary to the results of the word-level embedding models. For example, as shown in Table 8.5 (i.e., a correlation between BERT and each function for -*ey*), the mean accuracy of the BERT model was not similar to that of LOC (-0.218) and CRT (0.115). In addition, I found that the mean accuracy was highly correlated to that of THM (0.708) and FNS (0.733), which are the functions that account for a smaller portion of the total corpus size.

There are two reasons to support the fact that the BERT model performs better than the word-level embedding models for resolving word-level polysemy of Korean adverbial postpositions. First, it assigned each word to a vector on the basis of the context information, even if the form of the words is the same as each other. Second, it was able to recognize the functions of each postposition, with less influence of corpus size on model performance. This is due to the BERT model operating on the basis of the pre-trained model, which means that it had enough information on the attested language.

9.2.3 The relationship between the model performance and epoch

I made a hypothesis about the changes in the relationship between the model performance and the epoch (i.e., learning step) as follow:

- Hypothesis: The accuracy of the classification should be higher in larger epochs.

Previous studies have investigated various inquiries on language by using BERT. In these studies, they recommended setting smaller epoch sizes for better BERT training (e.g., Reimers and Gurevych, 2019, Reimers et al., 2019, Sun et al., 2019, Warstadt and Bowman, 2020). However, there was no specific scientific reason but rather a technical reason with respect to better implementation. To better understand why this is so, I investigated the relationship between the model performance and the epoch by developing a BERT-based visualization system. I predicted that the classification accuracy would improve as the epoch progressed. This is due to the previous studies that reported that the epoch size of BERT represents the learning step, and that the larger learning steps have better model performance (e.g., Reimers and Gurevych, 2019, Reimers et al., 2019, Sun et al., 2019, Warstadt and Bowman, 2020).

By examining the changes of sentence clusters by each epoch by using visualization, I found that the BERT model could recognize the functions of each postposition as the epoch (i.e., learning) progressed. Furthermore, this has revealed more details sub-described as two findings. First, the BERT model can identify functions at a satisfactory level, even if they are relatively infrequent, by way of sufficient epochs. For instance, as shown in Figure 8.19

(e.g., the distributional map for *-(u)lo* in epoch 12), EFF emerged as a distinguished cluster, though it is a function with low-frequency in the total corpus. Second, the BERT model is subjective to the extremely low-frequent items and/or semantic closeness between the items. For example, as shown in Figure 8.20 (e.g., the distributional map for *-(u)lo* in epoch 46), LOC could not form a designated cluster in the end. Many of the LOC instances (11 out of 15) belonged to the DIR group. This is due to the semantic closeness between DIR and LOC, which means that they are often difficult to distinguish one from the other.

9.2.4 Overall discussion of sentence-level embedding model: BERT

In this dissertation, I employed the BERT model in order to improve the limitations of traditional word-level embedding models. From this, I had three major findings. First, the BERT model is affected by the number of functions that the postposition has. Second, the classification accuracy of was less affected by the corpus size. Third, when the epoch progressed, it could recognize more functions. Moreover, I found that the BERT model showed better classification performance than previous studies, including the traditional word-level embedding models that I investigated for the same task in this dissertation.

In the following chapter, I will further describe the contribution of this dissertation for the task to classify the polysemous Korean adverbial postpositions *-ey*, *-eyse*, and *-(u)lo*.

Chapter 10

Conclusion

In this dissertation, I investigated computational accounts for interpreting polysemy of the three representative Korean adverbial postpositions: -*ey*, -*eyse*, and -*(u)lo*. In addition, I addressed the possible ways and limitations in applying computational methods to language data involving multiple form-function pairings in Korean.

10.1 Summary of major findings

I conducted this study in the following three steps: first of all, I identified the specific the functions of each postposition based on the classification system developed by the Sejong project and on previous studies on Korean adverbial postpositions. -*ey* has eight major functions, with *'location'* and *'goal'* occupying the majority of the occurrences. -*eyse* has two functions, *'source*' and *'location'*, and is used overwhelmingly more frequently than the others. -*(u)lo* has six functions, with the top three functions, *'final state*', *'instrumental'*, and *'directional'*, occupying more than 80 percent of the entire use.

Next, I made the classification/visualization models, one by using a combination of PPMI and SVD as a count-based model and the other by using SGNS as a prediction-based model with the basis of similarity-based estimation. In general, I found that, if a postposition had fewer functions, the classification model obtained a high classification accuracy. The PPMI-SVD model achieved high classification accuracy when the window size was large, indicating that for the best classification performance, it used the semantic characteristics of the large window sizes more than the syntactic ones. In contrast, the SGNS model showed low classification accuracy regardless of the window sizes. Moreover, I found that the PPMI-SVD model was affected by the corpus size more than the SGNS model was. This is because the PPMI-SVD model is sensitive to the token frequency of words, whereas the SGNS model is sensitive to the type frequency of words. Through the visualization, I found that (i) the clusters did not change considerably by the environments of word-level embeddings, and (ii) there were the two types of co-occurring words: the words that appeared frequently in the total corpus and the words that only appeared when the postposition used as a specific function.

Despite these findings, there were two issues with the performance of the two word-level embedding models. First, the models appeared to perform well only when the target functions occurred very frequently in the data, which means that the accuracy seemed to be affected by the corpus size of each function. Second, the model performance of my models was lower than of the models proposed in the previous studies for the same classification task. This is because of the technical nature of word-level embedding—they are *static* in that a single vector is assigned to each word.

Finally, to overcome these limitations, I applied BERT to transform all of the words into different vectors, while considering their contextual information for the same classification task. For the model training, I set the parameters such as *batch size* (32), *epoch* (50), *seed* (42), *epsilon* (0.00000008), and *learning rate* (0.00002). Then, I fine-tuned the pre-trained model (i.e., KoBERT; Jeon et al., 2019) by using a training set for each postposition according to 50 epochs (i.e., learning). For the classification task, the BERT model obtained high classification accuracy: 0.815 for *-ey*, 0.898 for *-eyse*, 0.813 for *-(u)lo*. This was higher than the model performance of previous studies and of the word-level embedding models I used. In addition, I found that the BERT model was not particularly affected by the corpus size of each function, which was contrary to the result shown by the word-level embedding models. The reasons for this are that the BERT model assigned each word a vector based on the contextual information and operated on the basis of the pre-trained model with a large amount of corpus data. Through the visualization, I found that the BERT model could recognize the functions of each postposition as the epoch (i.e., learning) progressed, even if the functions occupied a smaller portion of the total corpus size. This was also contrary to the results from the traditional word-level embedding models, which are known to be affected considerably by the size of corpus. This indicates that the BERT model can identify relatively infrequent functions at a satisfactory level by way of sufficient epochs. Moreover, this suggests that it is able to simulate how humans interpret the polysemy involving Korean adverbial postpositions more appropriately than the word-level embedding models.

10.2 Limitations and future works

Despite these findings, this dissertation remains limited. I acknowledge some limitations of this project as follows.

First of all, I focused only on three different Korean adverbial postpositions that have word-level polysemy. However, according to the statistical description in the Standard-Korean dictionary (1999), there are 361 postpositions in Korean. With this in mind, the findings from the three postpositions focused on in this dissertation are not enough to generalize all the postpositions in Korean. Even if these three are frequently used in Korean, the findings of this study should be further verified by more postpositions. Therefore, in the future, I would improve this study to cover more postpositions that have similar degrees of polysemy as *-ey*, *-eyse*, and *-(u)lo*.

Second, to determine the number and types of functions of each postposition, I used only the definition in the Sejong Electronic Dictionary. However, the specific functions of each postposition used in the Sejong corpus are somewhat different from those found in previous studies on Korean linguistics. For instance, some studies considered *'time'* as a function of *-ey*, but the Sejong corpus included it in the *'criteria'* function, and it also was not included as a function of *-ey* in the Sejong Electronic Dictionary. As shown in Figure 10.1, I confirmed that CRT contains *'time'* in the Sejong Electronic Dictionary.

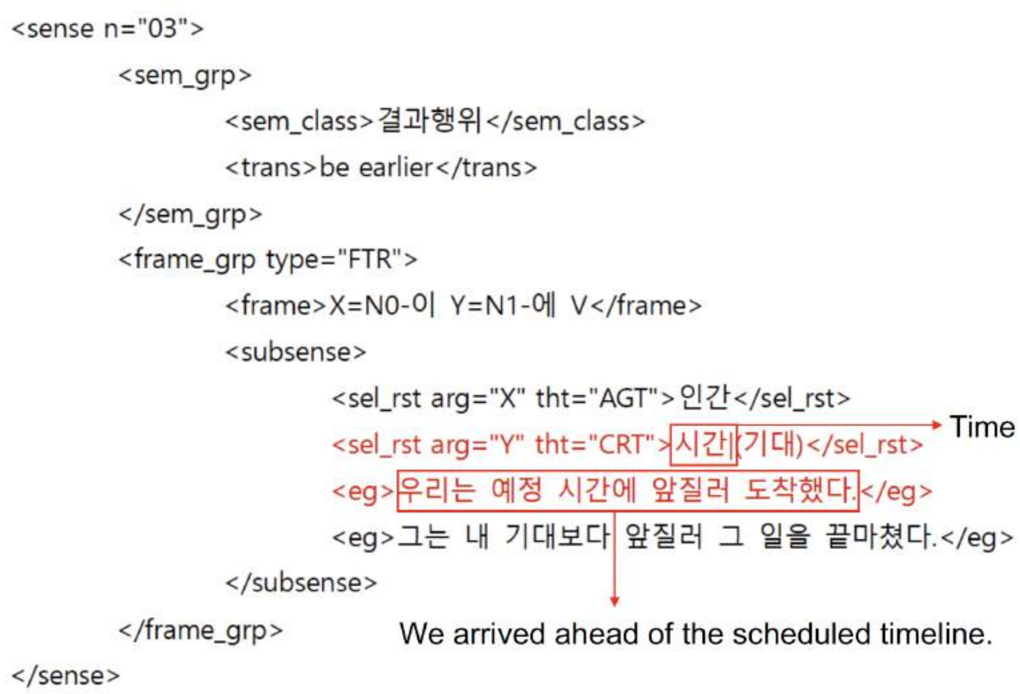

Figure 10.1: Example of an error extracted from the file *V-aphciluta* in the Sejong Electronic Dictionary

In addition, the number of functions of *-ey* varied across different studies, although the Sejong corpus set the number as eight. Therefore, in the future, I would refer to more previous studies to determine the number and types of functions for each postposition in order to include more details by further subdividing the functions.

Third, I employed three embedding models (i.e., PPMI-SVD, SGNS, and BERT) for the classification task in this dissertation. However, considering that other contextualized word-embedding models were released after BERT, such as the Generation Pre-trained Transformer 3 (GPT-3; Brown et al., 2020) or the Robustly Optimized BERT Pretraining Approach (RoBERTa; Liu et al., 2019b), it is necessary to use them in order to ensure methodological generalizability and attest to the recent computational methods in Korean, a language typologically different from the major Indo-European languages.

10.3 Implications of findings

Despite the limitations, this dissertation has two major implications.

First, it provides the possible ways and limitations of applying three different embedding models for the task of identifying the intended function of Korean adverbial postpositions. There were many previous studies on interpreting word-level polysemy of major Indo-European languages by employing existing word-level embedding models (Positive Pointwise Mutual Information and Singular Value Decomposition; Skip-Gram and Negative Sampling) or sentence-level embedding model (Bidirectional Encoder Representations from Transformers (BERT)) under the scheme of Distributional Semantic Modeling. Despite a good amount of research on English for this issue, very few studies have investigated polysemy interpretation of language typologically different from English. Thus, I turned my attention to Korean, an under-studied language in this regard, with a special focus on the relation between the three different embedding models and the word-level polysemy of adverbial postpositions. As a result, I found that the sentence-level embedding model, which assigned different vectors to all of the words, performed better in interpreting the functions of postposition than the word-level embedding models, which assigned a single vector to each word, regardless of context. Considering that previous research is skewed toward major Indo-European languages such as English, the attempt of this dissertation has a contribution to the methodological generalizability by applying computational account to a lesser-studied language such as Korean.

Second, this dissertation proposes two interactive visualization systems that help to identify the relationships between words or sentences and to show changes of the clusters by the environments (i.e., models, postpositions, window sizes, and epochs). Although the word-level and sentence-level embedding models have frequently been used in recent studies, it is very hard to understand how these embedding models interpret word-level polysemy. The first visualization system aimed to explore word-level embedding results. This makes us see the clusters of the postpositions and their co-occurring words in order to understand how the relationships of words changed based on the functions of each postposition. I found that there were two different types of co-occurring words related to each function: (i) words that appeared only when the postposition was used as a particular function and (ii) words that had a strong connection in language use regardless of which functions a postposition was used. However, the clusters of words were not changed much by the environments of word-level embedding. The second visualization system was developed to show how the sentence-level embedding model (i.e., BERT) recognizes the polysemy involving the postpositions. I found that the BERT model could identify the intended functions of postposition when the epoch progressed, with less sensitivity to data size. In addition, if the functions of each postposition have a semantic closeness to each other, the low-frequency function is contained within the high-frequency function. Considering that the visualization system could help understand the computational outcomes more easily and clearly through an intuitive (and yet, informative) display of language data, the attempt of this dissertation has a particular contribution for future studies.

References

Standard Korean Dictionary. National Institute of Korean Language, Seoul, South Korea, 1999.

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: A system for large-scale machine learning. *CoRR*, abs/1605.08695, 2016. URL http://arxiv.org/abs/1605.08695.

Eneko Agirre and Oier. Lopez de Lacalle. On robustness and domain adaptation using svd for word sense disambiguation. In *In Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*.

Myung-chel Ahn. The meaning of locative postposition '*-ey*'. *kwanak emwun yenkwu*, 7:245–268, 1983.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. Many languages, one parser. *Transactions of the Association for*

Computational Linguistics, 4:431–444, 2016. doi: 10.1162/tacl_a_00109. URL https://www.aclweb.org/anthology/Q16-1031.

Alain Auger and Caroline Barrière. Pattern-based approaches to semantic relation extraction: A state-of-the-art. *Terminology: international journal of theoretical and applied issues in specialized communication*, 14(1):1–19, 2008.

Jangseong Bae and Changki Lee. End-to-end learning of korean semantic role labeling using bidirectional lstm crf. In *Proc. of the KIISE Korea Computer Congress*, pages 566–568, 2015.

Jangseong Bae, Junho Oh, Hyunsun Hwang, and Changki Lee. Extending korean propbank for korean semantic role labeling and applying domain adaptation technique. *Korean Information Processing Society*, pages 44–47, 2014.

Jangseong Bae, Changki Lee, and Soojong Lim. Korean semantic role labeling using deep learning. *Korean Information Science Society*, 6:690–692, 2015.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014. URL http://arxiv.org/abs/1409.0473. cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. *52nd Annual Meeting of the Association* *for Computational Linguistics, ACL 2014 - Proceedings of the Conference*, 1:238–247, 2014. URL https://www.researchgate.net/publication/270877599_Don%27t_count_predict_A_systematic_comparison_of_context-counting_vs_context-predicting_semantic_vectors.

Yoshua Bengio, Patrick Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Jason Brownlee. Repeated k-fold cross-validation for model evaluation in python, 2020. URL https://machinelearningmastery.com/repeated-k-fold-cross-validation-with-python/.

John A. Bullinaria and J. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, 39:510–526, 2007.

John A. Bullinaria and Joseph P. Levy. Extracting semantic representations from word co-occurrence statistics: stop-lists, stemming, and svd. *Be-*

havior Research Methods, 44(3):890–907, Sep 2012. ISSN 1554-3528. doi: 10.3758/s13428-011-0183-8. URL https://doi.org/10.3758/s13428-011-0183-8.

Guibin Chen, Deheng Ye, Zhenchang Xing, Jieshan Chen, and Erik Cambria. Ensemble application of convolutional and recurrent neural networks for multi-label text categorization. In *IJCNN*, pages 2377–2383. IEEE, 2017. ISBN 978-1-5090-6182-2. URL http://dblp.uni-trier.de/db/conf/ijcnn/ijcnn2017.html#ChenYXCC17.

Jeong-mi Cho and Gil-cheng Kim. A study on the resolving of the ambiguity while interpretation of meaning in korean. *The Korean Institute of Information Scientists and Engineers*, 14(7):71–83, 1996.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012. URL https://www.aclweb.org/anthology/W14-4012.

Francois Chollet. Keras (github), 2015. URL https://github.com/fchollet/keras.

Miho Choo and Hye-young Kwak. *Using Korean*. Cambridge University Press, New York, NY, 2008.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. URL http://arxiv.org/abs/1412.3555. cite arxiv:1412.3555Comment: Presented in NIPS 2014 Deep Learning and Representation Learning Workshop.

Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1989.

Andy Clark. Embodied prediction. In *In T. K. Metzinger J. M. Windt (Eds.) Open MIND: 7(T). Frankfurt am Main: MIND Group*, 2015.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of bert's attention. *CoRR*, abs/1906.04341, 2019. URL http://arxiv.org/abs/1906.04341.

Andy Coenen, Emily Reif, Ann Yuan, Been Kim, Adam Pearce, Fernanda Viégas, and Martin Wattenberg. Visualizing and measuring the geometry of bert. 2019. URL http://arxiv.org/abs/1906.02715. cite arxiv:1906.02715Comment: 8 pages, 5 figures.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390177. URL https://doi.org/10.1145/1390156.1390177.

Ido Dagan, Shaul Marcus, and Shaul Markovitch. Contextual word similarity and estimation from sparse data. *Comput. Speech Lang.*, 9(2):123–152,

1995. URL http://dblp.uni-trier.de/db/journals/csl/csl9.html#DaganMM95.

Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *NIPS*, pages 3079–3087, 2015. URL http://dblp.uni-trier.de/db/conf/nips/nips2015.html#DaiL15.

Hoa Trang Dang. Overview of DUC 2005. In *Proceedings of the 2005 Document Understanding Workshop*, 2005. URL http://www-nlpir.nist.gov/projects/duc/pubs.html.

Mark Davies. The corpus of contemporary american english (coca): 400+ million words, 2008. URL http://corpus.byu.edu/coca.

Mark Davies. The wikipedia corpus: 4.6 million articles, 1.9 billion words, 2015. URL https://www.english-corpora.org/wiki/.

Guillaume Desagulier. Visualizing distances in a set of near synonyms: rather, quite, fairly, and pretty. In *In Corpus Methods for Semantics: Quantitative Studies in Polysemy and Synonymy*, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Cícero Nogueira dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1818–1826. JMLR.org, 2014. URL http://dblp.uni-trier.de/db/conf/icml/icml2014.html#SantosZ14.

C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

Katrin Erk. Vector space models of word meaning and phrase meaning: A survey. *Lang. Linguistics Compass*, 6(10):635–653, 2012. URL http://dblp.uni-trier.de/db/journals/llc/llc6.html#Erk12.

Kawin Ethayarajh. How contextual are contextualized word representations? comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 55–65, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1006. URL https://www.aclweb.org/anthology/D19-1006.

R. M. Fano. *Transmission of Information: A Statistical Theory of Communications*. MIT Press, 1961.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. *ACM Trans. Inf. Syst.*, 20(1):116–131, January 2002. ISSN 1046-8188. doi: 10.1145/503104.503110. URL https://doi.org/10.1145/503104.503110.

J. Firth. *A synopsis of linguistic theory 1930-1955*. Studies in Linguistic Analysis, Philological. Longman, 1957.

Marcos Garcia and Pablo Gamallo. An exploration of the linguistic knowledge for semantic relation extraction in spanish. In *Proceedings of Joint Workshop FAM-LbR/KRAQ'11 at IJCAI 2011.*, 2011.

Alfio Massimiliano Gliozzo, Claudio Giuliano, and Carlo Strapparava. Domain kernels for word sense disambiguation. In Kevin Knight, Hwee Tou Ng, and Kemal Oflazer, editors, *ACL*, pages 403–410. The Association for Computer Linguistics, 2005. URL http://dblp.uni-trier.de/db/conf/acl/acl2005.html#GliozzoGS05.

Stefan Th Gries. *Quantitative Methods in Linguistics*. International Encyclopedia of the Social and Behavioral Sciences. Amsterdam: Elsevier, 2015.

Sebastian Guggisberg. Fine tuning bert for text classification with farm, 2020.

Ben Hachey, Gabriel Murray, and David Reitter. Dimensionality reduction aids term co-occurrence based multi-document summarization. In *Proceedings of the COLING-ACL Workshop Task-Focused Summarization and Question Answering 2006*, pages 1–7, Sydney, Australia, 2006.

Michael Hahsler, Matthew Piekenbrock, and Derek Doran. Dbscan: Fast density-based clustering with r, 2019. URL https://cran.r-project.org/web/packages/dbscan/vignettes/dbscan.pdf.

Lushan Han, Tim Finin, Paul McNamee, Anupam Joshi, and Yelena

Yesha. Improving word similarity by augmenting pmi with estimates of word polysemy. *IEEE Trans. Knowl. Data Eng.*, 25(6):1307–1322, 2013. URL http://dblp.uni-trier.de/db/journals/tkde/tkde25.html#HanFMJY13.

Zellig Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954. doi: 10.1007/978-94-009-8467-7_1. URL https://link.springer.com/chapter/10.1007/978-94-009-8467-7_1.

Amir Hazem and Emmanuel Morin. Word co-occurrence counts prediction for bilingual terminology extraction from comparable corpora. In *IJCNLP*, pages 1392–1400. Asian Federation of Natural Language Processing / ACL, 2013. ISBN 978-4-9907348-0-0. URL http://dblp.uni-trier.de/db/conf/ijcnlp/ijcnlp2013.html#HazemM13.

Enrique Henestroza Anguiano and Pascal Denis. FreDist: Automatic construction of distributional thesauri for French. In *TALN - 18ème conférence sur le traitement automatique des langues naturelles*, pages 119–124, Montpellier, France, France, June 2011. URL https://hal.archives-ouvertes.fr/hal-00602004.

Minsuk Heo. Rnn basic (vanilla recurrent neural network), 2018. URL https://youtu.be/2AuMgtw-z6s?list=PLVNY1HnUlO27T2H_KspAKembHX_ru0Ha1.

Felix Hill, Roi Reichart, and Anna Korhonen. SimLex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, December 2014. doi: 10.1162/COLI_a_00237. URL https://www.aclweb.org/anthology/J15-4004.

Martin Hilpert. Change in modal meanings. *Constructions and Frames*, 8(1): 66–85, 2016.

Yunpyo Hong. On the case of directionality. *The Society of Korean Linguistics*, 6:111–132, 1978.

Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, 1933.

Jason C Hung and Che-Yu Yang. Word sense disambiguation using word ontology and concept distribution. *Journal of the Chinese institute of engineers*, 32(2):153–168, 2009.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora dataset release: Question pairs, 2017. URL https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs.

Heewon Jeon, Donggeon Lee, and Jangwon Park. Korean bert pre-trained cased (kobert), 2019. URL https://github.com/SKTBrain/KoBERT.

Byong-cheol Jeong. An integrated study on the particle '-*ey*' based on the simulation model. *The Linguistic Science Society*, 55:275–304, 2010.

Byeong-cheol Jo, Mi-ran Seok, Hye-jeong Song, Chan-young Park, Jongdae Kim, and Yu-seop Kim. Expansion of feature information for korean semantic role labeling. *Advanced Science and Technology Letters*, 120:798–802, 2015.

Heejung Jung. The meaning of postposition focus on '-*ey*'. *The Society of Korean Linguistics*, 17:153–175, 1988.

Daniel Jurafsky and James. H. Martin. *Speech and language processing: An Introduction to Natural Language Processing*. Computational Linguistics, and Speech Recognition, Prentice-Hall, 2019.

Sin-jae Kang and Jung-hye Park. Rule construction for determination of thematic roles by using large corpora and computational dictionaries. *Korean Information Processing Society*, 10(2):219–228, 2003.

Jun'ichi Kazama, Stijn De Saeger, Kow Kuroda, Masaki Murata, and Kentaro Torisawa. A bayesian method for robust estimation of distributional similarities. In Jan Hajic, Sandra Carberry, and Stephen Clark, editors, *ACL*, pages 247–256. The Association for Computer Linguistics, 2010. ISBN 978-1-932432-67-1. URL http://dblp.uni-trier.de/db/conf/acl/acl2010.html#KazamaSKMT10.

Byoung-soo Kim, Yong-hun Lee, Seung-hoon Na, Jun-gi Kim, and Jong-hyeok Lee. Bootstrapping for semantic role assignment of korean case marker. *Korea Information Science Society*, pages 4–6, 2006.

Byoung-soo Kim, Yong-hun Lee, and Jong-hyeok Lee. Unsupervised semantic role labeling for korean adverbial case. *Journal of KIISE: Software and Applications*, 34(2):32–39, 2007.

Wan-su Kim and Cheol-young Ock. Korean semantic role labeling using case frame and frequency. *The Korean Institute of Information Scientists and Engineers*, 6:651–653, 2015.

Wan-su Kim and Cheol-young Ock. Korean semantic role labeling using case frame dictionary and subcategorization. *The Korean Institute of Information Scientists and Engineers*, 43(12):1376–1384, 2016.

Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1998.

J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

Changki Lee, Soojong Lim, and Hyunki Kim. Korean semantic role labeling using structured svm. *The Korean Institute of Information Scientists and Engineers*, 42(2):220–226, 2015.

Kee-dong Lee. The meaning of the postpositions -*ey* and -*eyse*. *The Korean Language Society*, 173:9–34, 1981.

Namsun Lee. In the form '-*ey*' and the material '-*eyse*'. *kwanak emwun yenkwu*, 8:321–355, 1983.

Joseph P. Levy, John A. Bullinaria, and Malti Patel. Explorations in the derivation of word co-occurrence statistics. *South Pacific Journal of Psychology*, 10(1):99–111, 1999. doi: 10.1017/S0257543400001061.

Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P14-2050.

Omer Levy, Yoav Goldberg, and Ido Dagan. Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, December 2015. ISSN 2307-387X. doi: 10.1162/tacl_a_00134. URL https://www.mitpressjournals.org/doi/abs/10.1162/tacl_a_00134.

Jiwei Li, Xinlei Chen, Eduard H. Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *CoRR*, abs/1506.01066, 2015. URL http://dblp.uni-trier.de/db/journals/corr/corr1506.html#LiCHJ15.

Yongjie Lin, Yi Chern Tan, and Robert Frank. Open sesame: Getting inside bert's linguistic knowledge. *CoRR*, abs/1906.01698, 2019. URL http://arxiv.org/abs/1906.01698.

