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

Seongmin Mun

Université Paris Nanterre
200 av. de la République
92001 Nanterre Cedex
www.parisnanterre.fr

École doctorale 139 :
Connaissance, langage, modélisation
MoDyCo: L'UMR 7114 Modèles, Dynamiques, Corpus

Membre de l'université Paris Lumières

Seongmin MUN

La résolution de la polysémie à l'aide de modèles de vecteur de mots et la visualisation de données :

le cas des postpositions adverbiales -ey, -eyse, et -(u)lo en coréen

Thèse présentée et soutenue publiquement le 18/06/2021
en vue de l'obtention du doctorat de Sciences du langage de l'Université Paris Nanterre sous la direction de M. Guillaume Desagulier

Jury * :

Rapporteur·e :	Prof. Laurent PRÉVOT	PR, membre externe, Aix-Marseille Université
Rapporteur·e :	Prof. Iksoo KWON	PR, membre externe, Hankuk University of Foreign Studies
Membre du jury :	Prof. Delphine BATTISTELLI	PR, membre interne, Université Paris Nanterre
Membre du jury :	Prof. Iris TARAVELLA	PR, membre interne, Université Paris Nanterre
Membre du jury :	Dr. Caroline BRUN	Senior Researcher, Naver Labs Europe

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

Listings

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 construct 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 being 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 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 frequently used 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 LOC (Lo-

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

cation) as in (1)[2].

(1) 지붕 위에 구멍이 났다.
cipung wi-ey kwumeng-i na-ss-ta.
Roof top-LOC hole-NOM appear-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*. 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 relationships 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 too broad and not accurate or specific enough to capture all the essential functions of *-ey*. Based on 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). Another 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. In a more practical perspective, Song (2014) suggested the main function as an indication of a location or movement of a physical target. He also explained that extending the function could be interpreted 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 place 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 phamwuthye cinay-n-ta.
He-TOP all day study room-LOC bury in 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 내리꽂혔다.
Chelswu-ka tenci-n khal-i ttangpatak-ey naylyekkochy-ess-ta.
Chelswu-TOP throw knife-NOM ground-GOL stick-PST-DECL

'The knife thrown by Chelswu 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.
mwuntul-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 standard for quantitative classification of the specific property 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 directly receives 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
hyentayintul-un motwu chamtoyn cisik-ey
modern people-PL-TOP all true knowledge-THM
허기져있다.
hekicye-iss-ta.
hungry-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 있었다.
ku elin sonye-nun hwalospwul-ey son-ul noki-ko iss-ess-ta.
That young boy-TOP fire-INS hand-ACC melt-and 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 cecitoy-ess-ta.
street go out-NOM police-AGT stop-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	pakkwun-ul	chwuchenhay-ess-ta.
professor Kim-TOP	assistant-FNS	Pakk-ACC	recommend-PST-DECL

'Professor Kim recommended Pakk 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 ppopaolli-n-ta.
miner-PL-NOM sea-SRC oil-ACC pull-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 thayen-ass-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 above (8). 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 chopingtoy-ess-ta.
He-TOP representative lecturer-FNS invite-PST-DECL

'He was invited as a representative lecturer.'

The definition of instrument (INS) is the same as described above (6). 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	kamky-ess-ta.
wire-NOM	connection wire-INS	wind-PST-DECL

'The wire wound around with the connection wire.'

Direction (DIR) is a function to indicate the direction of the point at which the preceding word is directed. 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 *into* in English.

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

범인은	어두운	골목으로	달아났다.
pemi-nun	etwuwun	kolmok-ulo	talan-ass-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 above (3). 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	isssupni-ta.
patient-NOM	stomach cancer-EFF	very	suffer-and	be-DECL

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

The definition of criterion (CRT) is the same as described above (4). 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	time	interval-CRT	arrange-PST-DECL

'It was arranged at appropriate time intervals.'

The definition of LOC is the same as described above (1). 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	압송하다.
kyengchal-i	phiuyca-lul	kemchal-lo	apsongha-ta.
police-NOM	suspect-ACC	prosecution-LOC	transport do-DECL

'The police transport 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 follow this trend and develop 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 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).

(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, 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 en-

vironments 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 room.'

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

pakk-ulo nao-ass-ta.
outside-DIR go out-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 Mar-

tin, 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*.

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

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

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

$$u_1, u_2 = 1, 2$$

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

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

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

$$18u_1 + 36u_2 = 0$$

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

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

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.

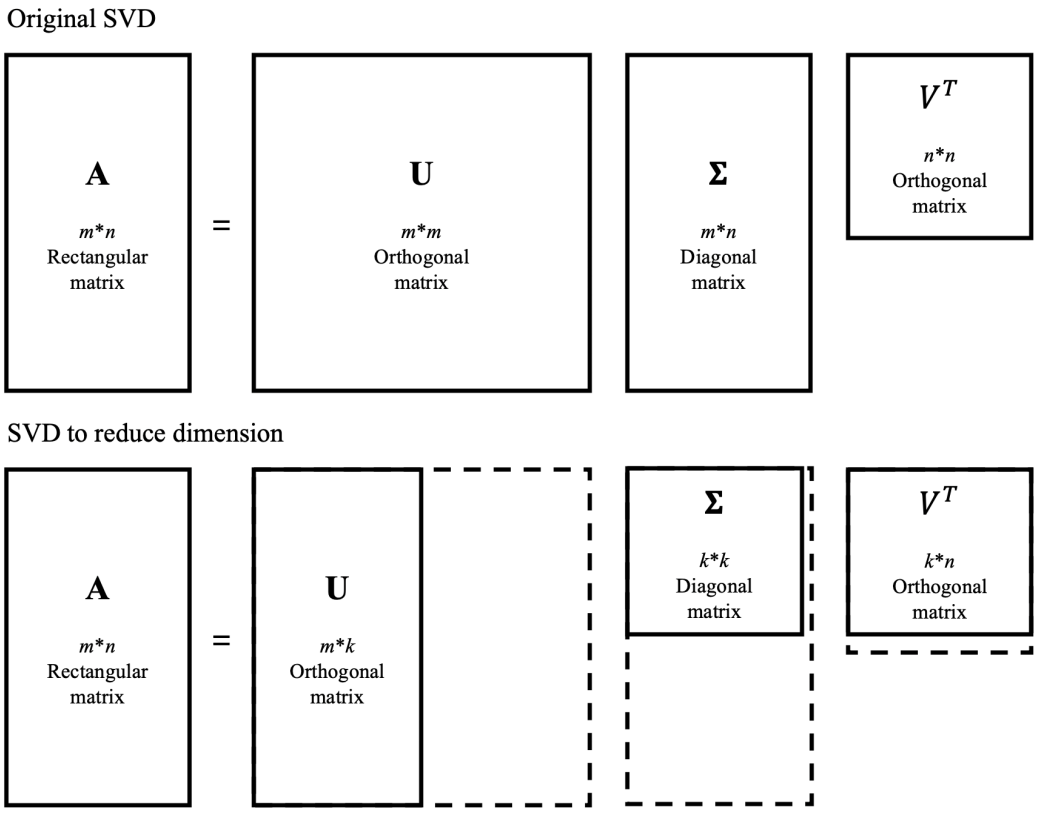

Figure 3.1: Original SVD and SVD to reduce dimension

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 disser-

tation, 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 nao-ta.
outside-DIR go out-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.