Pierre Lison and Andrei Kutuzov. Redefining context windows for word embedding models: An experimental study. *CoRR*, abs/1704.05781, 2017. URL http://dblp.uni-trier.de/db/journals/corr/corr1704.html#LisonK17.

Pierre Lison and Jörg Tiedemann. OpenSubtitles2016: Extracting large parallel corpora from movie and TV subtitles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 923–929, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL https://www.aclweb.org/anthology/L16-1147.

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. Linguistic knowledge and transferability of contextual representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1073–1094, Minneapolis, Minnesota, June 2019a. Association for Computational Linguistics. doi: 10.18653/v1/N19-1112. URL https://www.aclweb.org/anthology/N19-1112.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019b.

Daniel Loureiro and Alípio Jorge. Language modelling makes sense: Propagating representations through WordNet for full-coverage word sense disambiguation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5682–5691, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1569. URL https://www.aclweb.org/anthology/P19-1569.

Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008. ISSN ISSN 1533-7928. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

Gyeongheum Maeng. Cognitive semantics of korean postposition '-*ey*'. *The Journal of Korean Studies*, 41:325–366, 2016.

Irina Matveeva, Gina-Anne Levow, Ayman Farahat, and Christiaan Royer. Term represetation with generalized latent semantic analysis. In *In Proceedings of the 2005 Conference on Recent Advances in Natural Language Processing*, 2005.

Chris McCormick. Bert fine-tuning tutorial with pytorch, 2019. URL http://mccormickml.com/2019/07/22/BERT-fine-tuning/.

Oren Melamud, David McClosky, Siddharth Patwardhan, and Mohit Bansal. The role of context types and dimensionality in learning word embeddings. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *HLT-NAACL*, pages 1030–1040. The Association for Computational Linguistics, 2016. ISBN 978-1-941643-91-4. URL http://dblp.uni-trier.de/db/conf/naacl/naacl2016.html#MelamudMPB16.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, volume 2, page 3, 2010.

Tomas Mikolov, Anoop Deoras, Daniel Povey, Lukás Burget, and Jan Cernocký. Strategies for training large scale neural network language models. In David Nahamoo and Michael Picheny, editors, *ASRU*, pages 196–201. IEEE, 2011. ISBN 978-1-4673-0365-1. URL http://dblp.uni-trier.de/db/conf/asru/asru2011.html#MikolovDPBC11.

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann

LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013a. URL http://arxiv.org/abs/1301.3781.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. 2013b.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013c.

Seongmin Mun and Kyungwon Lee. Data analysis by integrating statistics and visualization: Visual verification for the prediction model. *Journal of Society of Design Convergence*, 15(6):195–214, 2016.

Seongmin Mun and Gyu-Ho Shin. Context window and polysemy interpretation: A case of korean adverbial postposition *-(u)lo*. In *IMPRS Conference 2020: Interdisciplinary Approaches to the Language Sciences, Max Planck Institute for Psycholinguistics*, 2020.

Seongmin Mun, Hyunwoo Han, Hyoji Ha, and Kyungwon Lee. A visual analysis on factors affecting repurchase intention in social commerce. *Journal of Society of Design Convergence*, 13(6):139–152, 2014.

Seongmin Mun, Gyeongcheol Choi, Sangkuk Lee, and Kyungwon Lee. Visual analysis for voting relationships in joseon dynasty, korea. pages 111–118, 2017.

Ki-sim Nam. The use of the korean postposition: focus on '*-ey*' and '*-(u)lo*'. *sekwang hakswul calyosa*, 1993.

Yoshiki Niwa and Yoshihiko Nitta. Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *Proceedings of the 15th International Conference On Computational Linguistics*, pages 304–309, Kyoto, Japan, 1994.

Ankur P. Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *CoRR*, abs/1606.01933, 2016. URL http://dblp.uni-trier.de/db/journals/corr/corr1606.html#ParikhT0U16.

Jeong-woon Park. A polysemy network of the korean instrumental case. *Korean Journal of Linguistics*, 24(3):405–425, 1999.

Seong-bae Park and Yung-taek Kim. Semantic role determination of korean adverbial postpositions. *Korea Information Science Society*, 4:339–401, 1998.

Seong-bae Park, Byoung-tak Zhang, and Yungtaek Kim. Decision tree based disambiguation of semantic roles for korean adverbial postpositions in korean-english machine translation. *The Korean Institute of Information Scientists and Engineers*, 27(6):668–677, 2000.

Tae-ho Park and Jeong-won Cha. Korean semantic role labeling using word sense. *The Korean Institute of Information Scientists and Engineers*, 6:590–592, 2017.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword fifth edition (ldc2011t07), 2011. URL https://catalog.ldc.upenn.edu/LDC2011T07.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL http://arxiv.org/abs/1912.01703.

Ted Pedersen, Amruta Purandare, and Anagha Kulkarni. Name discrimination by clustering similar contexts. In Alexander F. Gelbukh, editor, *CICLing*, volume 3406 of *Lecture Notes in Computer Science*, pages 226–237. Springer, 2005. ISBN 3-540-24523-5. URL http://dblp.uni-trier.de/db/conf/cicling/cicling2005.html#PedersenPK05.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *CoRR*, abs/1201.0490, 2012. URL http://arxiv.org/abs/1201.0490.

Yves Peirsman, Kris Heylen, and Dirk Speelman. Finding semantically related words in dutch: co-occurrences versus syntactic contexts. In *Proceedings* *of the 2007 Workshop on Contextual Information in Semantic Space Models*, pages 9–16, 2007.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018. URL http://arxiv.org/abs/1802.05365. cite arxiv:1802.05365Comment: NAACL 2018. Originally posted to openreview 27 Oct 2017. v2 updated for NAACL camera ready.

Soujanya Poria, Erik Cambria, Devamanyu Hazarika, Navonil Majumder, Amir Zadeh, and Louis-Philippe Morency. Context-dependent sentiment analysis in user-generated videos. In Regina Barzilay and Min-Yen Kan, editors, *ACL (1)*, pages 873–883. Association for Computational Linguistics, 2017. ISBN 978-1-945626-75-3. URL http://dblp.uni-trier.de/db/conf/acl/acl2017-1.html#PoriaCHMZM17.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018. URL https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019. URL http://arxiv.org/abs/1908.10084.

Nils Reimers, Benjamin Schiller, Tilman Beck, Johannes Daxenberger, Christian Stab, and Iryna Gurevych. Classification and clustering of arguments with contextualized word embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 567–578, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1054. URL https://www.aclweb.org/anthology/P19-1054.

Martin Riedl and Chris Biemann. There's no 'count or predict' but task-based selection for distributional models. In Claire Gardent and Christian Retoré, editors, *IWCS(2)*. The Association for Computer Linguistics, 2017. URL http://dblp.uni-trier.de/db/conf/iwcs/iwcs2017-2.html#RiedlB17.

H.C. Romersburg. *Cluster Analysis for Research*. Lifetime Learning Publications, Belmont, California, 1984.

G. Salton. *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice Hall, 1971.

Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, jun 1998. URL http://dx.doi.org/10.1023/A:1009745219419.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019. URL http://dblp.uni-trier.de/db/journals/corr/corr1910.html#abs-1910-01108.

Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *ICASSP*, pages 5149–5152. IEEE, 2012. ISBN 978-1-4673-0046-9. URL http://dblp.uni-trier.de/db/conf/icassp/icassp2012.html#SchusterN12.

Hinrich Schütze. Dimensions of meaning. In *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 787–796, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press. ISBN 0-8186-2630-5. URL http://portal.acm.org/citation.cfm?id=148132&dl=ACM&coll=GUIDE.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *ACL (1)*. The Association for Computer Linguistics, 2016. ISBN 978-1-945626-00-5. URL http://dblp.uni-trier.de/db/conf/acl/acl2016-1.html#SennrichHB16a.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2016. URL http://arxiv.org/abs/1611.01603. cite arxiv:1611.01603Comment: Published as a conference paper at ICLR 2017.

Yeolwon Seong. Korean parsing using sejong dictionary. *The Korean Institute of Information Scientists and Engineers*, pages 261–268, 2007.

Hyo-pil Shin. The 21st sejong project : with a focus on selk (sejong electronic lexicon of korean) and the knc (korean national corpus). In *In The 3rd International Joint Conference on Natural Langauge Processing*, 2008.

Myung-chul Shin, Yong-hun Lee, Mi-young Kim, You-jin Chung, and Jong-hyeok Lee. Semantic role assignment for korean adverbial case using sejong electronic dictionary. *Korea Information Science Society*, pages 120–126, 2005.

R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 01 1973. ISSN 0010-4620. doi: 10.1093/comjnl/16.1.30. URL https://doi.org/10.1093/comjnl/16.1.30.

Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In Lise Getoor and Tobias Scheffer, editors, *ICML*, pages 129–136. Omnipress, 2011. URL http://dblp.uni-trier.de/db/conf/icml/icml2011.html#SocherLNM11.

Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

Ho-Min Sohn. *The korean language*. Cambridge University Press, Cambridge, UK, 1999.

Dae-heon Song. A study on the adverbial case particles of '-*ey*' and '-*eyse*' for korean language education. *The Association of Korean Education*, 101: 457–484, 2014.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583, 2019. URL http://arxiv.org/abs/1905.05583.

Martin Sundermeyer, Ilya Oparin, Jean-Luc Gauvain, B. Freiberg, Ralf Schlüter, and Hermann Ney. Comparison of feedforward and recurrent neural network language models. In *ICASSP*, pages 8430–8434. IEEE, 2013. URL http://dblp.uni-trier.de/db/conf/icassp/icassp2013.html#SundermeyerOGFSN13.

Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton, editors, *EMNLP*, pages 1422–1432. The Association for Computational Linguistics, 2015. ISBN 978-1-941643-32-7. URL http://dblp.uni-trier.de/db/conf/emnlp/emnlp2015.html#TangQL15.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR*, abs/1903.12136, 2019. URL http://arxiv.org/abs/1903.12136.

W.S. Torgerson. Multidimensional scaling i: Theory and method. *Psychometrika*, 17:401—-419, 1952.

Peter D. Turney. A uniform approach to analogies, synonyms, antonyms, and associations. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 905–912, Manchester, UK, 2008.

Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *CoRR*, abs/1003.1141, 2010. URL http://arxiv.org/abs/1003.1141.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, page 5998–6008. Curran Associates, Inc., 2017. URL https://papers.nips.cc/paper/7181-attention-is-all-you-need.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. Grammar as a foreign language. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *NIPS*, pages 2773–2781, 2015. URL http://dblp.uni-trier.de/db/conf/nips/nips2015.html#VinyalsKKPSH15.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL https://www.aclweb.org/anthology/W18-5446.

Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based lstm for aspect-level sentiment classification. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *EMNLP*, pages 606–615. The Association for Computational Linguistics, 2016. ISBN 978-1-945626-25-8. URL http://dblp.uni-trier.de/db/conf/emnlp/emnlp2016.html#WangHZZ16.

Alex Warstadt and Samuel R. Bowman. Can neural networks acquire a structural bias from raw linguistic data?, 2020.

Gregor Wiedemann, Steffen Remus, Avi Chawla, and Chris Biemann. Does BERT make any sense? interpretable word sense disambiguation with contextualized embeddings. *CoRR*, abs/1909.10430, 2019. URL http://arxiv.org/abs/1909.10430.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *CoRR*, abs/1704.05426, 2017. URL http://dblp.uni-trier.de/db/journals/corr/corr1704.html#WilliamsNB17.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019. URL http://arxiv.org/abs/1910.03771.

Yige Xu, Xipeng Qiu, Ligao Zhou, and Xuanjing Huang. Improving bert fine-tuning via self-ensemble and self-distillation, 2020.

Dani Yogatama, Manaal Faruqui, Chris Dyer, and Noah A. Smith. Learning word representations with hierarchical sparse coding. *CoRR*, abs/1406.2035, 2014. URL http://dblp.uni-trier.de/db/journals/corr/corr1406.html#YogatamaFDS14.

Maayan Zhitomirsky-Geffet and Ido Dagan. Bootstrapping distributional feature vector quality. *Computational Linguistics*, 35(3):435–461, 2009. URL http://dblp.uni-trier.de/db/journals/coling/coling35.html#Zhitomirsky-GeffetD09.

Appendix A

Algorithms of this dissertation

The following Figures (A.1 - A.3) are the algorithms that I used in this dissertation.

Input: Methods (m_i), Kind of Postpositions (p_j), Number of folds (f_k), Range of context window (w_q)

Output: Word-level embedding in 500 dimensions (e_{500d})

Process;

```
foreach method in range(0, length(m_i))
    foreach postposition in range(0, length(p_j))
        foreach fold in range(0, length(f_k))
            (1) load DataFile by each postposition
            (2) generate Vocabulary
            if method == PPMI and SVD do:
                foreach window in range(0, length(w_q))
                    (3) apply PPMI formula into word-word matrix
                    (4) use scipy.linalg.svd for SVD in 500 dimensions
                    (5) save Word-level embedding in 500 dimensions (e_500d)
                end
            if method == SGNS do
                foreach window in range(0, length(w_q))
                    (3) use genism.word2vec for SGNS in 500 dimensions
                    (4) save Word-level embedding in 500 dimensions (e_500d)
                end
        end
    end
end
```

Figure A.1: Algorithm of the word-level embedding

Input: Methods (m_i), Kind of Postpositions (p_j), Number of folds (f_k), Range of context window (w_q)

Output: Cosine similarity scores between intended postposition and surrounding words (c_{600e})

Process;

foreach method **in** range(0, length(m_i))
    foreach postposition **in** range(0, length(p_j))
        foreach fold **in** range(0, length(f_k))
            foreach window **in** range(0, length(w_q))
                (1) load the word embedding result
                (2) calculate cosine similarity scores between intended postposition and surrounding words
                (3) load the test set
                (4) generate the list of words in each test set
                (5) compare the words that shared in the training set and test set
                (6) calculate the average of similarity scores between the different functions of postposition in the training set and the shared words
                (7) determine the function of postposition used in test set with the highest average of similarity scores
            end
        end
    end
end

Figure A.2: Algorithm of the similarity-based estimation

```
Input: Postpositions (p), Functions of each postposition (f), Epochs (e), Batches (b)

Output: the 2-dimensional distribution data

Parameters: epoch: 50, batch: 32, epsilon (eps): 0.00000008, learning rate (lr): 0.00002, seed: 42

Process;

foreach postposition in range(0, length(p))
    (1) pre-trained model load: BertForSequenceClassification (kobert, number of f)
    foreach epoch in range(0, length(e))
        (2) load the training set
        (3) use pre-trained model for training and fine-tuning: model.train()
        foreach batch in range(0, length(b))
            (4) calculate the loss & optimize the lr to reduce the loss
        end
        (5) load the testing set
        (6) use fine-tuned model for testing: model.eval()
        foreach batch in range(0, length(b))
            (7) calculate the accuracy rate of f
            (8) generate BERT outcomes of the testing set
        end
        (9) reduce the dimensions of BERT outcomes as two dimensions by using t-SNE
        (10) save the reduced t-SNE outcomes as CSV file
    end
end
```

Figure A.3: Algorithm of the BERT training

Appendix B

Code for the word-level embedding models

The following scripts are the code that I used for the training of *traditional word embedding models* (i.e., PPMI-SVD, SGNS) and *similarity-based estimation*.

Listing B.1: Python code for the word embedding by using the PPMI-SVD model

```python

class PPMI_SVD_Algorithm:

    def __init__ (self, fold, postposition, postposition_ko,
        window):
        self.fold = fold
        self.postposition = postposition
        self.postposition_ko = postposition_ko
        self.window = window

    def PPMI_SVD_Calculation(self):

        from collections import Counter
        import itertools
```

```
import nltk
from nltk.corpus import stopwords
import numpy as np
import pandas as pd
from scipy import sparse
from scipy.sparse import linalg
from sklearn.preprocessing import normalize
from sklearn.metrics.pairwise import cosine_similarity

trainDir = "../../Data/Input/Fold/" + str(self.fold) +
    "Fold/"+ self.postposition +"_train_" + str(self.
    fold) + ".csv"

# data load
df = pd.read_csv(trainDir)
print(df.head())
headlines = df['Sentence'].tolist()
headlines = [[tok for tok in headline.split()] for
    headline in headlines]
# remove single word headlines
headlines = [hl for hl in headlines if len(hl) > 1]
# show results
print(headlines[0:20])

# calculate a unigram vocabulary
tok2indx = dict()
unigram_counts = Counter()
for ii, headline in enumerate(headlines):
    if ii % 200000 == 0:
        print(f'finished {ii / len(headlines):.2%} of
            headlines')
```

```
            for token in headline:
                unigram_counts[token] += 1
                if token not in tok2indx:
                    tok2indx[token] = len(tok2indx)
        indx2tok = {indx: tok for tok, indx in tok2indx.items()}
        print('done')
        print('vocabulary size: {}'.format(len(unigram_counts)))
        print('most common: {}'.format(unigram_counts.most_common(10)))

        wordType = len(unigram_counts);

        for j in range(1, self.window):
            # Skipgrams
            back_window = j
            front_window = j
            skipgram_counts = Counter()
            for iheadline, headline in enumerate(headlines):
                for ifw, fw in enumerate(headline):
                    icw_min = max(0, ifw - back_window)
                    icw_max = min(len(headline) - 1, ifw + front_window)
                    icws = [ii for ii in range(icw_min, icw_max + 1) if ii != ifw]
                    for icw in icws:
                        skipgram = (headline[ifw], headline[icw])
                        skipgram_counts[skipgram] += 1
                if iheadline % 200000 == 0:
```

```
        print(f'finished {iheadline / len(headlines):.2%} of headlines')
print('done')
print('number of skipgrams: {}'.format(len(skipgram_counts)))
print('most common: {}'.format(skipgram_counts.most_common(10)))

# Word-Word Count Matrix
row_indxs = []
col_indxs = []
dat_values = []
ii = 0
for (tok1, tok2), sg_count in skipgram_counts.items():
    ii += 1
    if ii % 1000000 == 0:
        print(f'finished {ii / len(skipgram_counts):.2%} of skipgrams')
    tok1_indx = tok2indx[tok1]
    tok2_indx = tok2indx[tok2]

    row_indxs.append(tok1_indx)
    col_indxs.append(tok2_indx)
    dat_values.append(sg_count)

wwcnt_mat = sparse.csr_matrix((dat_values, (row_indxs, col_indxs)))
print('done')

# normalize each row using L2 norm
```

```
wwcnt_norm_mat = normalize(wwcnt_mat, norm='l2', axis=1)

##Word Similarity with Sparse Count Matrices
def ww_sim(word, mat, topn=len(tok2indx)):
    indx = tok2indx[word]
    if isinstance(mat, sparse.csr_matrix):
        v1 = mat.getrow(indx)
    else:
        v1 = mat[indx:indx + 1, :]
    sims = cosine_similarity(mat, v1).flatten()
    sindxs = np.argsort(-sims)
    sim_word_scores = [(indx2tok[sindx], sims[sindx]) for sindx in sindxs[0:topn]]
    return sim_word_scores

# Pointwise Mutual Information Matrices
num_skipgrams = wwcnt_mat.sum()
assert (sum(skipgram_counts.values()) == num_skipgrams)

# for creating sparce matrices
row_indxs = []
col_indxs = []

pmi_dat_values = []
ppmi_dat_values = []
spmi_dat_values = []
sppmi_dat_values = []

# smoothing
```

```
    alpha = 0.75
    nca_denom = np.sum(np.array(wwcnt_mat.sum(axis=0)).flatten() ** alpha)
    sum_over_words = np.array(wwcnt_mat.sum(axis=0)).flatten()
    sum_over_words_alpha = sum_over_words ** alpha
    sum_over_contexts = np.array(wwcnt_mat.sum(axis=1)).flatten()

    ii = 0
    for (tok1, tok2), sg_count in skipgram_counts.items():
        ii += 1
        if ii % 1000000 == 0:
            print(f'finished {ii / len(skipgram_counts):.2%} of skipgrams')
        tok1_indx = tok2indx[tok1]
        tok2_indx = tok2indx[tok2]

        nwc = sg_count
        Pwc = nwc / num_skipgrams
        nw = sum_over_contexts[tok1_indx]
        Pw = nw / num_skipgrams
        nc = sum_over_words[tok2_indx]
        Pc = nc / num_skipgrams

        nca = sum_over_words_alpha[tok2_indx]
        Pca = nca / nca_denom

        pmi = np.log2(Pwc / (Pw * Pc))
        ppmi = max(pmi, 0)
```

```

            spmi = np.log2(Pwc / (Pw * Pca))
            sppmi = max(spmi, 0)

            row_indxs.append(tok1_indx)
            col_indxs.append(tok2_indx)
            pmi_dat_values.append(pmi)
            ppmi_dat_values.append(ppmi)
            spmi_dat_values.append(spmi)
            sppmi_dat_values.append(sppmi)

        pmi_mat = sparse.csr_matrix((pmi_dat_values, (
            row_indxs, col_indxs)))
        ppmi_mat = sparse.csr_matrix((ppmi_dat_values, (
            row_indxs, col_indxs)))
        spmi_mat = sparse.csr_matrix((spmi_dat_values, (
            row_indxs, col_indxs)))
        sppmi_mat = sparse.csr_matrix((sppmi_dat_values, (
            row_indxs, col_indxs)))

        print('done')

        # Singular Value Decomposition
        matrix_use = ppmi_mat

        if wordType < 500:
            embedding_size = wordType - 1
        else:
            embedding_size = 500

```

```
uu, ss, vv = linalg.svds(matrix_use, embedding_size
    )
print('vocab size: {}'.format(len(unigram_counts)))
print('embedding size: {}'.format(embedding_size))
print('uu.shape: {}'.format(uu.shape))
print('ss.shape: {}'.format(ss.shape))
print('vv.shape: {}'.format(vv.shape))

unorm = uu / np.sqrt(np.sum(uu * uu, axis=1,
    keepdims=True))
vnorm = vv / np.sqrt(np.sum(vv * vv, axis=0,
    keepdims=True))
word_vecs = uu + vv.T
word_vecs_norm = word_vecs / np.sqrt(np.sum(
    word_vecs * word_vecs, axis=1, keepdims=True))

print(word_vecs_norm)

from sklearn.manifold import TSNE
X_embedded = TSNE(n_components=2, random_state=0).
    fit_transform(word_vecs_norm)

wordList = []
wordnum = 0
for typeeach in indx2tok:
    wordList.append(indx2tok[wordnum])
    wordnum += 1

tsne_df = pd.DataFrame({'X': X_embedded[:, 0], 'Y':
    X_embedded[:, 1], 'Word': wordList})
```

```
        tsne_df.to_csv("../../Data/Output/PPMI_SVD/" + self.postposition + "/t-SNE/" + self.postposition + "_tSNE_" + str(
                j) + ".csv")

        TSNE_dic = {}

        typenum = 0
        for typeeach in indx2tok:
            TSNE_dic[indx2tok[typenum]] = [X_embedded[typenum][0], X_embedded[typenum][1]]
            typenum = typenum + 1

        functionEy = ["LOC", "GOL", "EFF", "CRT", "THM", "INS", "AGT", "FNS"]
        functionEyse = ["SRC", "LOC"]
        functionLo = ["FNS", "INS", "DIR", "EFF", "CRT", "LOC"]

        if self.postposition == "Ey":
            for function in functionEy:
                word = self.postposition_ko + "/JKB" + "_" + function
                from numpy import dot
                from numpy.linalg import norm
                import numpy as np
                def cos_sim(A, B):
                    return dot(A, B) / (norm(A) * norm(B))
                target = np.array(TSNE_dic[word])
                outDir = "../../Data/Output/PPMI_SVD/" + self.postposition + "/Similarity/" +
```

```
                self.postposition + "_" + function + "
                _Similarity_" + str(
                 j) + ".csv"
            f = open(outDir, 'w')
            tsnenum = 0
            for typeeach in indx2tok:
                if indx2tok[tsnenum] != self.
                    postposition:
                    source = np.array(TSNE_dic[indx2tok
                        [tsnenum]])
                    normal_sim = (cos_sim(target,
                        source) + 1) / 2
                    data = str(indx2tok[tsnenum]) + ","
                         + str(normal_sim)
                    f.write(data + "\n")
                tsnenum = tsnenum + 1
            f.close()

    elif self.postposition == "Eyse":
        for function in functionEyse:
            word = self.postposition_ko + "/JKB" + "_"
                + function
            from numpy import dot
            from numpy.linalg import norm
            import numpy as np
            def cos_sim(A, B):
                return dot(A, B) / (norm(A) * norm(B))
            target = np.array(TSNE_dic[word])
            outDir = "../../Data/Output/PPMI_SVD/" +
                self.postposition + "/Similarity/" +
                self.postposition + "_" + function + "
```

```
                    _Similarity_" + str(
                    j) + ".csv"
                f = open(outDir, 'w')
                tsnenum = 0
                for typeeach in indx2tok:
                    if indx2tok[tsnenum] != self.
                        postposition:
                        source = np.array(TSNE_dic[indx2tok
                            [tsnenum]])
                        normal_sim = (cos_sim(target,
                            source) + 1) / 2
                        data = str(indx2tok[tsnenum]) + ","
                             + str(normal_sim)
                        f.write(data + "\n")
                    tsnenum = tsnenum + 1
                f.close()

        elif self.postposition == "Lo":
            for function in functionLo:
                word = self.postposition_ko + "/JKB" + "_"
                    + function
                from numpy import dot
                from numpy.linalg import norm
                import numpy as np
                def cos_sim(A, B):
                    return dot(A, B) / (norm(A) * norm(B))
                target = np.array(TSNE_dic[word])
                outDir = "../../Data/Output/PPMI_SVD/" +
                    self.postposition + "/Similarity/" +
                    self.postposition + "_" + function + "
                    _Similarity_" + str(
```

```
        j) + ".csv"
    f = open(outDir, 'w')
    tsnenum = 0
    for typeeach in indx2tok:
        if indx2tok[tsnenum] != self.
            postposition:
            source = np.array(TSNE_dic[indx2tok
                [tsnenum]])
            normal_sim = (cos_sim(target,
                source) + 1) / 2
            data = str(indx2tok[tsnenum]) + ","
                + str(normal_sim)
            f.write(data + "\n")
        tsnenum = tsnenum + 1
    f.close()
```

Listing B.2: Python code for the word embedding by using the SGNS model

```
class SGNS_Algorithm:

    def __init__(self, fold, postposition, postposition_ko,
        window):
        self.fold = fold
        self.postposition = postposition
        self.postposition_ko = postposition_ko
        self.window = window

    def SGNS_Calculation(self):

        from collections import Counter
        import itertools
        import nltk
        from nltk.corpus import stopwords
        import numpy as np
        import pandas as pd
        from scipy import sparse
        from scipy.sparse import linalg
        from sklearn.preprocessing import normalize
        from sklearn.metrics.pairwise import cosine_similarity

        trainDir = "../../Data/Input/Fold/" + str(self.fold) +
            "Fold/" + self.postposition + "_train_" + str(self.
            fold) + ".csv"

        # data load
        df = pd.read_csv(trainDir)
        print(df.head())
```

```python
headlines = df['Sentence'].tolist()
headlines = [[tok for tok in headline.split()] for headline in headlines]
# remove single word headlines
headlines = [hl for hl in headlines if len(hl) > 1]
# show results
print(headlines[0:20])

# calculate a unigram vocabulary
tok2indx = dict()
unigram_counts = Counter()
for ii, headline in enumerate(headlines):
    if ii % 200000 == 0:
        print(f'finished {ii / len(headlines):.2%} of headlines')
    for token in headline:
        unigram_counts[token] += 1
        if token not in tok2indx:
            tok2indx[token] = len(tok2indx)
indx2tok = {indx: tok for tok, indx in tok2indx.items()}
print('done')
print('vocabulary size: {}'.format(len(unigram_counts)))
print('most common: {}'.format(unigram_counts.most_common(10)))

wordType = len(unigram_counts);

for j in range(1, self.window):

```

```
    from gensim.models import Word2Vec

    if wordType < 500:
        embedding_size = wordType - 1
    else:
        embedding_size = 500

    embedding_model = Word2Vec(headlines, size=embedding_size, window=j, min_count=0, workers=4, iter=100, sg=1, negative=15, ns_exponent=0.75)

    matrix = []
    indxnum = 0
    for typeeach in indx2tok:
        line = list(embedding_model[indx2tok[typeeach]])
        matrix.append(line)
        indxnum = indxnum + 1

    embedded_matrix = np.array(matrix, dtype=np.float64)

    print(embedded_matrix)

    from sklearn.manifold import TSNE
    X_embedded = TSNE(n_components=2, random_state=0).fit_transform(embedded_matrix)

    wordList = []
    wordnum = 0
    for typeeach in indx2tok:
```

```
            wordList.append(indx2tok[wordnum])
            wordnum += 1

        tsne_df = pd.DataFrame({'X': X_embedded[:, 0], 'Y': X_embedded[:, 1], 'Word': wordList})
        tsne_df.to_csv(
            "../../Data/Output/SGNS/" + self.postposition + "/t-SNE/" + self.postposition + "_tSNE_" + str(
                j) + ".csv")

        # Word Similarity with Sparse Count Matrices
        def ww_sim(word, mat, topn=len(tok2indx)):
            indx = tok2indx[word]
            if isinstance(mat, sparse.csr_matrix):
                v1 = mat.getrow(indx)
            else:
                v1 = mat[indx:indx + 1, :]
            sims = cosine_similarity(mat, v1).flatten()
            sindxs = np.argsort(-sims)
            sim_word_scores = [(indx2tok[sindx], sims[sindx]) for sindx in sindxs[0:topn]]
            return sim_word_scores

        # Word Similarity
        def word_sim_report(word, sim_mat):
            output = {}
            sim_word_scores = ww_sim(word, embedded_matrix)
            for sim_word, sim_score in sim_word_scores:
                output[sim_word] = ((sim_score + 1) / 2)
            return output
```

```python

        TSNE_dic = {}

        typenum = 0
        for typeeach in indx2tok:
            TSNE_dic[indx2tok[typenum]] = [X_embedded[typenum][0], X_embedded[typenum][1]]
            typenum = typenum + 1

        functionEy = ["LOC", "GOL", "EFF", "CRT", "THM", "INS", "AGT", "FNS"]
        functionEyse = ["SRC", "LOC"]
        functionLo = ["FNS", "INS", "DIR", "EFF", "CRT", "LOC"]

        if self.postposition == "Ey":
            for function in functionEy:
                word = self.postposition_ko + "/JKB" + "_" + function
                from numpy import dot
                from numpy.linalg import norm
                import numpy as np
                def cos_sim(A, B):
                    return dot(A, B) / (norm(A) * norm(B))
                target = np.array(TSNE_dic[word])
                outDir = "../../Data/Output/SGNS/" + self.postposition + "/Similarity/" + self.postposition + "_" + function + "_Similarity_" + str(
                    j) + ".csv"
                f = open(outDir, 'w')
```

```python
                tsnenum = 0
                for typeeach in indx2tok:
                    if indx2tok[tsnenum] != self.postposition:
                        source = np.array(TSNE_dic[indx2tok[tsnenum]])
                        normal_sim = (cos_sim(target, source) + 1) / 2
                        data = str(indx2tok[tsnenum]) + "," + str(normal_sim)
                        f.write(data + "\n")
                    tsnenum = tsnenum + 1
                f.close()




        elif self.postposition == "Eyse":
            for function in functionEyse:
                word = self.postposition_ko + "/JKB" + "_" + function
                from numpy import dot
                from numpy.linalg import norm
                import numpy as np
                def cos_sim(A, B):
                    return dot(A, B) / (norm(A) * norm(B))
                target = np.array(TSNE_dic[word])
                outDir = "../../Data/Output/SGNS/" + self.postposition + "/Similarity/" + self.postposition + "_" + function + "_Similarity_" + str(
```

```
                j) + ".csv"
            f = open(outDir, 'w')
            tsnenum = 0
            for typeeach in indx2tok:
                if indx2tok[tsnenum] != self.postposition:
                    source = np.array(TSNE_dic[indx2tok[tsnenum]])
                    normal_sim = (cos_sim(target, source) + 1) / 2
                    data = str(indx2tok[tsnenum]) + "," + str(normal_sim)
                    f.write(data + "\n")
                tsnenum = tsnenum + 1
            f.close()

        elif self.postposition == "Lo":
            for function in functionLo:
                word = self.postposition_ko + "/JKB" + "_" + function
                from numpy import dot
                from numpy.linalg import norm
                import numpy as np
                def cos_sim(A, B):
                    return dot(A, B) / (norm(A) * norm(B))
                target = np.array(TSNE_dic[word])
                outDir = "../../Data/Output/SGNS/" + self.postposition + "/Similarity/" + self.postposition + "_" + function + "_Similarity_" + str(
                j) + ".csv"
```

```
f = open(outDir, 'w')
tsnenum = 0
for typeeach in indx2tok:
    if indx2tok[tsnenum] != self.postposition:
        source = np.array(TSNE_dic[indx2tok[tsnenum]])
        normal_sim = (cos_sim(target, source) + 1) / 2
        data = str(indx2tok[tsnenum]) + "," + str(normal_sim)
        f.write(data + "\n")
    tsnenum = tsnenum + 1
f.close()
```

Listing B.3: Python code for the similarity-based estimation

```
class SBEs:

    def __init__(self, method, postposition, postposition_ko, fold, window, function):
        self.method = method
        self.postposition = postposition
        self.postposition_ko = postposition_ko
        self.fold = fold
        self.window = window
        self.function = function

    def Processing(self):
        import numpy as np
        import pandas as pd

        functions = self.function
        functionDicDic = {}

        for function in functions:
            functionDic = {}
            functionDir = "../../Data/Output/" + self.method + "/" + self.postposition + "/" + str(self.fold) + "Fold/" + self.postposition + "_" + function + "_window_" + str(self.window) + ".csv"

            dfFunction = pd.read_csv(functionDir)
            words = dfFunction['word'].tolist()
            sims = dfFunction['similarity'].tolist()
            for k in range(0, len(words)):
```

```
                functionDic[words[k]] = sims[k]
            functionDicDic[function] = functionDic

        testDir = "../../Data/Input/Fold/" + str(self.fold) + "Fold/" + self.postposition + "_test_" + str(self.fold) + ".csv"

        df = pd.read_csv(testDir)
        headlines = df['Sentence'].tolist()

        countDic = {}
        frequencyDic = {}

        countDic["Total"] = 0
        frequencyDic["Total"] = 0

        for function in functions:
            countDic[function] = 0
            frequencyDic[function] = 0

        for sentence in headlines:
            originClass = ""
            token = sentence.split(" ")
            for eachToken in token:
                if (self.postposition_ko + "/") in eachToken and "JKB_" in eachToken:
                    originClass = eachToken.replace((self.postposition_ko + "/"), "").replace("JKB_", "")

            classifiedFunc = {}
```

```

        funcScore = {}
        matchNum = 0

        for eachToken in token:
            if functionDicDic.get("LOC").get(eachToken.
                strip()) == None or ((self.postposition_ko +
                 "/") in eachToken and "JKB_" in eachToken):
                pass
            else:
                matchNum = matchNum + 1
                for function in functions:
                    if funcScore.get(function) == None:
                        funcScore[function] =
                            functionDicDic.get(function).get
                            (eachToken.strip())
                    else:
                        funcScore[function] = funcScore.get
                            (function) + functionDicDic.get(
                            function).get(eachToken.strip())

        for function in functions:
            classifiedFunc[function] = funcScore.get(
                function) / matchNum

        dic_max = max(classifiedFunc.values())

        for x, y in classifiedFunc.items():
            if y == dic_max:
                for function in functions:
                    if originClass == function and
```

```
originClass == x:
                        countDic[function] = countDic.get(function) + 1
                if originClass == x:
                    countDic["Total"] = countDic.get("Total") + 1

        frequencyDic["Total"] = frequencyDic.get("Total") + 1

        for function in functions:
            if originClass == function:
                frequencyDic[function] = frequencyDic.get(function) + 1

    finalResult = {}

    totalAccuracy = countDic.get("Total") / frequencyDic.get("Total")
    finalResult["Total"] = totalAccuracy

    for function in functions:
        funcAccuracy = countDic.get(function) / frequencyDic.get(function)
        finalResult[function] = funcAccuracy

    averageAccuracy = 0
    for function in functions:
        averageAccuracy = averageAccuracy+finalResult.get(function)

```

```
        finalResult["TotalAverage"] = averageAccuracy / len(
            functions)

        return finalResult
```

Appendix C

Code for the sentence-level embedding model

The following script is the code that I used for the training of *contextualized word embedding model* (i.e., BERT).

Listing C.1: Python code for the BERT training by using the *BertForSequence-Classification*

```python
class BERT_Algorithm:

    def __init__(self, postposition, lablesNum):
        self.postposition = postposition
        self.lablesNum = lablesNum

    def BERT_Calculation(self):

        import tensorflow as tf
        # Get the GPU device name.
        device_name = tf.test.gpu_device_name()
        # The device name should look like the following:
        if device_name == '/device:GPU:0':
```

```python
    print('Found GPU at: {}'.format(device_name))
else:
    raise SystemError('GPU device not found')

import torch
# If there's a GPU available...
if torch.cuda.is_available():
    # Tell PyTorch to use the GPU.
    device = torch.device("cuda")
    print('There are %d GPU(s) available.' % torch.cuda.device_count())
    print('We will use the GPU:', torch.cuda.get_device_name(0))
# If not...
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")

!pip install transformers

# Mount Google Drive to this Notebook instance.
from google.colab import drive
drive.mount('/content/drive')

import tensorflow as tf
import torch
from transformers import BertTokenizer
from transformers import BertForSequenceClassification, AdamW, BertConfig
from transformers import get_linear_schedule_with_warmup
```

```
        from torch.utils.data import TensorDataset, DataLoader,
            RandomSampler, SequentialSampler
        from keras.preprocessing.sequence import pad_sequences
        from sklearn.model_selection import train_test_split
        import pandas as pd
        import numpy as np
        import random
        import time
        import datetime

        fileDir = "drive/My Drive/BERT/SM/KoBERT/Postposition/
           Data/test_"+self.postposition+".csv"
        fr = open(fileDir, 'r')
        contents= fr.readlines()
        fr.close()

        test = pd.DataFrame(columns=('index', 'Label', '
           Sentence'))
        i = 0
        index = ""
        label = ""
        sentence = ""
        for content in contents:
            if i == 0:
                pass
            else:
                infos = content.split(",")
                index = infos[0]
                label = int(infos[1])
                sentence = infos[2].replace("\n","")
                test.loc[i] = [index, label, sentence]
```

```
            i = i + 1

        fileDir = "drive/My Drive/BERT/SM/KoBERT/Postposition/
            Data/train_"+self.postposition+".csv"
        fr = open(fileDir, 'r')
        contents= fr.readlines()
        fr.close()
        train = pd.DataFrame(columns=('index', 'Label', '
            Sentence'))
        i = 0
        index = ""
        label = ""
        sentence = ""
        for content in contents:
            if i == 0:
                pass
            else:
                infos = content.split(",")
                index = infos[0]
                label = int(infos[1])
                sentence = infos[2].replace("\n","")
                train.loc[i] = [index, label, sentence]
            i = i + 1

        sentences = train['Sentence']
        sentences = ["[CLS] " + str(sentence) + " [SEP]" for
            sentence in sentences]

        labels = train['Label'].values
        labels_re = []
        for label in labels:
```

```
      labels_re.append(label)
    labels = labels_re

    tokenizer = KoBertTokenizer.from_pretrained('monologg/kobert')
    tokenized_texts = [tokenizer.tokenize(sent) for sent in sentences]

    MAX_LEN = 128
    input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]
    input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long", truncating="post", padding="post")

    attention_masks = []
    for seq in input_ids:
        seq_mask = [float(i>0) for i in seq]
        attention_masks.append(seq_mask)

    train_inputs, validation_inputs, train_labels, validation_labels = train_test_split(input_ids, labels,random_state=2018,test_size=0.1)
    train_masks, validation_masks, _, _ = train_test_split(attention_masks,input_ids,random_state=2018, test_size=0.1)
    train_inputs = torch.tensor(train_inputs)
    train_labels = torch.tensor(train_labels)
    train_masks = torch.tensor(train_masks)
    validation_inputs = torch.tensor(validation_inputs)
    validation_labels = torch.tensor(validation_labels)
    validation_masks = torch.tensor(validation_masks)
```

```
batch_size = 32

train_data = TensorDataset(train_inputs, train_masks,
    train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=
    train_sampler, batch_size=batch_size)
validation_data = TensorDataset(validation_inputs,
    validation_masks, validation_labels)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data,
    sampler=validation_sampler, batch_size=batch_size)

sentences = test['Sentence']
sentences = ["[CLS] " + str(sentence) + " [SEP]" for
    sentence in sentences]

labels = test['Label'].values
labels_re = []
for label in labels:
  labels_re.append(label)
labels = labels_re

tokenizer = KoBertTokenizer.from_pretrained('monologg/
    kobert')
tokenized_texts = [tokenizer.tokenize(sent) for sent in
     sentences]

MAX_LEN = 128
```

```
        input_ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_texts]
        input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long", truncating="post", padding="post")

        attention_masks = []

        for seq in input_ids:
            seq_mask = [float(i>0) for i in seq]
            attention_masks.append(seq_mask)
        test_inputs = torch.tensor(input_ids)
        test_labels = torch.tensor(labels)
        test_masks = torch.tensor(attention_masks)

        batch_size = 32

        test_data = TensorDataset(test_inputs, test_masks, test_labels)
        test_sampler = RandomSampler(test_data)
        test_dataloader = DataLoader(test_data, sampler=test_sampler, batch_size=batch_size)
        test_dataloader

        model = BertForSequenceClassification.from_pretrained("monologg/kobert", num_labels=self.lablesNum)
        model.cuda()

        def format_time(elapsed):
            elapsed_rounded = int(round((elapsed)))
            return str(datetime.timedelta(seconds=elapsed_rounded))
```

```python

optimizer = AdamW(model.parameters(),lr = 2e-5,eps = 1e
    -8)
epochs = 50
total_steps = len(train_dataloader) * epochs
scheduler = get_linear_schedule_with_warmup(optimizer,
    num_warmup_steps = 0,num_training_steps =
    total_steps)

seed_val = 42
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

if self.postposition == "Ey":

    def flat_accuracy(preds, labels):
        pred_flat = np.argmax(preds, axis=1).flatten()
        labels_flat = labels.flatten()
        return np.sum(pred_flat == labels_flat) / len(
            labels_flat)

    def FNS_flat_accuracy(preds, labels):
        pred_flat = np.argmax(preds, axis=1).flatten()
        labels_flat = labels.flatten()
        match_num = 0
        func_num = 0
        for i in range(0,len(pred_flat)):
          if (pred_flat[i] == labels_flat[i]) and (
              labels_flat[i] == 0):
```

```
        match_num += 1
      if labels_flat[i] == 0:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

def INS_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
      if (pred_flat[i] == labels_flat[i]) and (
         labels_flat[i] == 1):
        match_num += 1
      if labels_flat[i] == 1:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

def GOL_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
```

```
        if (pred_flat[i] == labels_flat[i]) and (labels_flat[i] == 2):
            match_num += 1
        if labels_flat[i] == 2:
            func_num += 1
    if match_num == 0 or func_num == 0:
        return 0
    else:
        return match_num / func_num

def EFF_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
        if (pred_flat[i] == labels_flat[i]) and (labels_flat[i] == 3):
            match_num += 1
        if labels_flat[i] == 3:
            func_num += 1
    if match_num == 0 or func_num == 0:
        return 0
    else:
        return match_num / func_num

def CRT_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
```

```
    for i in range(0,len(pred_flat)):
      if (pred_flat[i] == labels_flat[i]) and (labels_flat[i] == 4):
        match_num += 1
      if labels_flat[i] == 4:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

def LOC_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
      if (pred_flat[i] == labels_flat[i]) and (labels_flat[i] == 5):
        match_num += 1
      if labels_flat[i] == 5:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

def AGT_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
```

```
    func_num = 0
    for i in range(0,len(pred_flat)):
      if (pred_flat[i] == labels_flat[i]) and (labels_flat[i] == 6):
        match_num += 1
      if labels_flat[i] == 6:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

def THM_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
      #print(pred_flat[i]," / ",labels_flat[i])
      if (pred_flat[i] == labels_flat[i]) and (labels_flat[i] == 7):
        match_num += 1
      if labels_flat[i] == 7:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

model.zero_grad()