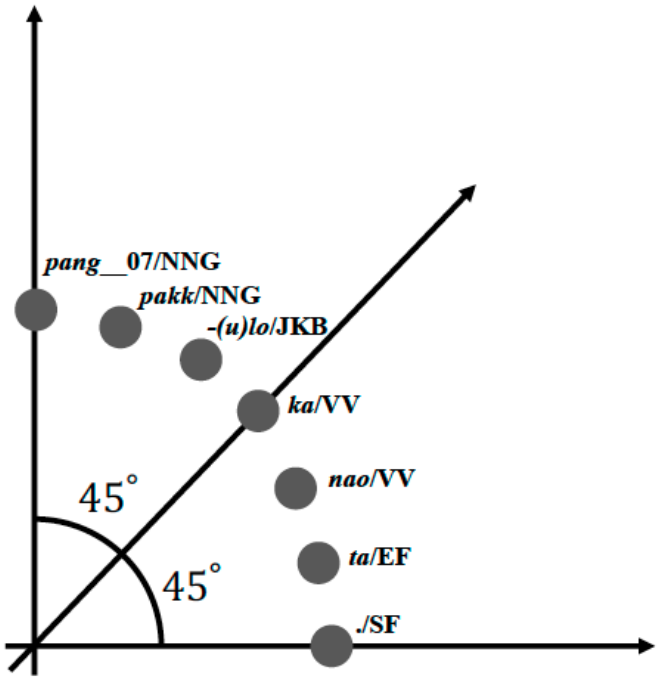

Figure 3.2: Visualization of results through the one-hot encoding

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.

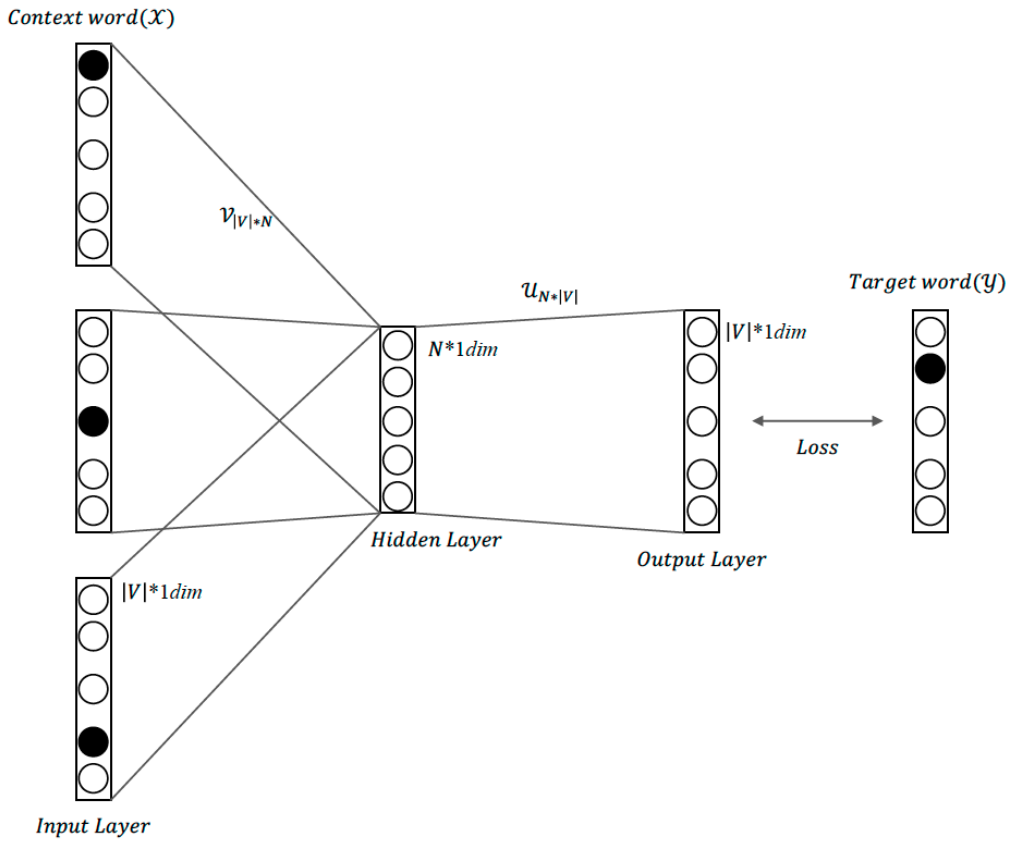

Figure 3.3: A framework of the CBOW model

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 probabil-

ities.

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

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.

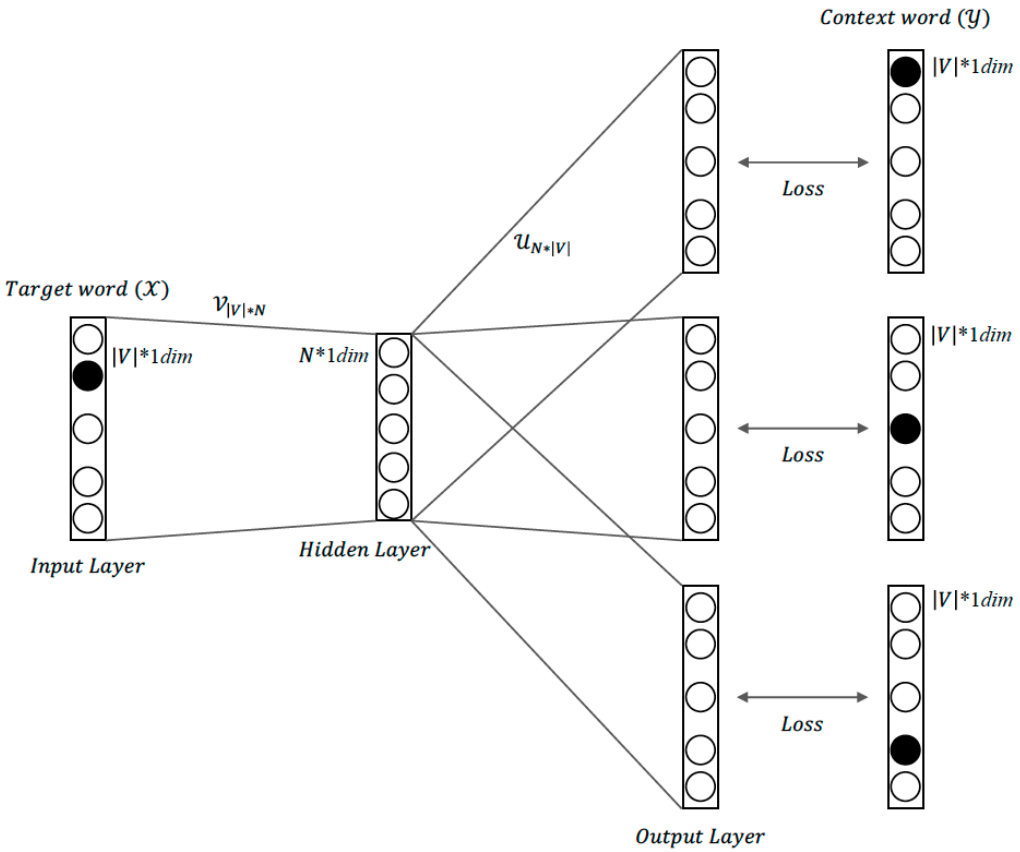

Figure 3.4: A framework of the Skip-gram model

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 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, I 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.

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).