```

```
        final_info = {}

        for epoch_i in range(0, epochs):

            print("")
            print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
            print('Training...')

            t0 = time.time()
            total_loss = 0
            model.train()

            for step, batch in enumerate(train_dataloader):
                if step % 500 == 0 and not step == 0:
                    elapsed = format_time(time.time() - t0)
                    print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format(step, len(train_dataloader), elapsed))

                batch = tuple(t.to(device) for t in batch)
                b_input_ids, b_input_mask, b_labels = batch
                outputs = model(b_input_ids,
                                token_type_ids=None,
                                attention_mask=b_input_mask,
                                labels=b_labels)

                loss = outputs[0]
                total_loss += loss.item()
                loss.backward()
```

```
            torch.nn.utils.clip_grad_norm_(model.
                parameters(), 1.0)
            optimizer.step()
            scheduler.step()
            model.zero_grad()

        avg_train_loss = total_loss / len(
            train_dataloader)

        print("")
        print("  Average training loss: {0:.2f}".format
            (avg_train_loss))
        print("  Training epcoh took: {:}".format(
            format_time(time.time() - t0)))

        print("")
        print("Running Validation...")

        t0 = time.time()

        model.eval()

        eval_loss, eval_accuracy = 0, 0
        nb_eval_steps, nb_eval_examples = 0, 0
        FNS_nb_eval_steps, FNS_eval_accuracy = 0, 0
        INS_nb_eval_steps, INS_eval_accuracy = 0, 0
        GOL_nb_eval_steps, GOL_eval_accuracy = 0, 0
        EFF_nb_eval_steps, EFF_eval_accuracy = 0, 0
        CRT_nb_eval_steps, CRT_eval_accuracy = 0, 0
        LOC_nb_eval_steps, LOC_eval_accuracy = 0, 0
```

```
        AGT_nb_eval_steps, AGT_eval_accuracy = 0, 0
        THM_nb_eval_steps, THM_eval_accuracy = 0, 0

        epoch_info = {}

        for batch in test_dataloader:
            batch = tuple(t.to(device) for t in batch)
            b_input_ids, b_input_mask, b_labels = batch
            with torch.no_grad():
                outputs = model(b_input_ids,
                                token_type_ids=None,
                                attention_mask=
                                    b_input_mask)

            logits = outputs[0]
            logits = logits.detach().cpu().numpy()
            label_ids = b_labels.to('cpu').numpy()

            tmp_eval_accuracy = flat_accuracy(logits,
                label_ids)
            eval_accuracy += tmp_eval_accuracy
            nb_eval_steps += 1

            FNS_tmp_eval_accuracy = FNS_flat_accuracy(
                logits, label_ids)
            FNS_eval_accuracy += FNS_tmp_eval_accuracy
            FNS_nb_eval_steps += 1

            INS_tmp_eval_accuracy = INS_flat_accuracy(
                logits, label_ids)
            INS_eval_accuracy += INS_tmp_eval_accuracy
```

```
                INS_nb_eval_steps += 1

                GOL_tmp_eval_accuracy = GOL_flat_accuracy(
                    logits, label_ids)
                GOL_eval_accuracy += GOL_tmp_eval_accuracy
                GOL_nb_eval_steps += 1

                EFF_tmp_eval_accuracy = EFF_flat_accuracy(
                    logits, label_ids)
                EFF_eval_accuracy += EFF_tmp_eval_accuracy
                EFF_nb_eval_steps += 1

                CRT_tmp_eval_accuracy = CRT_flat_accuracy(
                    logits, label_ids)
                CRT_eval_accuracy += CRT_tmp_eval_accuracy
                CRT_nb_eval_steps += 1

                LOC_tmp_eval_accuracy = LOC_flat_accuracy(
                    logits, label_ids)
                LOC_eval_accuracy += LOC_tmp_eval_accuracy
                LOC_nb_eval_steps += 1

                AGT_tmp_eval_accuracy = AGT_flat_accuracy(
                    logits, label_ids)
                AGT_eval_accuracy += AGT_tmp_eval_accuracy
                AGT_nb_eval_steps += 1

                THM_tmp_eval_accuracy = THM_flat_accuracy(
                    logits, label_ids)
                THM_eval_accuracy += THM_tmp_eval_accuracy
                THM_nb_eval_steps += 1
```

```
                print("  Accuracy: {0:.2f}".format(
                    eval_accuracy/nb_eval_steps))
                print("  Validation took: {:}".format(
                    format_time(time.time() - t0)))
                print("")
                print("  Detail accuracy  ")
                print("  FNS_Accuracy: {0:.2f}".format(
                    FNS_eval_accuracy/FNS_nb_eval_steps))
                print("  INS_Accuracy: {0:.2f}".format(
                    INS_eval_accuracy/INS_nb_eval_steps))
                print("  GOL_Accuracy: {0:.2f}".format(
                    GOL_eval_accuracy/GOL_nb_eval_steps))
                print("  EFF_Accuracy: {0:.2f}".format(
                    EFF_eval_accuracy/EFF_nb_eval_steps))
                print("  CRT_Accuracy: {0:.2f}".format(
                    CRT_eval_accuracy/CRT_nb_eval_steps))
                print("  LOC_Accuracy: {0:.2f}".format(
                    LOC_eval_accuracy/LOC_nb_eval_steps))
                print("  AGT_Accuracy: {0:.2f}".format(
                    AGT_eval_accuracy/AGT_nb_eval_steps))
                print("  THM_Accuracy: {0:.2f}".format(
                    THM_eval_accuracy/THM_nb_eval_steps))

                epoch_info["Total"] = round(eval_accuracy/
                    nb_eval_steps,3)
                epoch_info["Loss"] = round(avg_train_loss,3)
                epoch_info["FNS"] = round(FNS_eval_accuracy/
                    FNS_nb_eval_steps,3)
                epoch_info["INS"] = round(INS_eval_accuracy/
                    INS_nb_eval_steps,3)
```

```
epoch_info["GOL"] = round(GOL_eval_accuracy/
    GOL_nb_eval_steps,3)
epoch_info["EFF"] = round(EFF_eval_accuracy/
    EFF_nb_eval_steps,3)
epoch_info["CRT"] = round(CRT_eval_accuracy/
    CRT_nb_eval_steps,3)
epoch_info["LOC"] = round(LOC_eval_accuracy/
    LOC_nb_eval_steps,3)
epoch_info["AGT"] = round(AGT_eval_accuracy/
    AGT_nb_eval_steps,3)
epoch_info["THM"] = round(THM_eval_accuracy/
    THM_nb_eval_steps,3)

final_info["epoch"+str(epoch_i)] = epoch_info

model.eval()
test_input_ids = []
test_input_mask = []
test_labels = []

num = 0
for step, batch in enumerate(test_data):
  batch = tuple(t.to(device) for t in batch)

  b_input_ids, b_input_mask, b_labels = batch
  input_ids_arr = []
  input_mask_arr = []

  for i in range(0,len(b_input_ids)):
    input_ids_arr.append(int(b_input_ids[i]))
    input_mask_arr.append(int(b_input_mask[i]))
```

```

          test_input_ids.append(input_ids_arr)
          test_input_mask.append(input_mask_arr)
          test_labels.append(int(b_labels))

        test_input_ids = torch.tensor(test_input_ids)
        test_input_mask = torch.tensor(test_input_mask)
        test_labels = test_labels

        test_input_ids = test_input_ids.to(device)
        test_input_mask = test_input_mask.to(device)

        with torch.no_grad():
            outputs = model(test_input_ids,
                            token_type_ids=None,
                            attention_mask=
                              test_input_mask)

        sentence_vecs_sum = outputs[0]

        sentence_array = []
        for i in range(0,len(sentence_vecs_sum)):
          each_array = []
          for j in range(0,len(sentence_vecs_sum[i])):
            each_array.append(float(sentence_vecs_sum[i
              ][j]))
          sentence_array.append(each_array)

        initial_df = pd.DataFrame(sentence_array)

        from sklearn.manifold import TSNE
```

```
                tsne = TSNE(n_components=2, random_state=0)
                tsne_obj= tsne.fit_transform(initial_df)

                tsne_df = pd.DataFrame({'X':tsne_obj[:,0],'Y':tsne_obj[:,1],'Label':test_labels})

                import numpy as np
                import pandas as pd
                from plotnine import *

                print("")
                print("  Network visualization  ")
                print(ggplot(tsne_df, aes(x='X', y='Y')) + geom_point(aes(colour = 'Label')))

            print("")
            print("Training complete!")
            print("")
            print("Final result is below!")
            print(final_info)

        elif self.postposition == "Eyse":

            def flat_accuracy(preds, labels):
                pred_flat = np.argmax(preds, axis=1).flatten()
                labels_flat = labels.flatten()
                return np.sum(pred_flat == labels_flat) / len(labels_flat)

            def SRC_flat_accuracy(preds, labels):
                pred_flat = np.argmax(preds, axis=1).flatten()
```

```
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
      if (pred_flat[i] == labels_flat[i]) and (
         labels_flat[i] == 0):
        match_num += 1
      if labels_flat[i] == 0:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

def LOC_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
      if (pred_flat[i] == labels_flat[i]) and (
         labels_flat[i] == 1):
        match_num += 1
      if labels_flat[i] == 1:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

model.zero_grad()
```

```
final_info = {}

for epoch_i in range(0, epochs):

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')

    t0 = time.time()
    total_loss = 0
    model.train()

    for step, batch in enumerate(train_dataloader):
        if step % 500 == 0 and not step == 0:
            elapsed = format_time(time.time() - t0)
            print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format(step, len(train_dataloader), elapsed))

        batch = tuple(t.to(device) for t in batch)
        b_input_ids, b_input_mask, b_labels = batch
        outputs = model(b_input_ids,
                        token_type_ids=None,
                        attention_mask=b_input_mask,
                        labels=b_labels)

        loss = outputs[0]
        total_loss += loss.item()
        loss.backward()
```

```
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            optimizer.step()
            scheduler.step()
            model.zero_grad()

        avg_train_loss = total_loss / len(train_dataloader)

        print("")
        print("  Average training loss: {0:.2f}".format(avg_train_loss))
        print("  Training epcoh took: {:}".format(format_time(time.time() - t0)))

        print("")
        print("Running Validation...")

        t0 = time.time()
        model.eval()

        eval_loss, eval_accuracy = 0, 0
        nb_eval_steps, nb_eval_examples = 0, 0
        SRC_nb_eval_steps, SRC_eval_accuracy = 0, 0
        LOC_nb_eval_steps, LOC_eval_accuracy = 0, 0

        epoch_info = {}

        for batch in test_dataloader:
            batch = tuple(t.to(device) for t in batch)
            b_input_ids, b_input_mask, b_labels = batch
```

```
        with torch.no_grad():
            outputs = model(b_input_ids,
                            token_type_ids=None,
                            attention_mask=b_input_mask)

        logits = outputs[0]
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()

        tmp_eval_accuracy = flat_accuracy(logits, label_ids)
        eval_accuracy += tmp_eval_accuracy
        nb_eval_steps += 1

        SRC_tmp_eval_accuracy = SRC_flat_accuracy(logits, label_ids)
        SRC_eval_accuracy += SRC_tmp_eval_accuracy
        SRC_nb_eval_steps += 1

        LOC_tmp_eval_accuracy = LOC_flat_accuracy(logits, label_ids)
        LOC_eval_accuracy += LOC_tmp_eval_accuracy
        LOC_nb_eval_steps += 1

    print("  Accuracy: {0:.2f}".format(eval_accuracy/nb_eval_steps))
    print("  Validation took: {:}".format(format_time(time.time() - t0)))
    print("")
    print("  Detail accuracy  ")
```

```
    print("  SRC_Accuracy: {0:.2f}".format(SRC_eval_accuracy/SRC_nb_eval_steps))
    print("  LOC_Accuracy: {0:.2f}".format(LOC_eval_accuracy/LOC_nb_eval_steps))

    epoch_info["Total"] = round(eval_accuracy/nb_eval_steps,3)
    epoch_info["Loss"] = round(avg_train_loss,3)
    epoch_info["SRC"] = round(SRC_eval_accuracy/SRC_nb_eval_steps,3)
    epoch_info["LOC"] = round(LOC_eval_accuracy/LOC_nb_eval_steps,3)

    final_info["epoch"+str(epoch_i)] = epoch_info

    model.eval()
    test_input_ids = []
    test_input_mask = []
    test_labels = []

    num = 0
    for step, batch in enumerate(test_data):
      batch = tuple(t.to(device) for t in batch)

      b_input_ids, b_input_mask, b_labels = batch
      input_ids_arr = []
      input_mask_arr = []

      for i in range(0,len(b_input_ids)):
        input_ids_arr.append(int(b_input_ids[i]))
        input_mask_arr.append(int(b_input_mask[i]))
```

```
  test_input_ids.append(input_ids_arr)
  test_input_mask.append(input_mask_arr)
  test_labels.append(int(b_labels))

test_input_ids = torch.tensor(test_input_ids)
test_input_mask = torch.tensor(test_input_mask)
test_labels = test_labels

test_input_ids = test_input_ids.to(device)
test_input_mask = test_input_mask.to(device)

with torch.no_grad():
    outputs = model(test_input_ids,
                    token_type_ids=None,
                    attention_mask=test_input_mask)

sentence_vecs_sum = outputs[0]

sentence_array = []
for i in range(0,len(sentence_vecs_sum)):
  each_array = []
  for j in range(0,len(sentence_vecs_sum[i])):
    each_array.append(float(sentence_vecs_sum[i][j]))
  sentence_array.append(each_array)

initial_df = pd.DataFrame(sentence_array)

from sklearn.manifold import TSNE
```

```
                tsne = TSNE(n_components=2, random_state=0)
                tsne_obj= tsne.fit_transform(initial_df)

                tsne_df = pd.DataFrame({'X':tsne_obj[:,0],'Y':
                    tsne_obj[:,1],'Label':test_labels})

                import numpy as np
                import pandas as pd
                from plotnine import *

                print("")
                print("  Network visualization  ")
                print(ggplot(tsne_df, aes(x='X', y='Y')) +
                    geom_point(aes(colour = 'Label')))

            print("")
            print("Training complete!")
            print("")
            print("Final result is below!")
            print(final_info)

        elif self.postposition == "Lo":

            def flat_accuracy(preds, labels):
                pred_flat = np.argmax(preds, axis=1).flatten()
                labels_flat = labels.flatten()
                return np.sum(pred_flat == labels_flat) / len(
                    labels_flat)

            def FNS_flat_accuracy(preds, labels):
                pred_flat = np.argmax(preds, axis=1).flatten()
```

```
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
      if (pred_flat[i] == labels_flat[i]) and (
         labels_flat[i] == 0):
        match_num += 1
      if labels_flat[i] == 0:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

def INS_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
      if (pred_flat[i] == labels_flat[i]) and (
         labels_flat[i] == 1):
        match_num += 1
      if labels_flat[i] == 1:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

def DIR_flat_accuracy(preds, labels):
```

```
        pred_flat = np.argmax(preds, axis=1).flatten()
        labels_flat = labels.flatten()
        match_num = 0
        func_num = 0
        for i in range(0,len(pred_flat)):
          if (pred_flat[i] == labels_flat[i]) and (
              labels_flat[i] == 2):
            match_num += 1
          if labels_flat[i] == 2:
            func_num += 1
        if match_num == 0 or func_num == 0:
          return 0
        else:
          return match_num / func_num

    def EFF_flat_accuracy(preds, labels):
        pred_flat = np.argmax(preds, axis=1).flatten()
        labels_flat = labels.flatten()
        match_num = 0
        func_num = 0
        for i in range(0,len(pred_flat)):
          if (pred_flat[i] == labels_flat[i]) and (
              labels_flat[i] == 3):
            match_num += 1
          if labels_flat[i] == 3:
            func_num += 1
        if match_num == 0 or func_num == 0:
          return 0
        else:
          return match_num / func_num
```

```
def CRT_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
      if (pred_flat[i] == labels_flat[i]) and (
         labels_flat[i] == 4):
        match_num += 1
      if labels_flat[i] == 4:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num

def LOC_flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    match_num = 0
    func_num = 0
    for i in range(0,len(pred_flat)):
      if (pred_flat[i] == labels_flat[i]) and (
         labels_flat[i] == 5):
        match_num += 1
      if labels_flat[i] == 5:
        func_num += 1
    if match_num == 0 or func_num == 0:
      return 0
    else:
      return match_num / func_num
```

```

model.zero_grad()
final_info = {}

for epoch_i in range(0, epochs):

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')

    t0 = time.time()
    total_loss = 0
    model.train()

    for step, batch in enumerate(train_dataloader):
        if step % 500 == 0 and not step == 0:
            elapsed = format_time(time.time() - t0)
            print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format(step, len(train_dataloader), elapsed))

        batch = tuple(t.to(device) for t in batch)
        b_input_ids, b_input_mask, b_labels = batch
        outputs = model(b_input_ids,
                        token_type_ids=None,
                        attention_mask=b_input_mask,
                        labels=b_labels)

        loss = outputs[0]
```

```
            total_loss += loss.item()
            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.
                parameters(), 1.0)
            optimizer.step()
            scheduler.step()
            model.zero_grad()

        avg_train_loss = total_loss / len(
            train_dataloader)

        print("")
        print("  Average training loss: {0:.2f}".format
            (avg_train_loss))
        print("  Training epcoh took: {:}".format(
            format_time(time.time() - t0)))

        print("")
        print("Running Validation...")

        t0 = time.time()
        model.eval()

        eval_loss, eval_accuracy = 0, 0
        nb_eval_steps, nb_eval_examples = 0, 0
        FNS_nb_eval_steps, FNS_eval_accuracy = 0, 0
        INS_nb_eval_steps, INS_eval_accuracy = 0, 0
        DIR_nb_eval_steps, DIR_eval_accuracy = 0, 0
        EFF_nb_eval_steps, EFF_eval_accuracy = 0, 0
        CRT_nb_eval_steps, CRT_eval_accuracy = 0, 0
        LOC_nb_eval_steps, LOC_eval_accuracy = 0, 0
```

```

        epoch_info = {}

        for batch in test_dataloader:
            batch = tuple(t.to(device) for t in batch)
            b_input_ids, b_input_mask, b_labels = batch
            with torch.no_grad():
                outputs = model(b_input_ids,
                                token_type_ids=None,
                                attention_mask=
                                   b_input_mask)

            logits = outputs[0]

            logits = logits.detach().cpu().numpy()
            label_ids = b_labels.to('cpu').numpy()

            tmp_eval_accuracy = flat_accuracy(logits,
               label_ids)
            eval_accuracy += tmp_eval_accuracy
            nb_eval_steps += 1

            FNS_tmp_eval_accuracy = FNS_flat_accuracy(
               logits, label_ids)
            FNS_eval_accuracy += FNS_tmp_eval_accuracy
            FNS_nb_eval_steps += 1

            INS_tmp_eval_accuracy = INS_flat_accuracy(
               logits, label_ids)
            INS_eval_accuracy += INS_tmp_eval_accuracy
            INS_nb_eval_steps += 1
```

```

            DIR_tmp_eval_accuracy = DIR_flat_accuracy(logits, label_ids)
            DIR_eval_accuracy += DIR_tmp_eval_accuracy
            DIR_nb_eval_steps += 1

            EFF_tmp_eval_accuracy = EFF_flat_accuracy(logits, label_ids)
            EFF_eval_accuracy += EFF_tmp_eval_accuracy
            EFF_nb_eval_steps += 1

            CRT_tmp_eval_accuracy = CRT_flat_accuracy(logits, label_ids)
            CRT_eval_accuracy += CRT_tmp_eval_accuracy
            CRT_nb_eval_steps += 1

            LOC_tmp_eval_accuracy = LOC_flat_accuracy(logits, label_ids)
            LOC_eval_accuracy += LOC_tmp_eval_accuracy
            LOC_nb_eval_steps += 1

        print("  Accuracy: {0:.2f}".format(eval_accuracy/nb_eval_steps))
        print("  Validation took: {:}".format(format_time(time.time() - t0)))
        print("")
        print("  Detail accuracy  ")
        print("  FNS_Accuracy: {0:.2f}".format(FNS_eval_accuracy/FNS_nb_eval_steps))
        print("  INS_Accuracy: {0:.2f}".format(INS_eval_accuracy/INS_nb_eval_steps))
```

```
            print("  DIR_Accuracy: {0:.2f}".format(
                DIR_eval_accuracy/DIR_nb_eval_steps))
            print("  EFF_Accuracy: {0:.2f}".format(
                EFF_eval_accuracy/EFF_nb_eval_steps))
            print("  CRT_Accuracy: {0:.2f}".format(
                CRT_eval_accuracy/CRT_nb_eval_steps))
            print("  LOC_Accuracy: {0:.2f}".format(
                LOC_eval_accuracy/LOC_nb_eval_steps))

            epoch_info["Total"] = round(eval_accuracy/
                nb_eval_steps,3)
            epoch_info["Loss"] = round(avg_train_loss,3)
            epoch_info["FNS"] = round(FNS_eval_accuracy/
                FNS_nb_eval_steps,3)
            epoch_info["INS"] = round(INS_eval_accuracy/
                INS_nb_eval_steps,3)
            epoch_info["DIR"] = round(DIR_eval_accuracy/
                DIR_nb_eval_steps,3)
            epoch_info["EFF"] = round(EFF_eval_accuracy/
                EFF_nb_eval_steps,3)
            epoch_info["CRT"] = round(CRT_eval_accuracy/
                CRT_nb_eval_steps,3)
            epoch_info["LOC"] = round(LOC_eval_accuracy/
                LOC_nb_eval_steps,3)

            final_info["epoch"+str(epoch_i)] = epoch_info

            model.eval()
            test_input_ids = []
            test_input_mask = []
            test_labels = []
```

```
num = 0
for step, batch in enumerate(test_data):
  batch = tuple(t.to(device) for t in batch)
  b_input_ids, b_input_mask, b_labels = batch
  input_ids_arr = []
  input_mask_arr = []

  for i in range(0,len(b_input_ids)):
    input_ids_arr.append(int(b_input_ids[i]))
    input_mask_arr.append(int(b_input_mask[i]))

  test_input_ids.append(input_ids_arr)
  test_input_mask.append(input_mask_arr)
  test_labels.append(int(b_labels))

test_input_ids = torch.tensor(test_input_ids)
test_input_mask = torch.tensor(test_input_mask)
test_labels = test_labels
test_input_ids = test_input_ids.to(device)
test_input_mask = test_input_mask.to(device)

with torch.no_grad():
    outputs = model(test_input_ids,
                    token_type_ids=None,
                    attention_mask=
                       test_input_mask)

sentence_vecs_sum = outputs[0]

sentence_array = []
```

```
        for i in range(0,len(sentence_vecs_sum)):
          each_array = []
          for j in range(0,len(sentence_vecs_sum[i])):
            each_array.append(float(sentence_vecs_sum[i][j]))
          sentence_array.append(each_array)
        initial_df = pd.DataFrame(sentence_array)
        from sklearn.manifold import TSNE
        tsne = TSNE(n_components=2, random_state=0)
        tsne_obj= tsne.fit_transform(initial_df)
        tsne_df = pd.DataFrame({'X':tsne_obj[:,0],'Y':tsne_obj[:,1],'Label':test_labels})

        import numpy as np
        import pandas as pd
        from plotnine import *
        print("")
        print("  Network visualization  ")
        print(ggplot(tsne_df, aes(x='X', y='Y')) + geom_point(aes(colour = 'Label')))

    print("")
    print("Training complete!")
    print("")
    print("Final result is below!")
    print(final_info)

model.save_pretrained('drive/My Drive/BERT/SM/KoBERT/Postposition/Model/')
tokenizer.save_pretrained('drive/My Drive/BERT/SM/KoBERT/Postposition/Model/')
```

Appendix D

Code for the first visualization system (i.e., PostEmbedding)

The following script is the code that I used to develop the first visualization system (i.e., PostEmbedding).

Listing D.1: JavaScript code for developing PostEmbedding

```

<!DOCTYPE html>
<html>
    <head>
    <title>PostEmbedding</title><!--<link rel="stylesheet" href="./stylesheets/bubble_style.css">-->
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <script src="./javascripts/d3.v3.min.js" charset="utf-8"></script>
    <script src="./javascripts/d3.v4.js" charset="utf-8"></script>
    <script src="./javascripts/jquery-1.12.0.min.js" charset="utf-8"></script>
```

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<!--<link rel="stylesheet" href="./stylesheets/PostVis.css">-->

<script src="./Data/concordancedata.js" charset="utf-8"></script>
<script src="./Data/DSMsdata.js" charset="utf-8"></script>
<script src="./Data/Networkdata.js" charset="utf-8"></script>

<style>
body {
    margin: 0;
}

#header{
    position:relative;
    width:100%;
    height:50px;
    background-color:#00cec9;
    background-clip: content-box;
}

#left {
    position:relative;
    float:left;
    overflow:hidden;
```

```
        width:14%;
        height:920px;
        padding: 0.5%;
        background-color:whitesmoke;
        background-clip: content-box;
    }

    #left_top {
        position:relative;
        float:left;
        overflow:hidden;
        width:100%;
        height:475px;
        padding: 0.5%;
        background-clip: content-box;
    }

    #left_bottom {
        position:relative;
        float:left;
        overflow:hidden;
        width:100%;
        height:435px;
        padding: 0.5%;
        background-clip: content-box;
    }


    #section {
        position:relative;
        float:left;
```

```
    overflow:hidden;
    width:67%;
    height:920px;
    padding: 0.5%;
    background-clip: content-box;
}

#section_top {
    position:relative;
    float:left;
    overflow:hidden;
    width:100%;
    height:620px;
    padding-right: : 0.5%;
    padding-top: 0.5%;
    padding-left: 0.5%;
    background-clip: content-box;
}

#section_bottom {
    position:relative;
    float:left;
    overflow:hidden;
    width:100%;
    height:295px;
    padding-right: : 0.5%;
    padding-bottom: 0.5%;
    padding-left: 0.5%;
    background-clip: content-box;
}

```

```
#right {
    position:relative;
    float:right;
    overflow:hidden;
    width:19%;
    height:920px;
    padding: 0.5%;
    background-color:whitesmoke;
    background-clip: content-box;
}

#right_top {
    position:relative;
    float:left;
    overflow:hidden;
    width:100%;
    height:405px;
    padding: 0.5%;
    background-clip: content-box;
}


#right_bottom {
    position:relative;
    float:left;
    overflow:hidden;
    width:100%;
    height:505px;
    padding: 0.5%;
    background-clip: content-box;
}
```

```css

p#left_option {
    text-align: left;
    font-family: Open Sans;
    font-size: 1.5em;
    color: #666666;
    font-weight: bold;
    padding-top: 4%;
    padding-bottom: 4%;
    padding-left: 5%;
    margin: 0;
}

p#right_option {
    text-align: left;
    font-family: Open Sans;
    font-size: 1.6em;
    color: #666666;
    font-weight: bold;
    padding-top: 4%;
    padding-bottom: 4%;
    padding-left: 5%;
    margin: 0;
}

p#section_top_p {
    font-family: Open Sans;
    font-size: 1.5em;
    color: #666666;
    font-weight: bold;
    padding-left: 1%;
```

```
        margin: 0;
    }

    p#section_bottom_p {
        font-family: Open Sans;
        font-size: 1.5em;
        color: #666666;
        font-weight: bold;
        padding-left: 1%;
        margin: 0;
    }


    p#header_p_left {
        position: relative;
        text-align: left;
        font-family: Open Sans;
        font-size: 2em;
        color: white;
        font-weight: bold;
        padding-top: 0.5%;
        padding-bottom: 0.5%;
        padding-left: 1%;
        margin: 0;
    }

    p#header_p_right {
        position: relative;
        text-align: right;
        font-family: Open Sans;
        font-size: 2em;
```

```
    color: white;
    font-weight: bold;
    padding-top: 0.5%;
    padding-bottom: 0.5%;
    padding-right: 2%;
    margin: 0;
}

p#footer_p{
    font-size: 1em;
}

a#header_a{
    font-family: Open Sans;
    color: white;
    font-weight: bold;
    cursor: pointer;
}

a#footer_a{
    font-family: Open Sans;
    color: #f3c623;
    font-weight: bold;
    cursor: pointer;
}

#header_left{
    float:left;
    width:49%;
    height:50px;
    padding-top: 0.1%;
```

```
        background-clip: content-box;
    }

    #header_right{
        float:right;
        width:49%;
        height:50px;
        padding-top: 0.1%;
        background-clip: content-box;
    }

    #footer {
        height:20px;
        text-align: center;
        color: white;
        background-color:#717171;
        clear:both;
    }

    select#op_postposition { /*text-align-last:center;*/
       width: 90%;
       height: 30px;
       font-size: 17px;
       border-radius: 3px;
       position: relative;
       left:5%;
       background: white;
       cursor: pointer;
    }

    select#op_function { /*text-align-last:center;*/
```

```
        width: 90%;
        height: 30px;
        font-size: 17px;
        border-radius: 3px;
        position: relative;
        left:5%;
        background: white;
        cursor: pointer;
    }

    select#op_method { /*text-align-last:center;*/
        width: 90%;
        height: 30px;
        font-size: 17px;
        border-radius: 3px;
        position: relative;
        left:5%;
        background: white;
        cursor: pointer;
    }

    select#op_window {
        width: 90%;
        height: 30px;
        font-size: 17px;
        border-radius: 3px;
        position: relative;
        left:5%;
        background: white;
        cursor: pointer;
    }
```

```
input#onoff {
  cursor: pointer;
  position: relative;
  font-size: 14px;
  left:8%;
}


select#op_node_size {
   width: 90%;
   height: 30px;
   font-size: 17px;
   border-radius: 3px;
   position: relative;
   left:5%;
   background: white;
   cursor: pointer;
}

select#op_node_color {
   width: 90%;
   height: 30px;
   font-size: 17px;
   border-radius: 3px;
   position: relative;
   left:5%;
   background: white;
   cursor: pointer;
}

```

```
#container_leftbottom {
    border:2px solid #ccc;
    width:88%;
    height: 360px;
    position: absolute;
    left: 5%;
    overflow-y: scroll;
    overflow-x: auto;
    white-space: nowrap;
    border-radius: 10px;
    background: white;
}

#container_rightbottom {
    border:2px solid #ccc;
    width:92.5%;
    height: 430px;
    position: absolute;
    left: 5%;
    overflow-y: scroll;
    overflow-x: auto;
    white-space: nowrap;
    border-radius: 10px;
    background: white;
}

#container_section_bottom {
    border:2px solid #ccc;
    width:98%;
    height: 245px;
```

```
        position: absolute;
        left: 1%;
        overflow-y: scroll;
        overflow-x: auto;
        white-space: nowrap;
        border-radius: 8px;
        background: white;
    }

    .CB_leftbottom{
        cursor: pointer;
        position: relative;
        font-size: 14px;
        left:5%;
    }

    .networklinks line {
      stroke: #999;
      stroke-opacity: 0.6;
    }

    .networknodes circle {
      stroke: #666666;
      stroke-width: 1.5px;
    }

    table {
        font-size: 15px;
        max-height: 200px;
        overflow: auto;
        background: #ddd;
```

```
    box-shadow: 0 0 1px 1px #ddd;
}

th, td {
    background: #fff;
    padding: 8px 16px;
    padding-bottom: 0px;
}

thead th {
    background-color: #ddd;
    position: sticky;
    top: 0;
}

div.tooltip {
    position: absolute;
    text-align: left;
    padding: 5px;
    font-size: 17px;
    background-color: #efefef;
    border: solid 1px #cecece;
    border-radius: 8px;
    box-shadow: 0 3px 5px 0 #dfdfdf;
    pointer-events: none;
}

h5 {
    font-size: 15px;
}
```

```
g.networknodes text {
    font-size: 13px;
}

text#nodegroup.nodetext {
    font-size: 13px;
}

@media all and (min-width:951px) and (max-height: 1000px)
  { /*0.95*/
    #header{
        height:47.55px;
    }
    #header_left{
        height:47.55px;
    }

    #header_right{
        height:47.55px;
    }
    #left {
        height:874.00px;
    }
    #left_top {
        height:460.75px;
    }
    #left_bottom {
        height:403.75px;
    }
    p#left_option {
        font-size: 1.42em;
```

```
}
p#right_option {
    font-size: 1.52em;
}
p#section_top_p {
    font-size: 1.42em;
}
p#section_bottom_p {
    font-size: 1.42em;
}
p#header_p {
    font-size: 1.90em;
}
p#header_p_left {
    font-size: 1.9em;
}

p#header_p_right {
    font-size: 1.9em;
}

p#footer_p{
    font-size: 0.95em;
}
#section {
    height:874.92px;
}
#section_top {
    height:589.62px;
}
#section_bottom {
```

```
        height:280.25px;
    }
    #right {
        height:874.92px;
    }
    #right_top {
        height:380.75px;
    }
    #right_bottom {
        height:465.75px;
    }
    #footer {
        height:19px;
    }
    select#op_postposition {
        height: 28.53px;
        font-size: 16.16px;
    }
    select#op_function {
        height: 28.53px;
        font-size: 16.16px;
    }
    select#op_method {
        height: 28.53px;
        font-size: 16.16px;
    }
    select#op_window {
        height: 28.53px;
        font-size: 16.16px;
    }
    select#op_node_size {
```

```
        height: 28.53px;
        font-size: 16.16px;
    }
    select#op_node_color {
        height: 28.53px;
        font-size: 16.16px;
    }
    input#onoff {
        font-size: 13.30px;
    }
    #container_leftbottom {
        height: 342px;
    }
    #container_rightbottom {
        height: 420px;
    }
    #container_section_bottom {
        height: 232.75px;
    }
    div.tooltip {
        padding: 4.75px;
        font-size: 16.16px;
        border-radius: 7.60px;
    }
    .CB_leftbottom{
        font-size: 14.26px;
    }
    table {
        font-size: 14.26px;
    }
    img#header_img{
```

```
            width: 33.28px;
            height: 30.43px;
        }
        h5 {
            font-size: 14.26px;
        }
        g.networknodes text {
            font-size: 12.35px;
        }
        text#nodegroup.nodetext {
            font-size: 12.35px;
        }
    }
    @media all and (min-width:901px) and (max-height: 950px) {
        /*0.948*/
        #header{
            height:45.05px;
        }
        #header_left{
            height:45.05px;
        }

        #header_right{
            height:45.05px;
        }
        #left {
            height:828.55px;
        }
        #left_top {
            height:432.52px;
        }
```

```
#left_bottom {
    height:387.02px;
}
p#left_option {
    font-size: 1.35em;
}
p#right_option {
    font-size: 1.44em;
}
p#section_top_p {
    font-size: 1.35em;
}
p#section_bottom_p {
    font-size: 1.35em;
}
p#header_p {
    font-size: 1.80em;
}
p#header_p_left {
    font-size: 1.80em;
}

p#header_p_right {
    font-size: 1.80em;
}
p#footer_p{
    font-size: 0.9em;
}
#section {
    height:828.92px;
}
```

```
#section_top {
    height:558.62px;
}
#section_bottom {
    height:265.677px;
}
#right {
    height:828.92px;
}
#right_top {
    height:367.113px;
}
#right_bottom {
    height:432.433px;
}
#footer {
    height:18px;
}
select#op_postposition {
    height: 27.03px;
    font-size: 15.31px;
}
select#op_function {
    height: 27.03px;
    font-size: 15.31px;
}
select#op_method {
    height: 27.03px;
    font-size: 15.31px;
}
select#op_window {
```

```
        height: 27.03px;
        font-size: 15.31px;
    }
    select#op_node_size {
        height: 27.03px;
        font-size: 15.31px;
    }
    select#op_node_color {
        height: 27.03px;
        font-size: 15.31px;
    }
    input#onoff {
        font-size: 12.6px;
    }

    #container_leftbottom {
        height: 328.482px;
    }
    #container_rightbottom {
        height: 399.288px;
    }
    #container_section_bottom {
        height: 220.647px;
    }
    div.tooltip {
        padding: 4.50px;
        font-size: 15.31px;
        border-radius: 7.21px;
    }
    .CB_leftbottom{
        font-size: 13.51px;
```

```
        }
        table {
            font-size: 13.51px;
        }
        img#header_img{
            width: 31.53px;
            height: 28.83px;
        }
        h5 {
            font-size: 13.51px;
        }
        g.networknodes text {
            font-size: 11.71px;
        }
        text#nodegroup.nodetext {
            font-size: 11.71px;
        }
    }

    @media all and (min-width:819px) and (max-height: 900px) {
        /*0.909*/
        #header{
            height:40.95px;
        }
        #header_left{
            height:40.95px;
        }

        #header_right{
            height:40.95px;
        }
```

```
#left {
    height:753.15px;
}
#left_top {
    height:388.9px;
}
#left_bottom {
    height:356.05px;
}
p#left_option {
    font-size: 1.22em;
}
p#right_option {
    font-size: 1.31em;
}
p#section_top_p {
    font-size: 1.22em;
}
p#section_bottom_p {
    font-size: 1.22715em;
}
p#header_p {
    font-size: 1.63em;
}
p#header_p_left {
    font-size: 1.63em;
}

p#header_p_right {
    font-size: 1.63em;
}
```

```
        p#footer_p{
            font-size: 0.8em;
        }
        #section {
            height:753.56px;
        }
        #section_top {
            height:507.83px;
        }
        #section_bottom {
            height:241.5004px;
        }
        #right {
            height:753.56px;
        }
        #right_top {
            height:336.0691px;
        }
        #right_bottom {
            height:390.8982px;
        }
        #footer {
            height:16.4px;
        }
        select#op_postposition {
            height: 24.57px;
            font-size: 13.92px;
        }
        select#op_function {
            height: 24.57px;
            font-size: 13.92px;
```

```
    }
    select#op_method {
        height: 24.57px;
        font-size: 13.92px;
    }
    select#op_window {
        height: 24.57px;
        font-size: 13.92px;
    }
    select#op_node_size {
        height: 24.57px;
        font-size: 13.92px;
    }
    select#op_node_color {
        height: 24.57px;
        font-size: 13.92px;
    }
    input#onoff {
        font-size: 11.45px;
    }

    #container_leftbottom {
        height: 302.84px;
    }
    #container_rightbottom {
        height: 357.9498px;
    }
    #container_section_bottom {
        height: 200.5681px;
    }
    div.tooltip {
```

```
            padding: 4.09px;
            font-size: 13.92px;
            border-radius: 6.55px;
        }
        .CB_leftmiddle{
            font-size: 12.28px;
        }
        .CB_leftbottom{
            font-size: 12.28px;
        }
        table {
            font-size: 12.28px;
        }
        img#header_img{
            width: 28.66px;
            height: 26.21px;
        }
        h5 {
            font-size: 12.28px;
        }
        g.networknodes text {
            font-size: 10.64px;
        }
        text#nodegroup.nodetext {
            font-size: 10.64px;
        }
    }

    @media all and (min-width:701px) and (max-height: 818px) {
        /*0.856*/
        #header{
```

```
        height:35.05px;
    }
    #header_left{
        height:35.05px;
    }

    #header_right{
        height:35.05px;
    }
    #left {
        height:644.69px;
    }
    #left_top {
        height:325.89px;
    }
    #left_bottom {
        height:311.79px;
    }
    p#left_option {
        font-size: 1.05em;
    }
    p#right_option {
        font-size: 1.12em;
    }
    p#section_top_p {
        font-size: 1.05em;
    }
    p#section_bottom_p {
        font-size: 1.05em;
    }
    p#header_p {
```

```
        font-size: 1.40em;
    }
    p#header_p_left {
        font-size: 1.40em;
    }

    p#header_p_right {
        font-size: 1.40em;
    }
    p#footer_p{
        font-size: 0.7em;
    }
    #section {
        height:644.92px;
    }
    #section_top {
        height:434.62px;
    }
    #section_bottom {
        height:206.72px;
    }
    #right {
        height:644.92px;
    }
    #right_top {
        height:291.56px;
    }
    #right_bottom {
        height:331.12px;
    }
    #footer {
```

```
        height:14px;
    }
    select#op_postposition {
        height: 21.03px;
        font-size: 11.91px;
    }
    select#op_function {
        height: 21.03px;
        font-size: 11.91px;
    }
    select#op_method {
        height: 21.03px;
        font-size: 11.91px;
    }
    select#op_window {
        height: 21.03px;
        font-size: 11.91px;
    }
    select#op_node_size {
        height: 21.03px;
        font-size: 11.91px;
    }
    select#op_node_color {
        height: 21.03px;
        font-size: 11.91px;
    }
    input#onoff {
        font-size: 9.8px;
    }

    #container_leftbottom {
```

```
        height: 266.24px;
    }
    #container_rightbottom {
        height: 301.37px;
    }
    #container_section_bottom {
        height: 171.69px;
    }
    div.tooltip {
        padding: 3.50px;
        font-size: 11.91px;
        border-radius: 5.60px;
    }
    .CB_leftmiddle{
        font-size: 10.51px;
    }
    .CB_leftbottom{
        font-size: 10.51px;
    }
    table {
        font-size: 10.51px;
    }
    img#header_img{
        width: 24.53px;
        height: 22.43px;
    }
    h5 {
        font-size: 10.51px;
    }
    g.networknodes text {
        font-size: 9.11px;
```

```
    }
    text#nodegroup.nodetext {
        font-size: 9.11px;
    }
}

@media all and (min-width:450px) and (max-height: 700px) {
    /*0.642*/
    #header{
        height:22.5px;
    }
    #header_left{
        height:22.5px;
    }

    #header_right{
        height:22.5px;
    }
    #left {
        height:413.89px;
    }
    #left_top {
        height:200.9px;
    }
    #left_bottom {
        height:208.49px;
    }
    p#left_option {
        font-size: 0.675em;
    }
    p#right_option {
```

```
        font-size: 0.72em;
    }
    p#section_top_p {
        font-size: 0.675em;
    }
    p#section_bottom_p {
        font-size: 0.6741em;
    }
    p#header_p {
        font-size: 0.9em;
    }
    p#header_p_left {
        font-size: 0.9em;
    }

    p#header_p_right {
        font-size: 0.9em;
    }
    p#footer_p{
        font-size: 0.3em;
    }
    #section {
        height:414px;
    }
    #section_top {
        height:279px;
    }
    #section_bottom {
        height:132.7142px;
    }
    #right {
```

```
        height:414px;
    }
    #right_top {
        height:191.8039px;
    }
    #right_bottom {
        height:208.5866px;
    }
    #footer {
        height:9px;
    }
    select#op_postposition {
        height: 13.5px;
        font-size: 7.65px;
    }
    select#op_function {
        height: 13.5px;
        font-size: 7.65px;
    }
    select#op_method {
        height: 13.5px;
        font-size: 7.65px;
    }
    select#op_window {
        height: 13.5px;
        font-size: 7.65px;
    }
    select#op_node_size {
        height: 13.5px;
        font-size: 7.65px;
    }
```

```
        select#op_node_color {
            height: 13.5px;
            font-size: 7.65px;
        }
        input#onoff {
            font-size: 6.29px;
        }

        #container_leftbottom {
            height: 179.24px;
        }
        #container_rightbottom {
            height: 194.9495px;
        }
        #container_section_bottom {
            height: 110.225px;
        }
        div.tooltip {
            padding: 2.25px;
            font-size: 7.65px;
            border-radius: 3.6px;
        }
        .CB_leftmiddle{
            font-size: 6.75px;
        }
        .CB_leftbottom{
            font-size: 6.75px;
        }
        table {
            font-size: 6.75px;
        }
```

```
        img#header_img{
            width: 15.75px;
            height: 14.4px;
        }
        h5 {
            font-size: 6.75px;
        }
        g.networknodes text {
            font-size: 5.85px;
        }
        text#nodegroup.nodetext {
            font-size: 5.85px;
        }
    }
    </style>

    </head>
    <body>
        <div id="header">
            <div id="header_left">
                <p id="header_p_left" align="left">PostEmbedding</p>
            </div>
            <div id="header_right" align="right">
                <p id="header_p_right"><a id="header_a" href="https://github.com/seongmin-mun/VisualSystem/tree/master/Major/PostEmbedding">GitHub</a></p>
            </div>
        </div>
        <div id="left">
```

```
        <div id="left_top">
            <p id="left_option">Postposition</p>
            <select id="op_postposition">
                <option value="ey" selected="selected">-ey</option>
                <option value="eyse">-eyse</option>
                <option value="(u)lo">-(u)lo</option>
            </select>
            <p id="left_option">Method</p>
            <select id="op_method">
                <option value="ppmi_svd" selected="selected">PPMI & SVD</option>
                <option value="sgns">SGNS</option>
            </select>
            <p id="left_option">Context window size</p>
            <select id="op_window">
                <option value="window1" selected="selected">window 1</option>
                <option value="window2">window 2</option>
                <option value="window3">window 3</option>
                <option value="window4">window 4</option>
                <option value="window5">window 5</option>
                <option value="window6">window 6</option>
                <option value="window7">window 7</option>
                <option value="window8">window 8</option>
                <option value="window9">window 9</option>
                <option value="window10">window 10</option>
            </select>
            <p id="left_option">Node size</p>
            <select id="op_node_size">
```

```
                <option id="frequency_size" value="frequency" selected="selected">frequency</option>
                <option id="nomal_size" value="nomal">default</option>
            </select>
            <p id="left_option">Node color</p>
            <select id="op_node_color">
                <option id="class_color" value="pos" selected="selected">POS</option>
                <option id="nomal_color" value="nomal">default</option>
            </select>
            <p id="left_option">Text switch</p>
            <input class='CB_lefttop' type='checkbox' value='onoff' id='onoff' checked="checked"/><label class='CB_lefttop' style="padding-left: 13%;">On/Off</label>
        </div>
        <div id="left_bottom">
            <p id="left_option">Select POS</p>
            <div id="container_leftbottom">

            </div>
        </div>
    </div>
    <div id="section">
        <div id="section_top">
            <p id="section_top_p">Distributional semantic map with t-SNE</p>
        </div>
```

```
            <div id="section_bottom">
                <p id="section_bottom_p">Concordance table</p>
                <div id="container_section_bottom">
                    <table class="table" style="margin-bottom: 0px;">
                        <thead>
                            <tr>
                                <th>id</th>
                                <th>name</th>
                                <th>function</th>
                                <th>sentences</th>
                                <th>lexeme with POS</th>
                            </tr>
                        </thead>
                        <tbody id="concordancetable">
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
        <div id="right">

            <div id="right_top">
                <p id="right_option">Function</p>
                <select id="op_function">

                </select>
                <p id="right_option">Force directed graph</p>
            </div>
            <div id="right_bottom">
                <p id="right_option">Nearest words</p>
```

```
                <div id="container_rightbottom">
                    <table class="table" style="margin-bottom: 0px;">
                        <thead>
                            <tr>
                                <th>id</th>
                                <th>name</th>
                                <th>similarity</th>
                                <th>frequency</th>
                            </tr>
                        </thead>
                        <tbody id="similaritytable">
                        </tbody>
                    </table>
                </div>
            </div>
        </div>
        <div id="footer">
            <p id="footer_p">2020 - 2021, <a id="footer_a" href="https://seongmin-mun.github.io/MyWebsite/Seongmin/index.html">Seongmin Mun</a>. All rights reserved.<p>
        </div>
<script>
$(document).ready(function () {

var functionslist_ey = ['LOC','GOL','EFF','CRT','THM','INS','AGT','FNS'];
var functionsname_ey = ['Location','Goal','Effector','Criterion','Theme','Instrument','Agent','Final State'];]

```

```
var functionslist_eyse = ['SRC','LOC'];
var functionsname_eyse = ['Source','Location'];
//var functionslist_eyse_number_frame = [487,197];

var functionslist_lo = ['LOC','DIR','EFF','CRT','INS','FNS'];
var functionsname_lo = ['Location','Direction','Effector','
    Criterion','Instrument','Final State'];

function draw_op_function_after(post,list,name,name_kr){
//var idname_0 = post[0]+"_"+list[0];
var idname_0 = list[0];
$('#op_function').empty();
$("#op_function").append("<option id='"+idname_0+"' value='"+
    idname_0.toLowerCase()+"' selected='selected'>"+list[0]+" (
    "+name_kr[0]+", "+name[0]+")"+"</option>")
for (var i = 1; i < list.length ; i++) {
//var idname_i = post[i]+"_"+list[i];
var idname_i = list[i];
$("#op_function").append("<option id='"+idname_i+"' value='"+
    idname_i.toLowerCase()+"'>"+list[i]+" ("+name_kr[i]+", "+
    name[i]+")"+"</option>");
}
}

function op_function_change(){
var selected_postposition = $( "#op_postposition" ).val();
if (selected_postposition === "ey"){
draw_op_function_after("ey",functionslist_ey,functionsname_ey,
    functionsname_kr_ey)
} else if (selected_postposition === "eyse"){
```

```
draw_op_function_after("eyse",functionslist_eyse,functionsname_eyse,functionsname_kr_eyse)
} else if (selected_postposition === "(u)lo"){
draw_op_function_after("(u)lo",functionslist_lo,functionsname_lo,functionsname_kr_lo)
}
drawall();
}

var typeslist = ['NNG','NNP','NNB','NP','NR','VV','VA','MAG','MAJ','JKB'];
var typesname = ['Common Noun','Proper Noun','Bound Noun','Pronoun','Numeral','Verb','Adjective','General Adverb','Conjunctive Adverb','Adverbial Case Marker'];
var POS_name = ['NNG','NNP','NNB','NP','NR','VV','VA','MAG','MAJ','JKB'];
var POS_color = ['#4f4cb4','#003783','#6685c7','#7faded','#16a1c6','#ab1432','#6c039d','#1d5041','#4c9046','#5b2e90'];

for(var i = 0 ; i < typeslist.length; i++){
var color = ""
var currentPOS = typeslist[i]
if (currentPOS == POS_name[0]) {
color = POS_color[0]
} else if (currentPOS == POS_name[1]) {
color = POS_color[1]
} else if (currentPOS == POS_name[2]) {
color = POS_color[2]
} else if (currentPOS == POS_name[3]) {
color = POS_color[3]
} else if (currentPOS == POS_name[4]) {
```

```
color = POS_color[4]
} else if (currentPOS == POS_name[5]) {
color = POS_color[5]
} else if (currentPOS == POS_name[6]) {
color = POS_color[6]
} else if (currentPOS == POS_name[7]) {
color = POS_color[7]
} else if (currentPOS == POS_name[8]) {
color = POS_color[8]
} else if (currentPOS == POS_name[9]) {
color = POS_color[9]
}
$("#container_leftbottom").append("<input class='CB_leftbottom
    ' type='checkbox' value='"+typeslist[i]+"' id='
    CB_leftbottom_"+i+"' /> <label class='CB_leftbottom'><svg
    width='12' height='12'><rect width='11' height='11' rx='2'
    class='legendrect' style='fill:"+color+";opacity:0.9;'/>
    </svg> "+typeslist[i]+" ("+typesname_kr[i]+", "+typesname[i
    ]+")</label></br>");
}

function drawconcordance_table(data,post){
$('#concordancetable').empty();
var sentencedata = [];
if(post=="(u)lo"){
post = "lo";
}
for (var i = 0; i < data.length ; i++) {
if ((data[i].postposition === post)) {
sentencedata.push(data[i]);
}
```

```
}
for(var i = 0 ; i < sentencedata.length; i++){
for(var j = 0 ; j < sentencedata[i].sentences.length; j++){
$("#concordancetable").append("<tr style='padding: 0px;'><td>"+((i*40)+(j+1))+"</td><td>"+sentencedata[i].sentences[j].name+"</td><td>"+sentencedata[i].function.toUpperCase()+"</td><td>"+sentencedata[i].sentences[j].sentence+"</td><td>"+sentencedata[i].sentences[j].pos_sentence+"</td></tr>");
}
}
}

function checkbox() {

var data_checkbox = []
for(var i = 0; i < typeslist.length; i++){
if(checkedeachValue('CB_leftbottom_'+i)!=='null'){
data_checkbox.push(checkedeachValue('CB_leftbottom_'+i));
}
}
return data_checkbox;
}

function checkedeachValue(checkeddata){
var value;
var checkedValue = document.querySelector('#'+checkeddata+':checked');
if(checkedValue == null){
value = "null";
} else {
value = checkedValue.value;
```

```
}
return value;
}

var width = $(window).width()
var height = $(window).height()

var section_top_height = $("#section_top").height()

var svgSection = d3.select('#section_top').append('svg')
.attr('width', width * 0.655)
.attr('height', (section_top_height*0.935))
.call(d3.zoom().scaleExtent([0.5, 5]).on("zoom", function () {
svgSection.attr("transform", d3.event.transform)
}))
.append("g");

var right_top_height = $("#right_top").height()

var svgright_top = d3.select("#right_top")
.append("svg")
.attr("width", width * 0.172)
.attr("height", (right_top_height*0.63))

svgright_top.append("rect")
.attr("class", "svgright_rect")
.attr("x", width * 0.008)
.attr("y", 0)
.attr("width", (width * 0.162))
.attr("height",(right_top_height*0.63))
.attr("rx", 6)
```

```
.attr("ry", 6)
.attr("fill", "white")
.attr('stroke', '#C2C1C1')
.attr('stroke-width', '2')

var NodeGroup = svgSection.append("g");

var div_inner = d3.select("#section").append("div")
.attr("class", "tooltip")
.style("opacity", 0);

firstdrawdata();
op_function_change();
var originalsize = $("#nodegroup.nodetext").css("font-size");

d3.selectAll("#op_postposition").on("change",
    op_function_change);
d3.selectAll("#op_function").on("change", drawall);
d3.selectAll("#op_method").on("change", drawall);
d3.selectAll("#op_window").on("change", drawall);
d3.selectAll("#container_leftbottom").on("change", drawall);
d3.selectAll("#onoff").on("change", drawall);
d3.selectAll("#op_node_size").on("change", drawall);
d3.selectAll("#op_node_color").on("change", drawall);

function drawall(){
var selected_postposition = $( "#op_postposition" ).val();
var selected_function = $( "#op_function" ).val();
var selected_method = $( "#op_method" ).val();
var selected_window = $( "#op_window" ).val();
var selected_node_size = $( "#op_node_size" ).val();
```

```
var selected_node_color = $( "#op_node_color" ).val();
drawconcordance_table(sentence_concordance,
    selected_postposition)
var partofspeeches = checkbox();
changedrawdata(selected_postposition,selected_function,
    selected_method,selected_window,selected_node_size,
    selected_node_color,partofspeeches)
}

function firstdrawdata() {
var data = [];
for (var i = 0; i < network_info.length ; i++) {
if ((network_info[i].postposition === 'ey') && (network_info[i
    ].function === 'loc') && (network_info[i].method === '
    ppmi_svd') && (network_info[i].window === 'window1')) {
data.push(network_info[i]);
}
}

drawtable(data[0]);
drawnetwork(data[0]);
var map_data = [];
for (var i = 0; i < DSMs_info.length ; i++) {
if ((DSMs_info[i].postposition === 'ey') && (DSMs_info[i].
    method === 'ppmi_svd') && (DSMs_info[i].window === 'window1
    ')) {
for (var j = 0; j < DSMs_info[i].wordnet.length ; j++) {
var each = {
opacity_value: []
};
each.opacity_value.push(0.6);
```

```
var settings = $.extend({}, each, DSMs_info[i].wordnet[j]);
map_data.push(settings);
}
}
}

var w = width*0.6;
var h = section_top_height;
var padding = (section_top_height*0.12);

var xScale = d3.scale.linear()
.domain([d3.min(map_data, function(d) { return d.y; }), d3.max
    (map_data, function(d) { return d.x; })])
.range([0+padding, w-padding]);

var yScale = d3.scale.linear()
.domain([d3.min(map_data, function(d) { return d.y; }), d3.max
    (map_data, function(d) { return d.y; })])
.range([h-padding, 0+padding]);

NodeGroup.selectAll(".nodedot")
.data(map_data)
.enter()
.append("circle")
.attr("class", "nodedot")
.attr("id", "nodegroup")
.attr("cx", function (d) {
return xScale(d.x)
})
.attr("cy", function (d) {
return yScale(d.y)
```

```
})
.attr("r", function (d) {
if(d.pos=="JKB"){
return 10
} else {
var size = (d.frequency/30 * 4)
if (size <= 4) {
return 4
} else if (20 <= size) {
return 20
} else {
return size
}
}
})
.attr("fill", function (d) {
if (d.pos == POS_name[0]) {
return POS_color[0]
} else if (d.pos == POS_name[1]) {
return POS_color[1]
} else if (d.pos == POS_name[2]) {
return POS_color[2]
} else if (d.pos == POS_name[3]) {
return POS_color[3]
} else if (d.pos == POS_name[4]) {
return POS_color[4]
} else if (d.pos == POS_name[5]) {
return POS_color[5]
} else if (d.pos == POS_name[6]) {
return POS_color[6]
} else if (d.pos == POS_name[7]) {
```

```
return POS_color[7]
} else if (d.pos == POS_name[8]) {
return POS_color[8]
} else if (d.pos == POS_name[9]) {
return POS_color[9]
}
})
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", function (d) {
return d.opacity_value
})
.style("cursor", "pointer")
.on("mouseover", function (d) {
d3.select(this)
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", 1)
})
.on("mouseout", function (d) {
d3.select(this)
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", function (d) {
return d.opacity_value
});
})
.on("mouseenter", function (d) {
if(d.pos=="JKB"){
div_inner.transition()
.duration(200)
```

```
.style("opacity", 0.85);
div_inner.html("<strong>Selected word</strong><br/><h5>Name_kr
     : "+d.name_kr + "<h5/><h5>Name_eng : "  + d.name_eng + "
    <h5/><h5>POS : "  + d.pos_long+ "<h5/><h5>POS_kr : "  +
    d.pos_kr+ "<h5/><h5>POS_eng : "  + d.pos_eng + "<h5/><h5>
    Frequency : "  + d.frequency+"<h5/>")
.style("right", "20px")
.style("top", "20px");
} else {
div_inner.transition()
.duration(200)
.style("opacity", 0.85);
div_inner.html("<strong>Selected word</strong><br/><h5>Name_kr
     : "+d.name_kr + "<h5/><h5>Name_eng : "  + d.name_eng + "
    <h5/><h5>POS : "  + d.pos+ "<h5/><h5>POS_kr : "  + d.pos_kr
    + "<h5/><h5>POS_eng : "  + d.pos_eng + "<h5/><h5>Frequency
    : "  + d.frequency+"<h5/>")
.style("right", "20px")
.style("top", "20px");
}
})
.on("mouseleave", function () {
div_inner.transition()
.duration(500)
.style("opacity", 0);
});

NodeGroup.selectAll(".nodetext")
.data(map_data)
.enter()
.append("text")
```

```
.attr("class", "nodetext")
.attr("id", "nodegroup")
.text(function (d) {
if(d.pos=="JKB"){
return d.name_kr+"/"+d.name_eng+"/"+d.pos_long;
} else {
return d.name_kr+"/"+d.name_eng+"/"+d.pos;
}
})
.attr("x", function (d) {
return xScale(d.x) + 10
})
.attr("y", function (d) {
return yScale(d.y) + 4
})
.attr("font-family", "sans-serif")
.attr("fill", "rgb(51,51,51)")
.attr("opacity", function (d) {
return d.opacity_value
})
.style("cursor", "pointer")
.on("mouseover", function () {
d3.select(this)
.attr("opacity", 1);
})
.on("mouseout", function (d) {
d3.select(this)
.attr("opacity", function (d) {
 return d.opacity_value
});
})
```

```
.on("mouseenter", function (d) {
if(d.pos=="JKB"){
div_inner.transition()
.duration(200)
.style("opacity", 0.85);
div_inner.html("<strong>Selected word</strong><br/><h5>Name_kr
    : "+d.name_kr + "<h5/><h5>Name_eng : "  + d.name_eng + "
    <h5/><h5>POS : "  + d.pos_long+ "<h5/><h5>POS_kr : "  +
    d.pos_kr+ "<h5/><h5>POS_eng : "  + d.pos_eng + "<h5/><h5>
    Frequency : "  + d.frequency+"<h5/>")
.style("right", "20px")
.style("top", "20px");
} else {
div_inner.transition()
.duration(200)
.style("opacity", 0.85);
div_inner.html("<strong>Selected word</strong><br/><h5>Name_kr
    : "+d.name_kr + "<h5/><h5>Name_eng : "  + d.name_eng + "
    <h5/><h5>POS : "  + d.pos+ "<h5/><h5>POS_kr : "  + d.pos_kr
    + "<h5/><h5>POS_eng : "  + d.pos_eng + "<h5/><h5>Frequency
    : "  + d.frequency+"<h5/>")
.style("right", "20px")
.style("top", "20px");
}

})
.on("mouseleave", function () {
div_inner.transition()
.duration(500)
.style("opacity", 0);
});
```

```
}

function changedrawdata(selected_postposition,
    selected_function,selected_method,selected_window,
    selected_node_size,selected_node_color,partofspeeches) {

$('#similaritytable').empty();
svgright_top.selectAll(".networklinks").remove();
svgright_top.selectAll(".networknodes").remove();

var data = [];
for (var i = 0; i < network_info.length ; i++) {
if ((network_info[i].postposition === selected_postposition)
    && (network_info[i].function === selected_function) && (
    network_info[i].method === selected_method) && (
    network_info[i].window === selected_window)) {
data.push(network_info[i]);
}
}

drawtable(data[0]);
drawnetwork(data[0]);

var map_data = [];
for (var i = 0; i < DSMs_info.length ; i++) {
if ((DSMs_info[i].postposition === selected_postposition) && (
    DSMs_info[i].method === selected_method) && (DSMs_info[i].
    window === selected_window)) {
for (var j = 0; j < DSMs_info[i].wordnet.length ; j++) {
var each = {
opacity_value: []
```