The Sejong electronic dictionary consists of 31,093 frames involving 15,181

[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

```
<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

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 sen-

tences 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 inter-rater 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].

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

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

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 order of FNS, INS, and DIR 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).

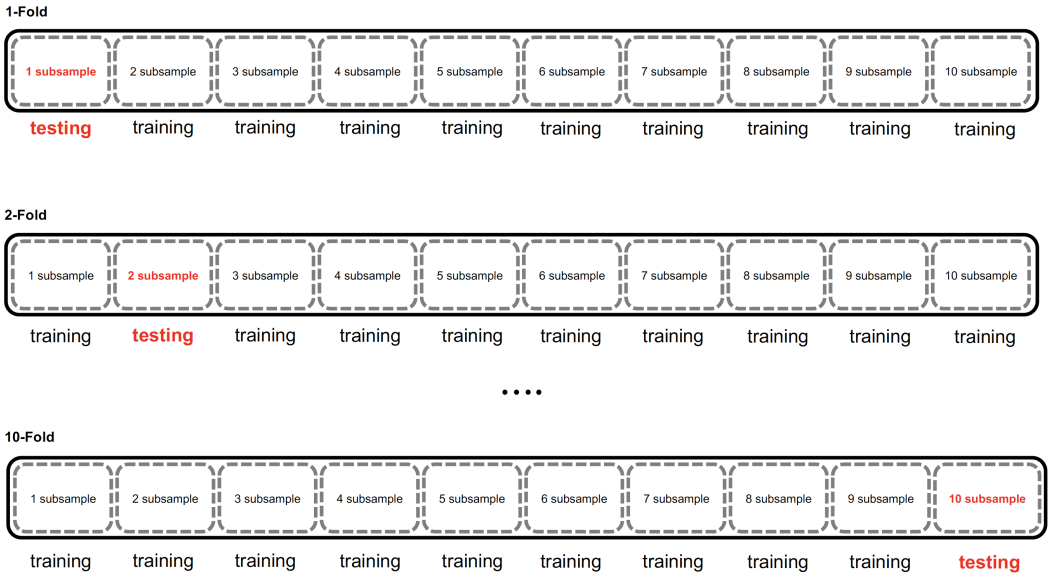

Figure 4.4: The process of the *k*-fold cross-validation technique

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 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 the meaning of the target word even it 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 the training set and test 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 postpositions * 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.

[3]The entire code for the word-level embedding models that I developed are available at: PPMI-SVD and SGNS