```
};
if(partofspeeches.length > 0 == true){
var checked = false;
for(var k = 0; k < partofspeeches.length ; k++){
if(DSMs_info[i].wordnet[j].pos == partofspeeches[k]){
    checked = true;
}
}
if(checked == true){
  each.opacity_value.push(0.9);
} else {
  each.opacity_value.push(0.2);
}
} else if(partofspeeches.length > 0 == false){
each.opacity_value.push(0.6);
}
var settings = $.extend({}, each, DSMs_info[i].wordnet[j]);
map_data.push(settings);
}
}
}

var w = width*0.6;
var h = section_top_height;
var padding = (section_top_height*0.12);
var xScale = d3.scale.linear()
.domain([d3.min(map_data, function(d) { return d.y; }), d3.max
    (map_data, function(d) { return d.x; })])
.range([0+padding, w-padding]);
var yScale = d3.scale.linear()
```

```
.domain([d3.min(map_data, function(d) { return d.y; }), d3.max(map_data, function(d) { return d.y; })])
.range([h-padding, 0+padding]);
var circle = NodeGroup.selectAll(".nodedot")
.data(map_data);

circle.enter()
.append("circle")
.attr("class", "nodedot")
.attr("id", "nodegroup")
.attr("cx", function (d) {
return xScale(d.x)
})
.attr("cy", function (d) {
return yScale(d.y)
})
.attr("r", function (d){
if (selected_node_size === "nomal") {
return 4;
} else if (selected_node_size === "frequency") {
if(d.pos=="JKB"){
return 10
} else {
var size = (d.frequency/30 * 4)
if (size <= 4) {
  return 4
} else if (20 <= size) {
  return 20
} else {
  return size
}
```

```
}
}
})
.attr("fill", function (d){
if (selected_node_color === "nomal") {
return "rgb(51,51,51)"
} else if (selected_node_color === "pos") {
if (d.pos == POS_name[0]) {
  return POS_color[0]
} else if (d.pos == POS_name[1]) {
  return POS_color[1]
} else if (d.pos == POS_name[2]) {
  return POS_color[2]
} else if (d.pos == POS_name[3]) {
  return POS_color[3]
} else if (d.pos == POS_name[4]) {
  return POS_color[4]
} else if (d.pos == POS_name[5]) {
  return POS_color[5]
} else if (d.pos == POS_name[6]) {
  return POS_color[6]
} else if (d.pos == POS_name[7]) {
  return POS_color[7]
} else if (d.pos == POS_name[8]) {
  return POS_color[8]
} else if (d.pos == POS_name[9]) {
  return POS_color[9]
}
}
})
.attr("stroke", "black")
```

```
.attr("stroke-width", "1px")
.attr("opacity", function (d) {
return d.opacity_value
})
.style("cursor", "pointer");

circle.transition()
.duration(2000)
.attr("cx", function (d) {
return xScale(d.x)
})
.attr("cy", function (d) {
return yScale(d.y)
})
.attr("r", function (d){
if (selected_node_size === "nomal") {
return 4;
} else if (selected_node_size === "frequency") {
if(d.pos=="JKB"){
return 10
} else {
var size = (d.frequency/30 * 4)
if (size <= 4) {
  return 4
} else if (20 <= size) {
  return 20
} else {
  return size
}
}
}
```

```
})
.attr("fill", function (d){
if (selected_node_color === "nomal") {
return "rgb(51,51,51)"
} else if (selected_node_color === "pos") {
if (d.pos == POS_name[0]) {
  return POS_color[0]
} else if (d.pos == POS_name[1]) {
  return POS_color[1]
} else if (d.pos == POS_name[2]) {
  return POS_color[2]
} else if (d.pos == POS_name[3]) {
  return POS_color[3]
} else if (d.pos == POS_name[4]) {
  return POS_color[4]
} else if (d.pos == POS_name[5]) {
  return POS_color[5]
} else if (d.pos == POS_name[6]) {
  return POS_color[6]
} else if (d.pos == POS_name[7]) {
  return POS_color[7]
} else if (d.pos == POS_name[8]) {
  return POS_color[8]
} else if (d.pos == POS_name[9]) {
  return POS_color[9]
}
}
})
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", function (d) {
```

```
return d.opacity_value
})
.style("cursor", "pointer");

circle.exit().remove();

var text = NodeGroup.selectAll(".nodetext")
.data(map_data);

text.enter()
.append("text")
.attr("class", "nodetext")
.attr("id", "nodegroup")
.text(function (d) {
if(d.pos=="JKB"){
return d.name_kr+"/"+d.name_eng+"/"+d.pos_long;
} else {
return d.name_kr+"/"+d.name_eng+"/"+d.pos;
}
})
.attr("x", function (d) {
return xScale(d.x) + 10
})
.attr("y", function (d) {
return yScale(d.y) + 4
})
.attr("font-family", "sans-serif")
.attr("fill", "rgb(51,51,51)")
.attr("opacity", function (d) {
return d.opacity_value
})
```

```
.style("cursor", "pointer");

text.transition()
.duration(2000)
.text(function (d) {
if(d.pos=="JKB"){
return d.name_kr+"/"+d.name_eng+"/"+d.pos_long;
} else {
return d.name_kr+"/"+d.name_eng+"/"+d.pos;
}
})
.attr("x", function (d) {
return xScale(d.x) + 10
})
.attr("y", function (d) {
return yScale(d.y) + 4
})
.attr("font-family", "sans-serif")
.attr("fill", "rgb(51,51,51)")
.attr("opacity", function (d) {
return d.opacity_value
})
.style("cursor", "pointer");

text.exit().remove();

if(checkedeachValue("onoff")=='null'){
svgSection.selectAll(".nodetext").remove();
}
}

```

```
function drawtable_first(data){
$('#similaritytable').empty();
for(var i = 0 ; i < data.links.length; i++){
$("#similaritytable").append("<tr style='padding: 0px;'><td>"+
    i+"</td><td>"+data.links[i].target+"</td><td>"+data.links[i
    ].value+"</td><td>"+data.nodes[i].frequency+"</td></tr>");
}
}

function drawtable(data){
$('#similaritytable').empty();
console.log(typeof(data.links[0].target));

if(typeof(data.links[0].target)=="object"){
for(var i = 0 ; i < data.links.length; i++){
$("#similaritytable").append("<tr style='padding: 0px;'><td>"+
    i+"</td><td>"+data.links[i].target.id+"</td><td>"+
    data.links[i].value+"</td><td>"+data.nodes[i].frequency+"</
    td></tr>");
}
} else {
for(var i = 0 ; i < data.links.length; i++){
$("#similaritytable").append("<tr style='padding: 0px;'><td>"+
    i+"</td><td>"+data.links[i].target+"</td><td>"+data.links[i
    ].value+"</td><td>"+data.nodes[i].frequency+"</td></tr>");
}
}
}

function drawnetwork(data){

```

```
svgright_top.selectAll(".networklinks").remove();
svgright_top.selectAll(".networknodes").remove();

var simulation = d3.forceSimulation()
.force("link", d3.forceLink().id(function(d) { return d.id; })
    .distance(function (d) { return (right_top_height*0.828)
    *0.28}))
.force("charge", d3.forceManyBody())
.force("center", d3.forceCenter(width * 0.14 / 2, (
    right_top_height*0.63) / 2));

var link = svgright_top.append("g")
.attr("class", "networklinks")
.selectAll("line")
.data(data.links)
.enter().append("line")
.attr("stroke-width", function(d) { return Math.sqrt(d.value
    *6); });

var node = svgright_top.append("g")
.attr("class", "networknodes")
.selectAll("g")
.data(data.nodes)
.enter().append("g")

var circles = node.append("circle")
.attr("r", function(d) {
var size = d.frequency/2
if(15 < size){
return 15;
} else if (size < 4){
```

```
return 4;
} else {
return size;
}
})
.attr("fill", function (d) {
if (d.pos == POS_name[0]) {
return POS_color[0]
} else if (d.pos == POS_name[1]) {
return POS_color[1]
} else if (d.pos == POS_name[2]) {
return POS_color[2]
} else if (d.pos == POS_name[3]) {
return POS_color[3]
} else if (d.pos == POS_name[4]) {
return POS_color[4]
} else if (d.pos == POS_name[5]) {
return POS_color[5]
} else if (d.pos == POS_name[6]) {
return POS_color[6]
} else if (d.pos == POS_name[7]) {
return POS_color[7]
} else if (d.pos == POS_name[8]) {
return POS_color[8]
} else if (d.pos == POS_name[9]) {
return POS_color[9]
} else {
return '#C2C1C1'
}
})
.call(d3.drag()
```

```
.on("start", dragstarted)
.on("drag", dragged)
.on("end", dragended));

var lables = node.append("text")
.text(function(d) {
return d.id;//.split("/")[0]+"/"+d.id_eng;
})
.attr('x', 6)
.attr('y', 3)
.attr('opacity',0.4)
.on("mouseover", function () {
d3.select(this).attr("opacity", 1);
})
.on("mouseout", function (d) {
d3.select(this).attr("opacity", 0.4);
});

node.append("title")
.text(function(d) { return d.id; });

simulation.nodes(data.nodes)
.on("tick", ticked);

simulation.force("link")
.links(data.links);

function ticked() {
link.attr("x1", function(d) { return d.source.x; })
.attr("y1", function(d) { return d.source.y; })
.attr("x2", function(d) { return d.target.x; })
```

```
.attr("y2", function(d) { return d.target.y; });

node.attr("transform", function(d) {
return "translate(" + d.x + "," + d.y + ")";
})
}

function dragstarted(d) {
if (!d3.event.active) simulation.alphaTarget(0.3).restart();
d.fx = d.x;
d.fy = d.y;
}

function dragged(d) {
d.fx = d3.event.x;
d.fy = d3.event.y;
}

function dragended(d) {
if (!d3.event.active) simulation.alphaTarget(0);
d.fx = null;
d.fy = null;
}
}
})
</script>
</body>
</html>
```

Appendix E

Code for the second visualization system (i.e., PostBERT)

The following script is the code that I used to develop the second visualization system (i.e., PostBERT).

Listing E.1: JavaScript code for developing PostBERT

```
<!DOCTYPE html>
<html>
    <head>
    <title>PostBERT</title><!--<link rel="stylesheet" href="./
        stylesheets/bubble_style.css">-->
    <meta http-equiv="Content-Type" content="text/html;
        charset=utf-8">
    <script src="./javascripts/d3.v3.min.js" charset="utf-8">
        </script>
    <script src="./javascripts/d3.v4.js" charset="utf-8">
        </script>
    <script src="./javascripts/jquery-1.12.0.min.js" charset="
        utf-8"></script>
```

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<!--<link rel="stylesheet" href="./stylesheets/PostVis.css">-->

<script src="./Data/Madpdata.js" charset="utf-8"></script>
<script src="./Data/Sentencedata.js" charset="utf-8"></script>
<script src="./Data/Accuracydata.js" charset="utf-8"></script>
<script src="./Data/Clusterdata.js" charset="utf-8"></script>

<style>
body {
    margin: 0;
}

#header{
    position:relative;
    width:100%;
    height:50px;
    background-color:#f3c623;
    background-clip: content-box;
}

#left {
    position:relative;
```

```
        float:left;
        overflow:hidden;
        width:14%;
        height:920px;
        padding: 0.5%;
        background-color:whitesmoke;
        background-clip: content-box;
    }

    #left_top {
        position:relative;
        float:left;
        overflow:hidden;
        width:100%;
        height:85px;
        padding: 0.5%;
        background-clip: content-box;
    }

    #left_middle {
        position:relative;
        float:left;
        overflow:hidden;
        width:100%;
        height:250px;
        padding: 0.5%;
        background-clip: content-box;
    }

    #left_bottom {
        position:relative;
```

```
        float:left;
        overflow:hidden;
        width:100%;
        height:290px;
        padding: 0.5%;
        background-clip: content-box;
    }

    #left_bottom_bottom {
        position:relative;
        float:left;
        overflow:hidden;
        width:100%;
        height:355px;
        padding: 0.5%;
        background-clip: content-box;
    }


    #section {
        position:relative;
        float:left;
        overflow:hidden;
        width:57%;
        height:920px;
        padding: 0.5%;
        background-clip: content-box;
    }

    #section_top {
        position:relative;
```

```
        float:left;
        overflow:hidden;
        width:100%;
        height:795px;
        padding-right: : 0.5%;
        padding-top: 0.5%;
        padding-left: 0.5%;
        background-clip: content-box;
    }

    #section_bottom {
        position:relative;
        float:left;
        overflow:hidden;
        width:100%;
        height:120px;
        padding-right: : 0.5%;
        padding-bottom: 0.5%;
        padding-left: 0.5%;
        background-clip: content-box;
    }

    #section_bottom_left {
        position:relative;
        float:left;
        overflow:hidden;
        width:12%;
        height:80px;
        padding-right: : 0.5%;
        padding-bottom: 0.5%;
        padding-left: 0.5%;
```

```
    background-clip: content-box;
}

#section_bottom_right {
    position:relative;
    float:left;
    overflow:hidden;
    width:88%;
    height:80px;
    padding-right: : 0.5%;
    padding-bottom: 0.5%;
    padding-left: 0.5%;
    background-clip: content-box;
}

#right {
    position:relative;
    float:right;
    overflow:hidden;
    width:29%;
    height:920px;
    padding: 0.5%;
    background-color:whitesmoke;
    background-clip: content-box;
}

#right_top {
    position:relative;
    float:left;
    overflow:hidden;
    width:100%;
```

```
        height:300px;
        padding: 0.5%;
        background-clip: content-box;
    }

    #right_middle {
        position:relative;
        float:left;
        overflow:hidden;
        width:100%;
        height:300px;
        padding: 0.5%;
        background-clip: content-box;
    }


    #right_bottom {
        position:relative;
        float:left;
        overflow:hidden;
        width:100%;
        height:305px;
        padding: 0.5%;
        background-clip: content-box;
    }

    #header_left{
        float:left;
        width:49%;
        height:50px;
        padding-top: 0.1%;
```

```
    background-clip: content-box;
}

#header_right{
    float:right;
    width:49%;
    height:50px;
    padding-top: 0.1%;
    background-clip: content-box;
}

#footer {
    height:20px;
    text-align: center;
    color: white;
    background-color:#717171;
    clear:both;
}


p#left_option {
    text-align: left;
    font-family: Open Sans;
    font-size: 1.5em;
    color: #666666;
    font-weight: bold;
    padding-top: 4%;
    padding-bottom: 4%;
    padding-left: 5%;
    margin: 0;
}
```

```

p#right_option {
    text-align: left;
    font-family: Open Sans;
    font-size: 1.6em;
    color: #666666;
    font-weight: bold;
    padding-top: 4%;
    padding-bottom: 4%;
    padding-left: 5%;
    margin: 0;
}

p#section_top_p {
    font-family: Open Sans;
    font-size: 1.5em;
    color: #666666;
    font-weight: bold;
    padding-left: 1%;
    margin: 0;
}

p#section_bottom_p {
    font-family: Open Sans;
    font-size: 1.5em;
    color: #666666;
    font-weight: bold;
    padding-left: 1%;
    margin: 0;
}

```

```
p#header_p_left {
    position: relative;
    text-align: left;
    font-family: Open Sans;
    font-size: 2em;
    color: white;
    font-weight: bold;
    padding-top: 0.5%;
    padding-bottom: 0.5%;
    padding-left: 1%;
    margin: 0;
}

p#header_p_right {
    position: relative;
    text-align: right;
    font-family: Open Sans;
    font-size: 2em;
    color: white;
    font-weight: bold;
    padding-top: 0.5%;
    padding-bottom: 0.5%;
    padding-right: 2%;
    margin: 0;
}

p#footer_p{
    font-size: 1em;
}

a#header_a{
```

```
        font-family: Open Sans;
        color: white;
        font-weight: bold;
        cursor: pointer;
    }

    a#footer_a{
        font-family: Open Sans;
        color: #f3c623;
        font-weight: bold;
        cursor: pointer;
    }


    select#op_postposition { /*text-align-last:center;*/
       width: 90%;
       height: 30px;
       font-size: 17px;
       border-radius: 3px;
       position: relative;
       left:5%;
       background: white;
       cursor: pointer;
    }

    /*select#op_node_color {
       width: 90%;
       height: 30px;
       font-size: 17px;
       border-radius: 3px;
       position: relative;
```

```
    left:5%;
    background: white;
    cursor: pointer;
}*/

#container_leftmiddle {
    border:2px solid #ccc;
    width:88%;
    height: 200px;
    position: absolute;
    left: 5%;
    overflow-y: scroll;
    overflow-x: auto;
    white-space: nowrap;
    border-radius: 10px;
    background: white;
}

.CB_leftmiddle {
    cursor: pointer;
    position: relative;
    font-size: 14px;
    left:5%;
}


#container_leftbottom {
    border:2px solid #ccc;
    width:88%;
    height: 240px;
    position: absolute;
```

```
        left: 5%;
        overflow-y: scroll;
        overflow-x: auto;
        white-space: nowrap;
        border-radius: 10px;
        background: white;
    }

    .CB_leftbottom{
        cursor: pointer;
        position: relative;
        font-size: 14px;
        left:5%;
    }



    #play-button {
      position: absolute;
      top: 25%;
      background: #f08080;
      padding-right: 10px;
      border-radius: 3px;
      border: none;
      color: white;
      margin: 0;
      width: 80%;
      cursor: pointer;
      height: 40%;
      font: 13px sans-serif;
    }
```

```
#play-button:hover {
  background-color: #696969;
}

#play-button:active {
  background-color: #002657;
}


.ticks {
  font: 10px sans-serif;
}

.track,
.track-inset,
.track-overlay {
  stroke-linecap: round;
}

.track {
  stroke: #000;
  stroke-opacity: 0.3;
  stroke-width: 10px;
}

.track-inset {
  stroke: #dcdcdc;
  stroke-width: 8px;
}

```

```
.track-overlay {
  pointer-events: stroke;
  stroke-width: 50px;
  cursor: pointer;
}

.handle {
  fill: #fff;
  stroke: #000;
  stroke-opacity: 0.5;
  stroke-width: 1.25px;
}


div.tooltip {
    position: absolute;
    text-align: left;
    padding: 5px;
    font-size: 17px;
    background-color: #efefef;
    border: solid 1px #cecece;
    border-radius: 8px;
    box-shadow: 0 3px 5px 0 #dfdfdf;
    pointer-events: none;
}

div.epoch {
    position: absolute;
    text-align: right;
    padding: 5px;
    font-size: 40px;
```

```
    background-color: white;
    border: solid 1px white;
    border-radius: 8px;
    pointer-events: none;
}

#corBar:hover {
        fill: orange;
    }

#tooltip_top {
    position: absolute;
    width: 120px;
    height: auto;
    padding: 10px;
    background-color: white;
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
    border-radius: 10px;
    -webkit-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
    -moz-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
    box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
    pointer-events: none;
    font-size: 16px;
}

#tooltip_top.hidden {
    display: none;
}

#tooltip_top p {
```

```
        margin: 0;
        font-family: sans-serif;
        line-height: 20px;
    }


    #tooltip_middle {
        position: absolute;
        width: 120px;
        height: auto;
        padding: 10px;
        background-color: white;
        -webkit-border-radius: 10px;
        -moz-border-radius: 10px;
        border-radius: 10px;
        -webkit-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
        -moz-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
        box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
        pointer-events: none;
        font-size: 16px;
    }

    #tooltip_middle.hidden {
        display: none;
    }

    #tooltip_middle p {
        margin: 0;
        font-family: sans-serif;\
        line-height: 20px;
    }
```

```
.rightbottom_bar:hover {
    fill: #f08080;
}

#tooltip_bottom {
    position: absolute;
    width: 160px;
    height: auto;
    padding: 10px;
    background-color: white;
    -webkit-border-radius: 10px;
    -moz-border-radius: 10px;
    border-radius: 10px;
    -webkit-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
    -moz-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
    box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
    pointer-events: none;
    opacity: 0.7;
}

#tooltip_bottom.hidden {
    display: none;
}

#tooltip_bottom p {
    margin: 0;
    font-family: sans-serif;
    font-size: 16px;
    line-height: 20px;
}
```

```

p.tooltip_topdiv{
    font-size: 16px;
}
p.tooltip_middlediv{
    font-size: 16px;
}

h5 {
    font-size: 15px;
}


@media all and (min-width:951px) and (max-height: 1000px)
   { /*0.95*/
    #header{
        height:47.55px;
    }
    #header_left{
        height:47.55px;
    }

    #header_right{
        height:47.55px;
    }

    #left {
        height:874px;
    }

    #left_top {
```

```
        height:80.75px;
    }

    #left_middle {
        height:237.5px;
    }

    #left_bottom {
        height:275.5px;
    }

    #left_bottom_bottom {
        height:337.25px;
    }
    #section {
        height:874px;
    }

    #section_top {
        height:755.25px;
    }

    #section_bottom {
        height:114px;
    }

    #section_bottom_left {
        height:76px;
    }

    #section_bottom_right {
```

```
        height:76px;
    }

    #right {
        height:874px;
    }

    #right_top {
        height:285px;
    }

    #right_middle {
        height:285px;
    }
    #right_bottom {
        height:289.75px;
    }

    #footer {
        height:19px;
    }

    p#left_option {
        font-size: 1.425em;
    }
    p#section_top_p {
        font-size: 1.425em;
    }
    p#section_bottom_p {
        font-size: 1.425em;
    }
```

```
p#right_option {
    font-size: 1.52em;
}
p#header_p_left {
    font-size: 1.9em;
}
p#header_p_right {
    font-size: 1.9em;
}
p#footer_p{
    font-size: 0.95em;
}
select#op_postposition {
   height: 28.5px;
   font-size: 16.15px;
}
#container_leftmiddle {
    height: 190px;
    border-radius: 9.5px;
}
#container_leftbottom {
    height: 228px;
    border-radius: 9.5px;
}
.CB_leftmiddle {
    font-size: 13.3px;
}

.CB_leftbottom{
    font-size: 13.3px;]
}
```

```
        #play-button {
          font: 12.35px sans-serif;
        }
        .ticks {
          font: 9.5px sans-serif;
        }
        .track {
          stroke-width: 9.5px;
        }
        .track-inset {
          stroke-width: 7.6px;
        }
        .track-overlay {
          stroke-width: 47.5px;
        }
        div.tooltip {
            font-size: 16.15px;
        }
        div.epoch {
            font-size: 38px;
        }
        #tooltip_top {
            width: 114px;
            font-size: 15.2px;
        }
        #tooltip_middle {
            width: 114px;
            font-size: 15.2px;
        }
        #tooltip_bottom {
            width: 152px;
```

```
        }
        #tooltip_bottom p {
            font-size: 15.2px;
        }
        h5 {
            font-size: 14.25px;
        }

        text.rightbottom_text {
            font-size: 19px;
        }
        text.rangetext{
            font-size: 19.95px;
        }

    }


    @media all and (min-width:901px) and (max-height: 950px) {
        /*0.90*/
        #header{
            height:45px;
        }
        #header_left{
            height:45px;
        }

        #header_right{
            height:45px;
        }

```

```
        #left {
            height:828px;
        }

        #left_top {
            height:76.5px;
        }

        #left_middle {
            height:225px;
        }

        #left_bottom {
            height:261px;
        }

        #left_bottom_bottom {
            height:319.5px;
        }
        #section {
            height:828px;
        }

        #section_top {
            height:715.5px;
        }

        #section_bottom {
            height:108px;
        }

```

```
#section_bottom_left {
    height:72px;
}

#section_bottom_right {
    height:72px;
}

#right {
    height:828px;
}

#right_top {
    height:270px;
}

#right_middle {
    height:270px;
}


#right_bottom {
    height:274.5px;
}

#footer {
    height:18px;
}

p#left_option {
    font-size: 1.35em;
```

```
        }
        p#section_top_p {
            font-size: 1.35em;
        }
        p#section_bottom_p {
            font-size: 1.35em;
        }
        p#right_option {
            font-size: 1.44em;
        }
        p#header_p_left {
            font-size: 1.8em;
        }
        p#header_p_right {
            font-size: 1.8em;
        }
        p#footer_p{
            font-size: 0.9em;
        }
        select#op_postposition {
           height: 27px;
           font-size: 15.3px;
        }
        #container_leftmiddle {
            height: 180px;
            border-radius: 9px;
        }
        #container_leftbottom {
            height: 216px;
            border-radius: 9px;
        }
```

```
.CB_leftmiddle {
    font-size: 12.6px;
}

.CB_leftbottom{
    font-size: 12.6px;]
}
#play-button {
  font: 11.7px sans-serif;
}
.ticks {
  font: 9px sans-serif;
}
.track {
  stroke-width: 9px;
}
.track-inset {
  stroke-width: 7.2px;
}
.track-overlay {
  stroke-width: 45px;
}
div.tooltip {
    font-size: 15.3px;
}
div.epoch {
    font-size: 36px;
}
#tooltip_top {
    width: 108px;
    font-size: 14.4px;
```

```
        }
        #tooltip_middle {
            width: 108px;
            font-size: 14.4px;
        }
        #tooltip_bottom {
            width: 144px;
        }
        #tooltip_bottom p {
            font-size: 14.4px;
        }
        h5 {
            font-size: 13.5px;
        }

        text.rightbottom_text {
            font-size: 18px;
        }
        text.rangetext{
            font-size: 18.9px;
        }
    }


    @media all and (min-width:819px) and (max-height: 900px) {
        /*0.82*/
        #header{
            height:40.95px;
        }
        #header_left{
            height:40.95px;
```

```
}

#header_right{
    height:40.95px;
}
#left {
    height:754.4px;
}

#left_top {
    height:69.7px;
}

#left_middle {
    height:205px;
}

#left_bottom {
    height:237.8px;
}

#left_bottom_bottom {
    height:291.1px;
}
#section {
    height:754.4px;
}

#section_top {
    height:651.9px;
}
```

```

        #section_bottom {
            height:98.4px;
        }

        #section_bottom_left {
            height:65.6px;
        }

        #section_bottom_right {
            height:65.6px;
        }

        #right {
            height:754.4px;
        }

        #right_top {
            height:246px;
        }

        #right_middle {
            height:246px;
        }


        #right_bottom {
            height:250.1px;
        }

        #footer {
```

```
        height:16.4px;
    }

    p#left_option {
        font-size: 1.23em;
    }
    p#section_top_p {
        font-size: 1.23em;
    }
    p#section_bottom_p {
        font-size: 1.23em;
    }
    p#right_option {
        font-size: 1.312em;
    }
    p#header_p_left {
        font-size: 1.64em;
    }
    p#header_p_right {
        font-size: 1.64em;
    }
    p#footer_p{
        font-size: 0.82em;
    }
    select#op_postposition {
       height: 24.6px;
       font-size: 13.94px;
    }
    #container_leftmiddle {
        height: 164px;
        border-radius: 8.2px;
```

```
        }
        #container_leftbottom {
            height: 196.8px;
            border-radius: 8.2px;
        }
        .CB_leftmiddle {
            font-size: 11.48px;
        }

        .CB_leftbottom{
            font-size: 11.48px;]
        }
        #play-button {
          font: 10.66px sans-serif;
        }
        .ticks {
          font: 8.2px sans-serif;
        }
        .track {
          stroke-width: 8.2px;
        }
        .track-inset {
          stroke-width: 6.56px;
        }
        .track-overlay {
          stroke-width: 41px;
        }
        div.tooltip {
            font-size: 13.94px;
        }
        div.epoch {
```

```
            font-size: 32.8px;
        }
        #tooltip_top {
            width: 98.4px;
            font-size: 13.12px;
        }
        #tooltip_middle {
            width: 98.4px;
            font-size: 13.12px;
        }
        #tooltip_bottom {
            width: 131.2px;
        }
        #tooltip_bottom p {
            font-size: 13.12px;
        }
        h5 {
            font-size: 12.3px;
        }

        text.rightbottom_text {
            font-size: 16.4px;
        }
        text.rangetext{
            font-size: 17.22px;
        }
    }

    @media all and (min-width:701px) and (max-height: 818px) {
        /*0.72*/
        #header{
```

```
        height:35.05px;
    }
    #header_left{
        height:35.05px;
    }

    #header_right{
        height:35.05px;
    }
    #left {
        height:662.4px;
    }

    #left_top {
        height:61.2px;
    }

    #left_middle {
        height:180px;
    }

    #left_bottom {
        height:208.8px;
    }

    #left_bottom_bottom {
        height:255.6px;
    }
    #section {
        height:662.4px;
    }
```

```
#section_top {
    height:572.4px;
}

#section_bottom {
    height:86.4px;
}

#section_bottom_left {
    height:57.6px;
}

#section_bottom_right {
    height:57.6px;
}

#right {
    height:662.4px;
}

#right_top {
    height:216px;
}

#right_middle {
    height:216px;
}


#right_bottom {
```

```
        height:219.6px;
    }

    #footer {
        height:14.4px;
    }

    p#left_option {
        font-size: 1.08em;
    }
    p#section_top_p {
        font-size: 1.08em;
    }
    p#section_bottom_p {
        font-size: 1.08em;
    }
    p#right_option {
        font-size: 1.152em;
    }
    p#header_p_left {
        font-size: 1.44em;
    }
    p#header_p_right {
        font-size: 1.44em;
    }
    p#footer_p{
        font-size: 0.72em;
    }
    select#op_postposition {
       height: 21.6px;
       font-size: 12.24px;
```

```
}
#container_leftmiddle {
    height: 144px;
    border-radius: 7.2px;
}
#container_leftbottom {
    height: 172.8px;
    border-radius: 7.2px;
}
.CB_leftmiddle {
    font-size: 10.08px;
}

.CB_leftbottom{
    font-size: 10.08px;]
}
#play-button {
  font: 9.36px sans-serif;
}
.ticks {
  font: 7.2px sans-serif;
}
.track {
  stroke-width: 7.2px;
}
.track-inset {
  stroke-width: 5.76px;
}
.track-overlay {
  stroke-width: 36px;
}
```

```
        div.tooltip {
            font-size: 12.24px;
        }
        div.epoch {
            font-size: 28.8px;
        }
        #tooltip_top {
            width: 86.4px;
            font-size: 11.52px;
        }
        #tooltip_middle {
            width: 86.4px;
            font-size: 11.52px;
        }
        #tooltip_bottom {
            width: 115.2px;
        }
        #tooltip_bottom p {
            font-size: 11.52px;
        }
        h5 {
            font-size: 10.8px;
        }

        text.rightbottom_text {
            font-size: 14.4px;
        }
        text.rangetext{
            font-size: 15.12px;
        }
    }
```

```
@media all and (min-width:450px) and (max-height: 700px) {
    /*0.45*/
    #header{
        height:22.5px;
    }
    #header_left{
        height:22.5px;
    }

    #header_right{
        height:22.5px;
    }
    #left {
        height:414px;
    }

    #left_top {
        height:38.25px;
    }

    #left_middle {
        height:112.5px;
    }

    #left_bottom {
        height:130.5px;
    }

    #left_bottom_bottom {
        height:159.75px;
```

```
        }
        #section {
            height:414px;
        }

        #section_top {
            height:357.75px;
        }

        #section_bottom {
            height:54px;
        }

        #section_bottom_left {
            height:36px;
        }

        #section_bottom_right {
            height:36px;
        }

        #right {
            height:414px;
        }

        #right_top {
            height:135px;
        }

        #right_middle {
            height:135px;
```

```
}


#right_bottom {
    height:137.25px;
}

#footer {
    height:9px;
}

p#left_option {
    font-size: 0.675em;
}
p#section_top_p {
    font-size: 0.675em;
}
p#section_bottom_p {
    font-size: 0.675em;
}
p#right_option {
    font-size: 0.72em;
}
p#header_p_left {
    font-size: 0.9em;
}
p#header_p_right {
    font-size: 0.9em;
}
p#footer_p{
    font-size: 0.3em;
```

```
            }
            select#op_postposition {
               height: 13.5px;
               font-size: 7.65px;
            }
            #container_leftmiddle {
                height: 90px;
                border-radius: 4.5px;
            }
            #container_leftbottom {
                height: 108px;
                border-radius: 4.5px;
            }
            .CB_leftmiddle {
                font-size: 6.3px;
            }

            .CB_leftbottom{
                font-size: 6.3px;]
            }
            #play-button {
              font: 5.85px sans-serif;
            }
            .ticks {
              font: 4.5px sans-serif;
            }
            .track {
              stroke-width: 4.5px;
            }
            .track-inset {
              stroke-width: 3.6px;
```

```
}
.track-overlay {
  stroke-width: 22.5px;
}
div.tooltip {
    font-size: 7.65px;
}
div.epoch {
    font-size: 18px;
}
#tooltip_top {
    width: 54px;
    font-size: 7.2px;
}
#tooltip_middle {
    width: 54px;
    font-size: 7.2px;
}
#tooltip_bottom {
    width: 72px;
}
#tooltip_bottom p {
    font-size: 7.2px;
}
h5 {
    font-size: 6.75px;
}

text.rightbottom_text {
    font-size: 9px;
}
```

```
        text.rangetext{
            font-size: 9.45px;
        }
    }
</style>
</head>
<body>
    <div id="header">
        <div id="header_left">
            <p id="header_p_left" align="left">PostBERT</p
                >
        </div>
        <div id="header_right" align="right">
            <p id="header_p_right"><a id="header_a" href="
                https://github.com/seongmin-mun/
                VisualSystem/tree/master/Major/PostBERT">
                GitHub</a></p>
        </div>
    </div>
    <div id="left">
        <div id="left_top">
            <p id="left_option">Postposition</p>
            <select id="op_postposition">
                <option value="ey" selected="selected">
                    -ey</option>
                <option value="eyse">-eyse</option>
                <option value="(u)lo">-(u)lo</option>
            </select>
        </div>
        <div id="left_middle">
            <p id="left_option">Select function</p>
```

```
                <div id="container_leftmiddle">
                </div>
            </div>
            <div id="left_bottom">
                <p id="left_option">Select sentence</p>
                <div id="container_leftbottom">
                </div>
            </div>
            <div id="left_bottom_bottom">
                <p id="left_option">Density cluster</p>

            </div>
        </div>
        <div id="section">
            <div id="section_top">
                <p id="section_top_p">t-SNE visualization of
                    BERT sentence classification</p>
            </div>
            <div id="section_bottom">
                <p id="section_bottom_p">Current epoch</p>
                <div id="section_bottom_left">
                    <button id="play-button">Play</button>
                </div>
                <div id="section_bottom_right">
                </div>
            </div>
        </div>
        <div id="right">
            <div id="right_top">
                <p id="right_option">Overall accuracy & Loss
                    </p>
```

```
                    <div id="tooltip_top" style='display:none'>
                    </div>
                </div>
                <div id="right_middle">
                    <p id="right_option">Individual accuracy</p>
                    <div id="tooltip_middle" style='display:none'>
                    </div>
                </div>
                <div id="right_bottom">
                    <p id="right_option">Bar chart for density
                        cluster</p>
                    <div id="tooltip_bottom" class="hidden">
                    </div>
                    </div>
                </div>
            </div>
            <div id="footer">
                <p id="footer_p">2020 - 2021, <a id="footer_a"
                   href="https://seongmin-mun.github.io/MyWebsite/
                   Seongmin/index.html">Seongmin Mun</a>. All
                   rights reserved.<p>
            </div>
<script>
$(document).ready(function () {

var function_name = ['agt','crt','eff','fns','gol','ins','loc'
    ,'thm','src','dir'];
var function_color = ['#993366','#FF99CC','#FF00FF','#FF6600',
    '#000080','#003300',' #0066CC','#666699','#FFCC00','#5b2e90
    '];

```

```
var LeftsectionWidth = $("#left_bottom_bottom").width()
var LeftsectionHeight = $("#left_bottom_bottom").height()

var LeftsvgSection = d3.select('#left_bottom_bottom').append('
    svg')
.attr('width', LeftsectionWidth)
.attr('height', (LeftsectionHeight*0.9))

var imgs = LeftsvgSection.append("image")
.attr("class", "PNG")
.attr("xlink:href", "https://seongmin-mun.github.io/
    VisualSystem/Major/PostBERT.ko/images/densityClusterPNG_r/
    Ey_tSNE_epoch_0.png")
.attr("x", LeftsectionWidth*0.05)
.attr("y", 0)
.attr('width', LeftsectionWidth*0.9)
.attr('height', LeftsectionWidth*0.9);

var sectionWidth = $("#section_top").width()
var sectionHeight = $("#section_top").height()

var svgSection = d3.select('#section_top').append('svg')
.attr('width', sectionWidth)
.attr('height', (sectionHeight*0.935))
.call(d3.zoom().scaleExtent([0.5, 5]).on("zoom", function () {
svgSection.attr("transform", d3.event.transform)
}))
.append("g");

var div_epoch = d3.select("#section").append("div")
.attr("class", "epoch")
```

```
.style("opacity", 0.8)
.style("right", sectionWidth*0.03+"px")
.style("top", sectionHeight*0.01+"px");

var textlabel = div_epoch.append("text")
.attr("class", "textlabel")
.attr("text-anchor", "middle")
.text("1 Epoch")
.attr("text-anchor", "end")
.attr("font-family", "Open Sans")
.attr("font-size", "25px")
.attr("fill", "#C2C1C1")

var NodeGroup = svgSection.append("g");

var div_inner = d3.select("#section").append("div")
.attr("class", "tooltip")
.style("opacity", 0);

var right_top_width = $("#right_top").width()
var right_top_height = $("#right_top").height()

var svgright_top = d3.select("#right_top")
.append("svg")
.attr("width", right_top_width*0.95)
.attr("height", (right_top_height*0.95))
.append('g')
.attr('transform', 'translate(' + 0 + ',' + 0 + ')');

svgright_top.append("rect")
.attr("class", "svgright_rect")
```

```
.attr("x", right_top_width * 0.05)
.attr("y", 0)
.attr("width", right_top_width*0.9)
.attr("height",(right_top_height*0.75))
.attr("rx", 6)
.attr("ry", 6)
.attr("fill", "white")
.attr('stroke', '#C2C1C1')
.attr('stroke-width', '2')

var epoch_right_top = ["0", "10", "20", "30", "40", "50"]

for (var k = 0; k < 6; k++) {
svgright_top.append("text").text(epoch_right_top[k]).attr("x", (((right_top_width * 0.82) *0.205) * k) + (right_top_width * 0.085)).attr("y", right_top_height*0.69).attr("text-anchor", "middle").attr("font-family", "Open Sans").attr("font-size", "21px").attr("fill", "#C2C1C1")
}

svgright_top.append("line").attr("x1", right_top_width * 0.05).attr("y1", right_top_height*0.6).attr("x2", right_top_width * 0.95).attr("y2", right_top_height*0.6).attr("stroke-width", "2px").attr("stroke", "#C2C1C1").style("stroke-dasharray", ("3, 3"))

for (var k = 0; k < 6; k++) {
svgright_top.append("line").attr("x1", ((right_top_width * 0.162) * k) + (right_top_width * 0.065)).attr("y1", right_top_height*0.01).attr("x2", ((right_top_width * 0.162) * k) + (right_top_width * 0.065)).attr("y2",
```

```
    right_top_height*0.75).attr("stroke-width", "2px").attr("
    stroke", "#C2C1C1").style("stroke-dasharray", ("3, 3"))
}

var right_middle_width = $("#right_middle").width()
var right_middle_height = $("#right_middle").height()

var svgright_middle = d3.select("#right_middle")
.append("svg")
.attr("width", right_middle_width*0.95)
.attr("height", (right_middle_height*0.95))

svgright_middle.append("rect")
.attr("class", "svgright_rect")
.attr("x", right_middle_width * 0.05)
.attr("y", 0)
.attr("width", right_middle_width*0.9)
.attr("height",(right_middle_height*0.75))
.attr("rx", 6)
.attr("ry", 6)
.attr("fill", "white")
.attr('stroke', '#C2C1C1')
.attr('stroke-width', '2')

var  epoch_right_middle = ["0", "10", "20", "30", "40", "50"]

for (var k = 0; k < 6; k++) {
svgright_middle.append("text").text(epoch_right_middle[k]).
    attr("x", (((right_middle_width * 0.82) *0.205) * k) + (
    right_middle_width * 0.085)).attr("y", right_middle_height
    *0.69).attr("text-anchor", "middle").attr("font-family", "
```

```
Open Sans").attr("font-size", "21px").attr("fill", "#C2C1C1")
}

svgright_middle.append("line").attr("x1",  right_middle_width * 0.05).attr("y1", right_middle_height*0.6).attr("x2", right_middle_width * 0.95).attr("y2", right_middle_height*0.6).attr("stroke-width", "2px").attr("stroke", "#C2C1C1").style("stroke-dasharray", ("3, 3"))

for (var k = 0; k < 6; k++) {
svgright_middle.append("line").attr("x1", ((right_middle_width * 0.162) * k) + (right_middle_width * 0.065)).attr("y1", right_middle_height*0.01).attr("x2", ((right_middle_width * 0.162) * k) + (right_middle_width * 0.065)).attr("y2", right_middle_height*0.75).attr("stroke-width", "2px").attr("stroke", "#C2C1C1").style("stroke-dasharray", ("3, 3"))
}

var right_bottom_width = $("#right_bottom").width()
var right_bottom_height = $("#right_bottom").height()

var svgright_bottom = d3.select("#right_bottom")
.append("svg")
.attr("width", right_bottom_width*0.95)
.attr("height", (right_bottom_height*0.95))

svgright_bottom.append("rect")
.attr("class", "svgright_rect")
.attr("x", right_bottom_width * 0.05)
.attr("y", 0)
```

```
.attr("width", right_bottom_width*0.9)
.attr("height",(right_bottom_height*0.75))
.attr("rx", 6)
.attr("ry", 6)
.attr("fill", "white")
.attr('stroke', '#C2C1C1')
.attr('stroke-width', '2')

var  epoch_right_bottom = ["0", "10", "20", "30", "40", "50"]

for (var k = 0; k < 6; k++) {
svgright_bottom.append("text").text(epoch_right_bottom[k]).attr("x", (((right_bottom_width * 0.82) *0.205) * k) + (right_bottom_width * 0.085)).attr("y", right_bottom_height*0.69).attr("text-anchor", "middle").attr("font-family", "Open Sans").attr("font-size", "21px").attr("fill", "#C2C1C1")
}

svgright_bottom.append("line").attr("x1",  right_bottom_width * 0.05).attr("y1", right_bottom_height*0.6).attr("x2", right_bottom_width * 0.95).attr("y2", right_bottom_height*0.6).attr("stroke-width", "2px").attr("stroke", "#C2C1C1").style("stroke-dasharray", ("3, 3"))

for (var k = 0; k < 6; k++) {
svgright_bottom.append("line").attr("x1", ((right_bottom_width * 0.1651) * k) + (right_bottom_width * 0.065)).attr("y1", right_bottom_height*0.01).attr("x2", ((right_bottom_width * 0.1651) * k) + (right_bottom_width * 0.065)).attr("y2", right_bottom_height*0.75).attr("stroke-width", "2px").attr(
```

```
    "stroke", "#C2C1C1").style("stroke-dasharray", ("3, 3"))
}

var SB_width = $("#section_bottom_right").width()
var SB_height = $("#section_bottom_right").height()

var SB_svg = d3.select("#section_bottom_right")
.append("svg")
.attr("width", SB_width)
.attr("height", SB_height);


var target = actual = 0;
var alpha = 0.2;
var timer = d3.timer(updateTween);
var stepTimer;
var moving = false;
var maxValue = 49;
var trailLength = 10;
var currentEpoch = 0;

var playButton = d3.select("#play-button");

var x = d3.scaleLinear()
.domain([1, 49])
.range([0, SB_width*0.9])
.clamp(true);

var slider = SB_svg.append("g")
.attr("class", "slider")
```

```
.attr("transform", "translate(" + SB_width*0.05 + "," +
    SB_height/2 + ")");

slider.append("line")
.attr("class", "track")
.attr("x1", x.range()[0])
.attr("x2", x.range()[1])
.select(function() { return this.parentNode.appendChild(
    this.cloneNode(true)); })
.attr("class", "track-inset")
.select(function() { return this.parentNode.appendChild(
    this.cloneNode(true)); })
.attr("class", "track-overlay")
.call(d3.drag()
.on("start.interrupt", function() { slider.interrupt(); })
.on("start drag", function() {
currentValue = d3.event.x;
update(x.invert(currentValue));
})
);

slider.insert("g", ".track-overlay")
.attr("class", "ticks")
.attr("transform", "translate(0," + 18 + ")")
.selectAll("text")
.data(x.ticks(10))
.enter().append("text")
.attr("x", x)
.attr("text-anchor", "middle")
.text(d => d+" epoch" );

```

```
const handle = slider.insert("circle", ".track-overlay")
.attr("class", "handle")
.attr("r", 9);

d3.select(window)
.on("keydown", keydowned);

playButton
.on("click", paused)
.each(paused);

function update(d) {
target = d;
moving = true;
timer.restart(updateTween);
drawall();
}

function updateTween() {
let diff = target - actual;
if (Math.abs(diff) < 1e-3) actual = target, timer.stop();
else actual += diff * alpha;
handle.attr("cx", x(actual));
currentEpoch = Math.round(actual)

return currentEpoch

}

function keydowned() {
let currentValue = actual;
```

```
if (d3.event.metaKey || d3.event.altKey) return;
switch (d3.event.keyCode) {
case 37: currentValue = Math.max(x.domain()[0], actual -
    trailLength); break;
case 39: currentValue = Math.min(x.domain()[1], actual +
    trailLength); break;
default: return;
}
update(currentValue);
paused();
}

function paused() {
if (moving) {
slider.interrupt();
clearInterval(stepTimer);
moving = false;
playButton.text("Play");
} else {
if (actual > maxValue) actual = 0;
stepTimer = setInterval(step, 1500);
moving = true;
playButton.text("Pause");
}
}

function step() {
if (actual > maxValue) paused();
else update(actual + trailLength / 10);
}

```

```
paused();
var functionslist_ey = ['LOC','GOL','EFF','CRT','THM','INS','
    AGT','FNS'];
var functionsname_ey = ['Location','Goal','Effector','
    Criterion','Theme','Instrument','Agent','Final State'];

var functionslist_eyse = ['SRC','LOC'];
var functionsname_eyse = ['Source','Location'];

var functionslist_lo = ['LOC','DIR','EFF','CRT','INS','FNS'];
var functionsname_lo = ['Location','Direction','Effector','
    Criterion','Instrument','Final State'];

function draw_op_function_after(post,list,name,name_kr){
$('#container_leftmiddle').empty();
if (post === "ey"){
for(var i = 0 ; i < list.length; i++){
var color = ""
var currentFunc = list[i].toLowerCase()
if (currentFunc == function_name[0]) {
color = function_color[0]
} else if (currentFunc == function_name[1]) {
color = function_color[1]
} else if (currentFunc == function_name[2]) {
color = function_color[2]
} else if (currentFunc == function_name[3]) {
color = function_color[3]
} else if (currentFunc == function_name[4]) {
color = function_color[4]
} else if (currentFunc == function_name[5]) {
color = function_color[5]
```

```
} else if (currentFunc == function_name[6]) {
color = function_color[6]
} else if (currentFunc == function_name[7]) {
color = function_color[7]
} else if (currentFunc == function_name[8]) {
color = function_color[8]
} else if (currentFunc == function_name[9]) {
color = function_color[9]
}
$("#container_leftmiddle").append("<input class='CB_leftmiddle
    ' type='checkbox' value='"+list[i].toLowerCase()+"' id='
    CB_leftmiddle_"+list[i].toLowerCase()+"' /> <label class='
    CB_leftmiddle'><svg width='12' height='12' ><rect width=
    '11' height='11' rx='2' class='legendrect' style='fill:"+
    color+";opacity:0.9;'/></svg> "+list[i]+" ("+name[i]+","+
    name_kr[i]+")</label></br>");
}
} else if (post === "eyse"){
for(var i = 0 ; i < list.length; i++){
var color = ""
var currentFunc = list[i].toLowerCase()
if (currentFunc == function_name[0]) {
color = function_color[0]
} else if (currentFunc == function_name[1]) {
color = function_color[1]
} else if (currentFunc == function_name[2]) {
color = function_color[2]
} else if (currentFunc == function_name[3]) {
color = function_color[3]
} else if (currentFunc == function_name[4]) {
color = function_color[4]
```

```
} else if (currentFunc == function_name[5]) {
color = function_color[5]
} else if (currentFunc == function_name[6]) {
color = function_color[6]
} else if (currentFunc == function_name[7]) {
color = function_color[7]
} else if (currentFunc == function_name[8]) {
color = function_color[8]
} else if (currentFunc == function_name[9]) {
color = function_color[9]
}
$("#container_leftmiddle").append("<input class='CB_leftmiddle
    ' type='checkbox' value='"+list[i].toLowerCase()+"' id='
    CB_leftmiddle_"+list[i].toLowerCase()+"' /> <label class='
    CB_leftmiddle'><svg width='12' height='12'><rect width='11'
     height='11' rx='2' class='legendrect' style='fill:"+color+
    ";opacity:0.9;'/></svg> "+list[i]+" ("+name[i]+","+name_kr[
    i]+")</label></br>");
}
} else if (post === "(u)lo"){
for(var i = 0 ; i < list.length; i++){
var color = ""
var currentFunc = list[i].toLowerCase()
if (currentFunc == function_name[0]) {
color = function_color[0]
} else if (currentFunc == function_name[1]) {
color = function_color[1]
} else if (currentFunc == function_name[2]) {
color = function_color[2]
} else if (currentFunc == function_name[3]) {
color = function_color[3]
```

```
} else if (currentFunc == function_name[4]) {
color = function_color[4]
} else if (currentFunc == function_name[5]) {
color = function_color[5]
} else if (currentFunc == function_name[6]) {
color = function_color[6]
} else if (currentFunc == function_name[7]) {
color = function_color[7]
} else if (currentFunc == function_name[8]) {
color = function_color[8]
} else if (currentFunc == function_name[9]) {
color = function_color[9]
}
$("#container_leftmiddle").append("<input class='CB_leftmiddle
    ' type='checkbox' value='"+list[i].toLowerCase()+"' id='
    CB_leftmiddle_"+list[i].toLowerCase()+"' /> <label class='
    CB_leftmiddle'><svg width='12' height='12'><rect width='11'
     height='11' rx='2' class='legendrect' style='fill:"+color+
    ";opacity:0.9;'/></svg> "+list[i]+" ("+name[i]+","+name_kr[
    i]+")</label></br>");
}
}
}

function op_function_change(){
var selected_postposition = $( "#op_postposition" ).val();
if (selected_postposition === "ey"){
draw_op_function_after("ey",functionslist_ey,functionsname_ey,
    functionsname_kr_ey)
draw_op_index_after("ey")

```

```
} else if (selected_postposition === "eyse"){
draw_op_function_after("eyse",functionslist_eyse,functionsname_eyse,functionsname_kr_eyse)
draw_op_index_after("eyse")
} else if (selected_postposition === "(u)lo"){
draw_op_function_after("(u)lo",functionslist_lo,functionsname_lo,functionsname_kr_lo)
draw_op_index_after("(u)lo")
}

drawall();
}

function draw_op_index_after(post){
$('#container_leftbottom').empty();
if (post === "ey"){
for(var i = 0 ; i < 467; i++){
$("#container_leftbottom").append("<input class='CB_leftbottom' type='checkbox' value='index"+i+"' id='CB_leftbottom_"+i+"' /> <label class='CB_leftbottom'>index_"+i+"</label></br>");
}
} else if (post === "eyse"){
for(var i = 0 ; i < 484; i++){
$("#container_leftbottom").append("<input class='CB_leftbottom' type='checkbox' value='index"+i+"' id='CB_leftbottom_"+i+"' /> <label class='CB_leftbottom'>index_"+i+"</label></br>");
}
} else if (post === "(u)lo"){
for(var i = 0 ; i < 467; i++){
```

```
$("#container_leftbottom").append("<input class='CB_leftbottom' type='checkbox' value='index"+i+"' id='CB_leftbottom_"+i+"' /> <label class='CB_leftbottom'>index_"+i+"</label></br>");
}
}
}

function functioncheckbox() {

var data_checkbox = []

var selected_postposition = $( "#op_postposition" ).val();
if (selected_postposition === "ey"){
for(var i = 0; i < functionslist_ey.length; i++){
if(checkedeachValue('CB_leftmiddle_'+functionslist_ey[i].toLowerCase())!=='null'){
data_checkbox.push(checkedeachValue('CB_leftmiddle_'+functionslist_ey[i].toLowerCase()));
}
}

} else if (selected_postposition === "eyse"){
for(var i = 0; i < functionslist_eyse.length; i++){
if(checkedeachValue('CB_leftmiddle_'+functionslist_eyse[i].toLowerCase())!=='null'){
data_checkbox.push(checkedeachValue('CB_leftmiddle_'+functionslist_eyse[i].toLowerCase()));
}
}
} else if (selected_postposition === "(u)lo"){
```

```
for(var i = 0; i < functionslist_lo.length; i++){
if(checkedeachValue('CB_leftmiddle_'+functionslist_lo[i].
    toLowerCase())!=='null'){
data_checkbox.push(checkedeachValue('CB_leftmiddle_'+
    functionslist_lo[i].toLowerCase()));
}
}
}

return data_checkbox;
}

function indexcheckbox() {

var data_checkbox = []

var selected_postposition = $( "#op_postposition" ).val();
if (selected_postposition === "ey"){
for(var i = 0 ; i < 467; i++){
if(checkedeachValue('CB_leftbottom_'+i)!=='null'){
data_checkbox.push(checkedeachValue('CB_leftbottom_'+i));
}
}

} else if (selected_postposition === "eyse"){
for(var i = 0 ; i < 484; i++){
if(checkedeachValue('CB_leftbottom_'+i)!=='null'){
data_checkbox.push(checkedeachValue('CB_leftbottom_'+i));
}
}
} else if (selected_postposition === "(u)lo"){
```

```
for(var i = 0 ; i < 467; i++){
if(checkedeachValue('CB_leftbottom_'+i)!=='null'){
data_checkbox.push(checkedeachValue('CB_leftbottom_'+i));
}
}
}

return data_checkbox;
}

function checkedeachValue(checkeddata){
var value;
var checkedValue = document.querySelector('#'+checkeddata+':
    checked');
if(checkedValue == null){
value = "null";
} else {
value = checkedValue.value;
}
return value;
}

op_function_change()

function right_top_draw(){
var selected_postposition = $( "#op_postposition" ).val();

EpochNow = updateTween()

EpochNoww = EpochNow

```

```
svgright_top.selectAll(".righttoppath").remove();

var dataTotal = []

for (var i = 0; i < Accuracy_info.length ; i++) {
if ((Accuracy_info[i].postposition === selected_postposition)) {
for(var j = 0; j < Accuracy_info[i].accuracy.length ; j++){
if(j<=EpochNoww){
dataTotal.push(Accuracy_info[i].accuracy[j])
}
}
}
}

funcList = []

for (var key in dataTotal[0]){
if((key === 'total')||(key === 'loss')){
funcList.push(key)
} else {
continue;
}
}

var final_data = []

for (var i = 0; i < funcList.length ; i++) {
currentFunc = funcList[i]
var dataT = {}
var dataY = [];
```

```
var currentaccuracy = 0
for (var j = 0; j < dataTotal.length ; j++) {
dataY.push(dataTotal[j][funcList[i]])
currentaccuracy = dataTotal[j][funcList[i]]
}
dataT['y'] = dataY
var dataX = [];
for (var j = 0; j < 50 ; j++) {
dataX.push(j)
}
dataT['x'] = dataX

var rearrangedData = dataT.x.map(function(d,i) {
return {x:d,y:dataT.y[i]};
})

var re_data = {}
re_data['name'] = currentFunc
re_data['show'] = true
re_data['currentLoss'] = currentaccuracy
re_data['history'] = rearrangedData

if (currentFunc == 'total') {
re_data['color'] = "#49441F"
} else if (currentFunc == 'loss') {
re_data['color'] = "#003366"
}

final_data.push(re_data)
}

```

```
var right_top_x = d3.scale.linear().domain([0,50]).range([right_top_width*0.07,right_top_width*0.9])
var right_top_y = d3.scale.linear().domain([-0.25,1.5]).range([right_top_height*0.7,0])

var line = d3.svg.line()
.x(function(d){ return right_top_x(d.x)})
.y(function(d){return right_top_y(d.y)})
.interpolate("linear");

const tooltip_top = d3.select('#tooltip_top');
const tooltipLine_top = svgright_top.append('line');

svgright_top.selectAll()
.data(final_data).enter()
.append('path')
.attr('class','righttoppath')
.attr('fill', 'none')
.attr('stroke', d => d.color)
.attr('stroke-width', 2)
.datum(d => d.history)
.attr('d', line)
.attr("opacity",0.7);

svgright_top.selectAll()
.data(final_data).enter()
.append('text')
.html(d => d.name)
.attr('fill', d => d.color)
.attr('alignment-baseline', 'middle')
.attr('x', right_top_width)
```

```
.attr('dx', '.5em')
.attr('y', d => right_top_y(d.currentLoss));
tipBox = svgright_top.append('rect')
.attr('width', right_top_width)
.attr('height', right_top_height)
.attr('opacity', 0)
.on('mousemove', drawTooltip)
.on('mouseout', removeTooltip);
function removeTooltip() {
if (tooltip_top) tooltip_top.style('display', 'none');
if (tooltipLine_top) tooltipLine_top.attr('stroke', 'none');
}
function drawTooltip() {
const x = Math.floor((right_top_x.invert(d3.mouse(tipBox.node())[0])+0.5));
final_data.sort((a, b) => {
return b.history.find(h => h.x == x).y - a.history.find(h => h.x == x).y;
})
tooltipLine_top.attr('stroke', 'black')
.attr('x1', right_top_x(x)-1)
.attr('x2', right_top_x(x)-1)
.attr('y1', 0)
.attr('y2', right_top_height)
.attr('opacity',0.7);
```

```
tooltip_top.html(x+1)
.style('display', 'block')
.style('right', right_top_width*0.07+"px")
.style('top', right_top_height*0.28+"px")
.style('opacity',0.7)
.selectAll()
.data(final_data).enter()
.append('div')
.style('color', d => d.color)
.html(d => d.name + ': ' + d.history.find(h => h.x == x).y);
}
}

function right_middle_draw(){
var selected_postposition = $( "#op_postposition" ).val();

EpochNow = updateTween()

EpochNoww = EpochNow

svgright_middle.selectAll(".rightmiddlepath").remove();

var dataTotal = []

for (var i = 0; i < Accuracy_info.length ; i++) {
if ((Accuracy_info[i].postposition === selected_postposition))
    {
for(var j = 0; j < Accuracy_info[i].accuracy.length ; j++){
if(j<=EpochNoww){
dataTotal.push(Accuracy_info[i].accuracy[j])
```

```
}
}
}
}

funcList = []

for (var key in dataTotal[0]){
if((key === 'epoch')||(key === 'total')||(key === 'loss')||(
    key === 'correlation')){
continue;
} else {
funcList.push(key)
}

}

var final_data = []

for (var i = 0; i < funcList.length ; i++) {
currentFunc = funcList[i]
var dataT = {}
var dataY = [];
var currentaccuracy = 0
for (var j = 0; j < dataTotal.length ; j++) {
dataY.push(dataTotal[j][funcList[i]])
currentaccuracy = dataTotal[j][funcList[i]]
}
dataT['y'] = dataY
var dataX = [];
for (var j = 0; j < 50 ; j++) {
```

```
dataX.push(j)
}
dataT['x'] = dataX

var rearrangedData = dataT.x.map(function(d,i) {
return {x:d,y:dataT.y[i]};
})

var re_data = {}
re_data['name'] = currentFunc
re_data['show'] = true
re_data['currentLoss'] = currentaccuracy
re_data['history'] = rearrangedData

if (currentFunc == function_name[0]) {
re_data['color'] = function_color[0]
} else if (currentFunc == function_name[1]) {
re_data['color'] = function_color[1]
} else if (currentFunc == function_name[2]) {
re_data['color'] = function_color[2]
} else if (currentFunc == function_name[3]) {
re_data['color'] = function_color[3]
} else if (currentFunc == function_name[4]) {
re_data['color'] = function_color[4]
} else if (currentFunc == function_name[5]) {
re_data['color'] = function_color[5]
} else if (currentFunc == function_name[6]) {
re_data['color'] = function_color[6]
} else if (currentFunc == function_name[7]) {
re_data['color'] = function_color[7]
} else if (currentFunc == function_name[8]) {
```

```
re_data['color'] = function_color[8]
} else if (currentFunc == function_name[9]) {
re_data['color'] = function_color[9]
}

final_data.push(re_data)

}

var right_middle_x = d3.scale.linear().domain([0,50]).range([
    right_middle_width*0.07,right_middle_width*0.9])
var right_middle_y = d3.scale.linear().domain([-0.25,1.1]).
    range([right_middle_height*0.72,0])

var line = d3.svg.line()
.x(function(d){ return right_middle_x(d.x)})
.y(function(d){return right_middle_y(d.y)})
.interpolate("linear");

const tooltip_middle = d3.select('#tooltip_middle');
const tooltipLine_middle = svgright_middle.append('line');

svgright_middle.selectAll()
.data(final_data).enter()
.append('path')
.attr('class','rightmiddlepath')
.attr('fill', 'none')
.attr('stroke', d => d.color)
.attr('stroke-width', 2)
.datum(d => d.history)
.attr('d', line)
```

```
.attr("opacity",0.7);

svgright_middle.selectAll()
.data(final_data).enter()
.append('text')
.html(d => d.name)
.attr('fill', d => d.color)
.attr('alignment-baseline', 'middle')
.attr('x', right_middle_width)
.attr('dx', '.5em')
.attr('y', d => right_middle_y(d.currentLoss));

tipBox = svgright_middle.append('rect')
.attr('width', right_middle_width)
.attr('height', right_middle_height)
.attr('opacity', 0)
.on('mousemove', drawTooltip)
.on('mouseout', removeTooltip);

function removeTooltip() {
if (tooltip_middle) tooltip_middle.style('display', 'none');
if (tooltipLine_middle) tooltipLine_middle.attr('stroke', 'none');
}

function drawTooltip() {
const x = Math.floor((right_middle_x.invert(d3.mouse(tipBox.node())[0])+0.5));

final_data.sort((a, b) => {
```

```
return b.history.find(h => h.x == x).y - a.history.find(h =>
    h.x == x).y;
})

tooltipLine_middle.attr('stroke', 'black')
.attr('x1', right_middle_x(x)-1)
.attr('x2', right_middle_x(x)-1)
.attr('y1', 0)
.attr('y2', right_middle_height)
.attr('opacity',0.7);

tooltip_middle.html(x+1)
.style('display', 'block')
.style('right', right_top_width*0.07+"px")
.style('top', right_middle_height*0.28+"px")
.style('opacity',0.7)
.selectAll()
.data(final_data).enter()
.append('div')
.style('color', d => d.color)
.html(d => d.name + ': ' + d.history.find(h => h.x == x).y);
}
}

function right_bottom_draw(){
var selected_postposition = $( "#op_postposition" ).val();

EpochNow = updateTween()

EpochNoww = EpochNow

```

```
svgright_bottom.selectAll(".rightbottom_bar").remove();
svgright_bottom.selectAll(".rightbottom_text").remove();

var dataTotal = [0]

for (var i = 0; i < Density_info.length ; i++) {
if ((Density_info[i].postposition === selected_postposition))
    {
for(var j = 0; j < Density_info[i].cluster.length ; j++){
if(j<=EpochNoww){
dataTotal.push(Density_info[i].cluster[j].clusterNumber)
}
}
}
}

yScaleMax = 0;
if(selected_postposition==="ey"){
yScaleMax = 7;
} else if (selected_postposition==="eyse"){
yScaleMax = 3;
} else {
yScaleMax = 7;
}

var right_bottom_x = d3.scale.linear()
.domain([0,50])
.range([right_bottom_width*0.05,right_bottom_width*0.88], 0.1)
    ;

var right_bottom_y = d3.scale.linear()
```

```
.domain([0, yScaleMax])
.range([0,right_bottom_height*0.62]);

svgright_bottom.selectAll("rect")
.data(dataTotal)
.enter()
.append("rect")
.attr("class","rightbottom_bar")
.attr("x", function(d, i) {
return right_bottom_x(i);
})
.attr("y", function(d) {
return (right_bottom_height*0.6) - right_bottom_y(d);
})
.attr("width", right_bottom_width*0.015)
.attr("height", function(d) {
return right_bottom_y(d);
})
.attr("fill", function(d) {
return "#666666";
})
.on("mouseover", function(d, i) {

d3.select("#tooltip_bottom")
.style("right", right_bottom_width*0.07 + "px")
.style("top",  right_bottom_height*0.28 + "px")
.html("<p><strong>Selected epoch: </strong>"+i+"<br><strong>
    Cluster number: </strong>"+d+"</p>")

d3.select("#tooltip_bottom").classed("hidden", false);

```

```
})
.on("mouseout", function() {

d3.select("#tooltip_bottom").classed("hidden", true);

});

svgright_bottom.append("text")
.attr("class","rightbottom_text")
.text("Current cluster number: "+dataTotal[EpochNow+1])
.attr("x", right_bottom_width*0.08)
.attr("y", right_bottom_height*0.1)
.attr("text-anchor", "start")
.attr("font-family", "Open Sans")
.attr("font-size", "20px")
.attr("fill", "#666666")
}

firstdrawdata();

d3.selectAll("#op_postposition").on("change",
    op_function_change);
d3.selectAll("#container_leftmiddle").on("change", drawall);
d3.selectAll("#container_leftbottom").on("change", drawall);

function drawall(){

var selected_postposition = $( "#op_postposition" ).val();

var functions = functioncheckbox();
var indexs = indexcheckbox();
```

```
changedrawdata(selected_postposition,functions,indexs)

}

function firstdrawdata() {
var data = {};
for (var i = 0; i < Map_info.length ; i++) {
if ((Map_info[i].postposition === 'ey') && (Map_info[i].epoch
    === 'epoch0')) {
var innerArray = [];
for(var j = 0; j < Sentence_info.length ; j++){
if (Sentence_info[j].postposition === 'ey') {
for(var k = 0; k < Map_info[i].sentences.length ; k++){
var sentenceDic = {}
sentenceDic["index"] = Sentence_info[j].sentences[k].index
sentenceDic["function"] = Sentence_info[j].sentences[k].
    function
sentenceDic["sentence"] = Sentence_info[j].sentences[k].
    sentence
sentenceDic["sentence_pos"] = Sentence_info[j].sentences[k].
    sentence_pos
sentenceDic["X"] = Map_info[i].sentences[k].X
sentenceDic["Y"] = Map_info[i].sentences[k].Y
sentenceDic["opacity_value"] = 0.7
innerArray.push(sentenceDic)
}
}
}
data["postposition"] = Map_info[i].postposition
data["epoch"] = Map_info[i].epoch
data["sentences"] = innerArray
```

```
}
}

var w = sectionWidth;
var h = sectionHeight;
var padding = (sectionHeight*0.12);

var xScale = d3.scale.linear()
.domain([d3.min(data.sentences, function(d) { return d.Y; }),
    d3.max(data.sentences, function(d) { return d.X; })])
.range([0+padding, w-padding]);

var yScale = d3.scale.linear()
.domain([d3.min(data.sentences, function(d) { return d.Y; }),
    d3.max(data.sentences, function(d) { return d.Y; })])
.range([h-padding, 0+padding]);

NodeGroup.selectAll(".nodedot")
.data(data.sentences)
.enter()
.append("circle")
.attr("class", "nodedot")
.attr("id", function (d) {
return d.index
})
.attr("cx", function (d) {
return xScale(d.X)
})
.attr("cy", function (d) {
return yScale(d.Y)
})
```

```
.attr("r", 3)
.attr("fill", function (d) {
if (d.function == function_name[0]) {
return function_color[0]
} else if (d.function == function_name[1]) {
return function_color[1]
} else if (d.function == function_name[2]) {
return function_color[2]
} else if (d.function == function_name[3]) {
return function_color[3]
} else if (d.function == function_name[4]) {
return function_color[4]
} else if (d.function == function_name[5]) {
return function_color[5]
} else if (d.function == function_name[6]) {
return function_color[6]
} else if (d.function == function_name[7]) {
return function_color[7]
} else if (d.function == function_name[8]) {
return function_color[8]
} else if (d.function == function_name[9]) {
return function_color[9]
}
})
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", function (d) {
return d.opacity_value
})
.style("cursor", "help")
.on("mouseover", function (d) {
```

```
d3.select(this)
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", 1)
.attr("fill","#FF0000")
})
.on("mouseout", function (d) {
d3.select(this)
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", function (d) {
return d.opacity_value
})
.attr("fill", function (d) {
if (d.function == function_name[0]) {
    return function_color[0]
} else if (d.function == function_name[1]) {
    return function_color[1]
} else if (d.function == function_name[2]) {
    return function_color[2]
} else if (d.function == function_name[3]) {
    return function_color[3]
} else if (d.function == function_name[4]) {
    return function_color[4]
} else if (d.function == function_name[5]) {
    return function_color[5]
} else if (d.function == function_name[6]) {
    return function_color[6]
} else if (d.function == function_name[7]) {
    return function_color[7]
} else if (d.function == function_name[8]) {
```

```
    return function_color[8]
} else if (d.function == function_name[9]) {
    return function_color[9]
}
});
})
.on("mouseenter", function (d) {
div_inner.transition()
.duration(200)
.style("opacity", 0.85);
div_inner.html("<strong>Selected sentence</strong><br/><h5>
    Index : "+d.index + "<h5/><h5>Function : "  + d.function +
    "<h5/><h5>Sentence : "  + d.sentence+ "<h5/><h5>SentencePOS
     : "  + d.sentence_pos+ "<h5/>")
.style("left", "20px")
.style("top", sectionHeight*0.07+"px");
})
.on("mouseleave", function () {
div_inner.transition()
.duration(500)
.style("opacity", 0);
});
}

function changedrawdata(selected_postposition,functionarray,
    indexarray) {
if ((selected_postposition === 'ey') || (selected_postposition
     === '(u)lo')){
for(var i = 467; i < 484 ; i++){
svgSection.selectAll("#index"+i).remove();
}
```

```
}

EpochNow = updateTween()

right_top_draw();
right_middle_draw();
right_bottom_draw();

var textlabel = div_epoch.selectAll(".textlabel")
textlabel.enter()
.append("text")
.attr("class", "label")
.text((EpochNow+1)+" Epoch")
textlabel.transition()
.duration(10)
.text((EpochNow+1)+" Epoch")
textlabel.exit().remove();

var data = {};
for (var i = 0; i < Map_info.length ; i++) {
if ((Map_info[i].postposition === selected_postposition) && (
    Map_info[i].epoch === 'epoch'+EpochNow)) {
var innerArray = [];
for(var j = 0; j < Sentence_info.length ; j++){
if (Sentence_info[j].postposition === selected_postposition &&
     (Map_info[i].epoch === 'epoch'+EpochNow)) {
for(var k = 0; k < Map_info[i].sentences.length ; k++){
var sentenceDic = {}
sentenceDic["index"] = Sentence_info[j].sentences[k].index
sentenceDic["function"] = Sentence_info[j].sentences[k].
    function
```

```
sentenceDic["sentence"] = Sentence_info[j].sentences[k].
    sentence
sentenceDic["sentence_pos"] = Sentence_info[j].sentences[k].
    sentence_pos
sentenceDic["X"] = Map_info[i].sentences[k].X
sentenceDic["Y"] = Map_info[i].sentences[k].Y
if((functionarray.length > 0 == true)&&(indexarray.length > 0
    == true)){

var checked = false;
for(var q = 0; q < functionarray.length ; q++){
if(Sentence_info[j].sentences[k].function == functionarray[q])
    {
checked = true;
}
}

if(checked == true){
var ichecked = false;
for(var t = 0; t < indexarray.length ; t++){
if(Sentence_info[j].sentences[k].index == indexarray[t]){
ichecked = true;
}
}
if(ichecked == true){
sentenceDic["opacity_value"] = 0.9
} else {
sentenceDic["opacity_value"] = 0.2
}
} else {
sentenceDic["opacity_value"] = 0.2
```

```
}
} else if((functionarray.length > 0 == false)&&(indexarray.length > 0 == true)){
var checked = false;
for(var q = 0; q < indexarray.length ; q++){
if(Sentence_info[j].sentences[k].index == indexarray[q]){
checked = true;
}
}
if(checked == true){
sentenceDic["opacity_value"] = 0.9
} else {
sentenceDic["opacity_value"] = 0.2
}
} else if((functionarray.length > 0 == true)&&(indexarray.length > 0 == false)){
var checked = false;
for(var q = 0; q < functionarray.length ; q++){
if(Sentence_info[j].sentences[k].function == functionarray[q]){
checked = true;
}
}
if(checked == true){
sentenceDic["opacity_value"] = 0.9
} else {
sentenceDic["opacity_value"] = 0.2
}
} else {
sentenceDic["opacity_value"] = 0.7
}
```

```
innerArray.push(sentenceDic)
}
}
}
data["postposition"] = Map_info[i].postposition
data["epoch"] = Map_info[i].epoch
data["sentences"] = innerArray
}
}

currentPost = ''

if (selected_postposition === 'ey') {
currentPost = 'Ey'
} else if (selected_postposition === 'eyse') {
currentPost = 'Eyse'
} else if (selected_postposition === '(u)lo') {
currentPost = 'Lo'
}

LeftsvgSection.selectAll(".PNG").remove();

var imgs = LeftsvgSection.append("image")
.attr("class", "PNG")
.attr("xlink:href", "https://seongmin-mun.github.io/
    VisualSystem/Major/PostBERT.ko/images/densityClusterPNG_r/"
    +currentPost+"_tSNE_epoch_"+EpochNow+".png")
.attr("x", LeftsectionWidth*0.05)
.attr("y", 0)
.attr('width', LeftsectionWidth*0.9)
.attr('height', LeftsectionWidth*0.9);
```

```
var w = sectionWidth;
var h = sectionHeight;
var padding = (sectionHeight*0.12);

var xScale = d3.scale.linear()
.domain([d3.min(data.sentences, function(d) { return d.Y; }),
   d3.max(data.sentences, function(d) { return d.X; })])
.range([0+padding, w-padding]);

var yScale = d3.scale.linear()
.domain([d3.min(data.sentences, function(d) { return d.Y; }),
   d3.max(data.sentences, function(d) { return d.Y; })])
.range([h-padding, 0+padding]);

var circle = NodeGroup.selectAll(".nodedot")
.data(data.sentences);

circle.enter()
.append("circle")
.attr("class", "nodedot")
.attr("id", function (d) {
return d.index
})
.attr("cx", function (d) {
return xScale(d.X)
})
.attr("cy", function (d) {
return yScale(d.Y)
})
.attr("r", 3)
```

```
.attr("fill", function (d) {
if (d.function == function_name[0]) {
return function_color[0]
} else if (d.function == function_name[1]) {
return function_color[1]
} else if (d.function == function_name[2]) {
return function_color[2]
} else if (d.function == function_name[3]) {
return function_color[3]
} else if (d.function == function_name[4]) {
return function_color[4]
} else if (d.function == function_name[5]) {
return function_color[5]
} else if (d.function == function_name[6]) {
return function_color[6]
} else if (d.function == function_name[7]) {
return function_color[7]
} else if (d.function == function_name[8]) {
return function_color[8]
} else if (d.function == function_name[9]) {
return function_color[9]
}
})
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", function (d) {
return d.opacity_value
})
.style("cursor", "help")
.on("mouseover", function (d) {
d3.select(this)
```

```
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", 1)
.attr("fill","#FF0000")
})
.on("mouseout", function (d) {
d3.select(this)
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", function (d) {
return d.opacity_value
})
.attr("fill", function (d) {
if (d.function == function_name[0]) {
return function_color[0]
} else if (d.function == function_name[1]) {
return function_color[1]
} else if (d.function == function_name[2]) {
return function_color[2]
} else if (d.function == function_name[3]) {
return function_color[3]
} else if (d.function == function_name[4]) {
return function_color[4]
} else if (d.function == function_name[5]) {
return function_color[5]
} else if (d.function == function_name[6]) {
return function_color[6]
} else if (d.function == function_name[7]) {
return function_color[7]
} else if (d.function == function_name[8]) {
return function_color[8]
```

```
} else if (d.function == function_name[9]) {
return function_color[9]
}
});
})
.on("mouseenter", function (d) {
div_inner.transition()
.duration(200)
.style("opacity", 0.85);
div_inner.html("<strong>Selected sentence</strong><br/><h5>
    Index : "+d.index + "<h5/><h5>Function : "  + d.function +
    "<h5/><h5>Sentence : "  + d.sentence+ "<h5/><h5>SentencePOS
     : "  + d.sentence_pos+ "<h5/>")
.style("left", "20px")
.style("top", sectionHeight*0.07+"px");
})
.on("mouseleave", function () {
div_inner.transition()
.duration(500)
.style("opacity", 0);
})

circle.transition()
.duration(2000)
.attr("cx", function (d) {
return xScale(d.X)
})
.attr("cy", function (d) {
return yScale(d.Y)
})
.attr("r", 3)
```

```
.attr("fill", function (d) {
if (d.function == function_name[0]) {
return function_color[0]
} else if (d.function == function_name[1]) {
return function_color[1]
} else if (d.function == function_name[2]) {
return function_color[2]
} else if (d.function == function_name[3]) {
return function_color[3]
} else if (d.function == function_name[4]) {
return function_color[4]
} else if (d.function == function_name[5]) {
return function_color[5]
} else if (d.function == function_name[6]) {
return function_color[6]
} else if (d.function == function_name[7]) {
return function_color[7]
} else if (d.function == function_name[8]) {
return function_color[8]
} else if (d.function == function_name[9]) {
return function_color[9]
}
})
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", function (d) {
return d.opacity_value
})
.style("cursor", "help")
.on("mouseover", function (d) {
d3.select(this)
```

```
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", 1)
.attr("fill","#FF0000")
})
.on("mouseout", function (d) {
d3.select(this)
.attr("stroke", "black")
.attr("stroke-width", "1px")
.attr("opacity", function (d) {
return d.opacity_value
})
.attr("fill", function (d) {
if (d.function == function_name[0]) {
return function_color[0]
} else if (d.function == function_name[1]) {
return function_color[1]
} else if (d.function == function_name[2]) {
return function_color[2]
} else if (d.function == function_name[3]) {
return function_color[3]
} else if (d.function == function_name[4]) {
return function_color[4]
} else if (d.function == function_name[5]) {
return function_color[5]
} else if (d.function == function_name[6]) {
return function_color[6]
} else if (d.function == function_name[7]) {
return function_color[7]
} else if (d.function == function_name[8]) {
return function_color[8]
```

```
} else if (d.function == function_name[9]) {
return function_color[9]
}
});
})
.on("mouseenter", function (d) {
div_inner.transition()
.duration(200)
.style("opacity", 0.85);
div_inner.html("<strong>Selected sentence</strong><br/><h5>
    Index : "+d.index + "<h5/><h5>Function : "  + d.function +
    "<h5/><h5>Sentence : "  + d.sentence+ "<h5/><h5>SentencePOS
     : "  + d.sentence_pos+ "<h5/>")
.style("left", "20px")
.style("top", sectionHeight*0.07+"px");
})
.on("mouseleave", function () {
div_inner.transition()
.duration(500)
.style("opacity", 0);
});

circle.exit().remove();
}
})
</script>
</body>
</html>
```