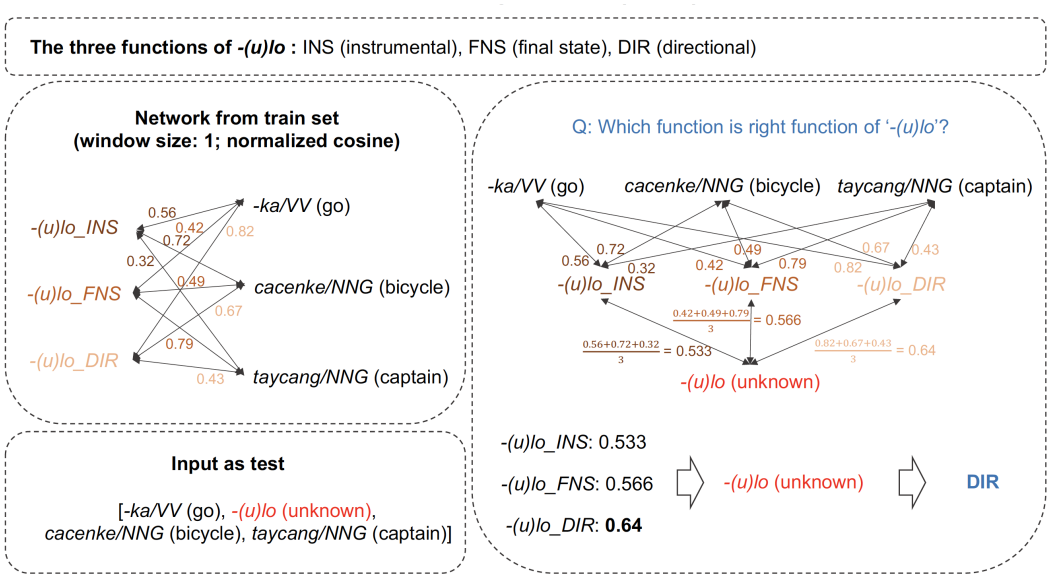

Figure 4.7: The classification model process adapted from Dagan et al. (1993): a case of *-(u)lo*

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.

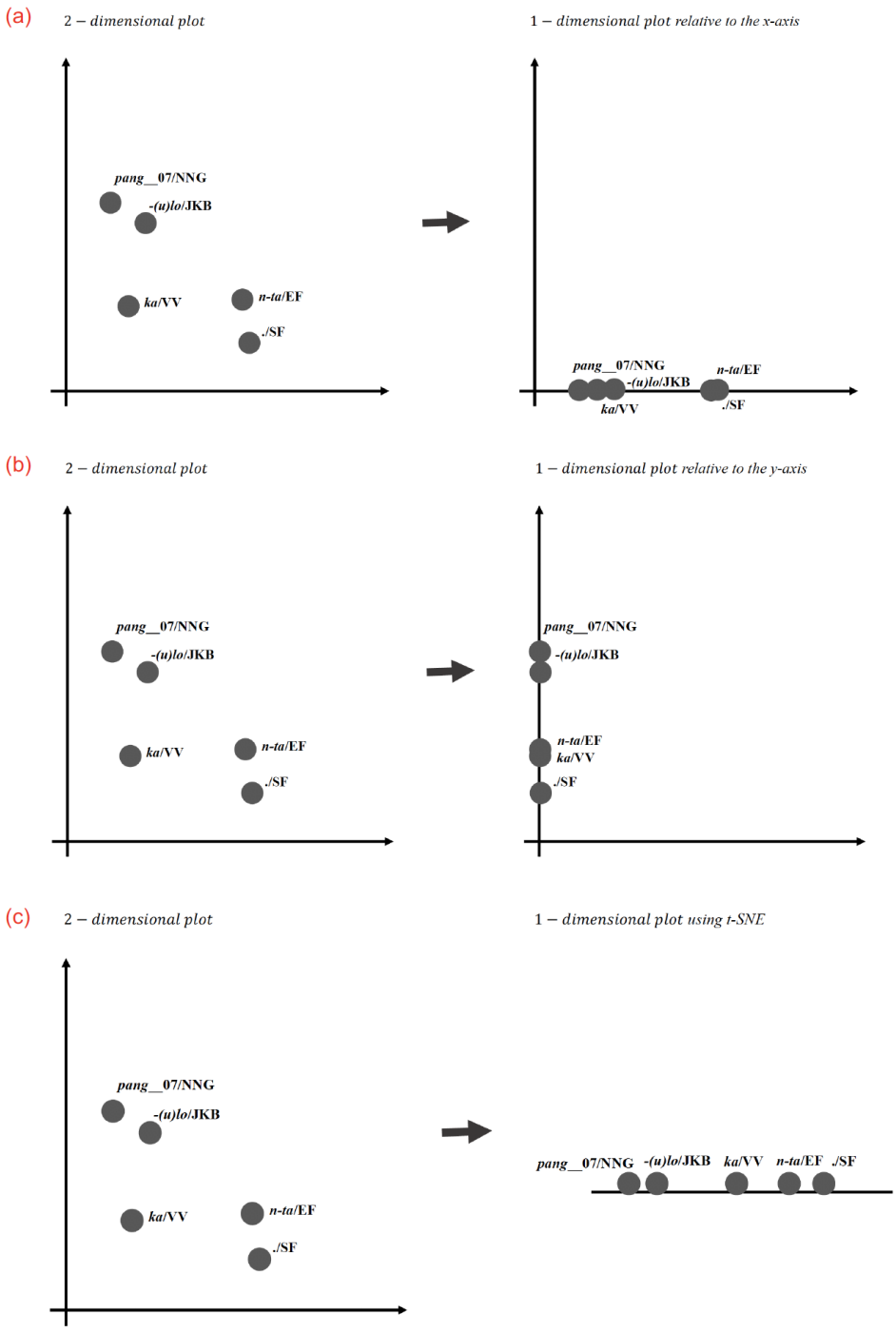

Figure 4.8: Reducing a two-dimensional plot to a one-dimensional plot using the t-SNE

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.

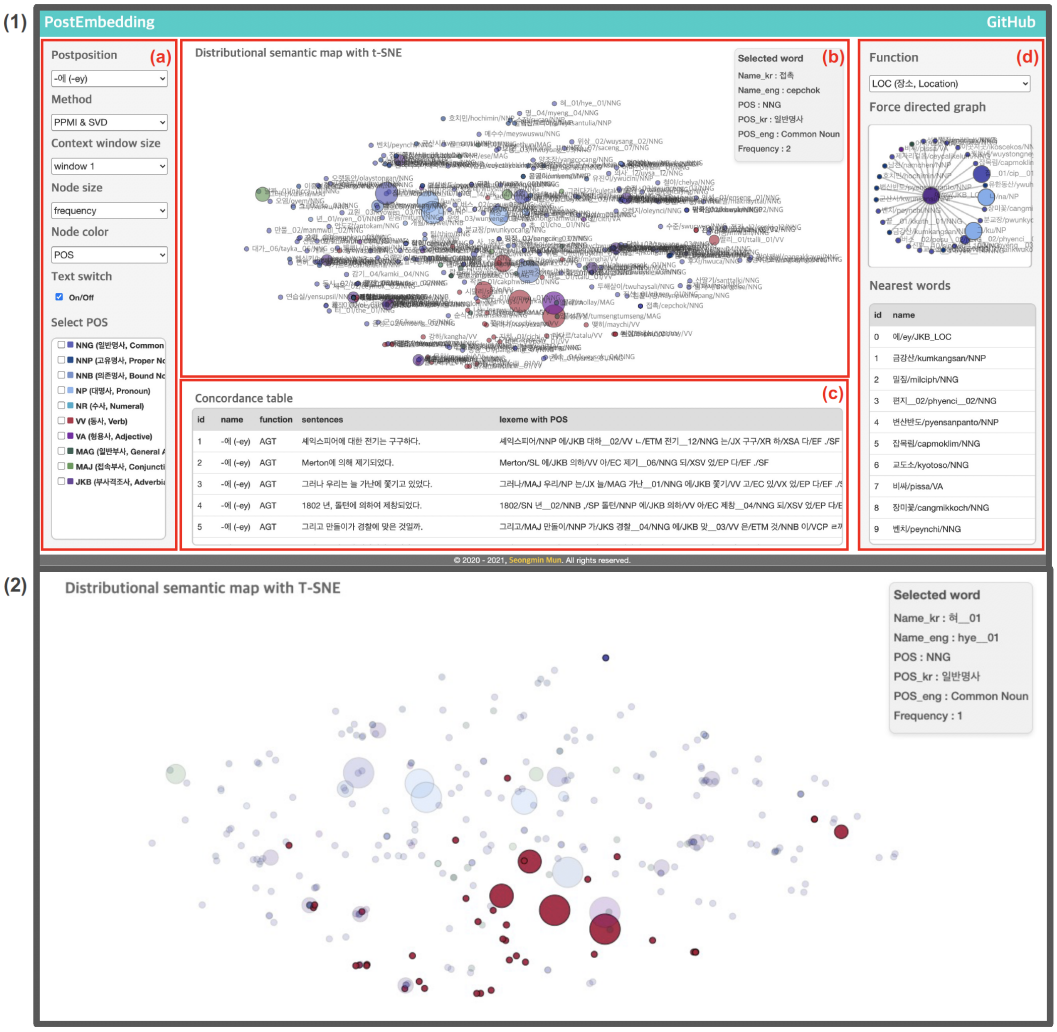

Figure 4.9: Interface of visualization system (1) and the main view of the system (2)

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.

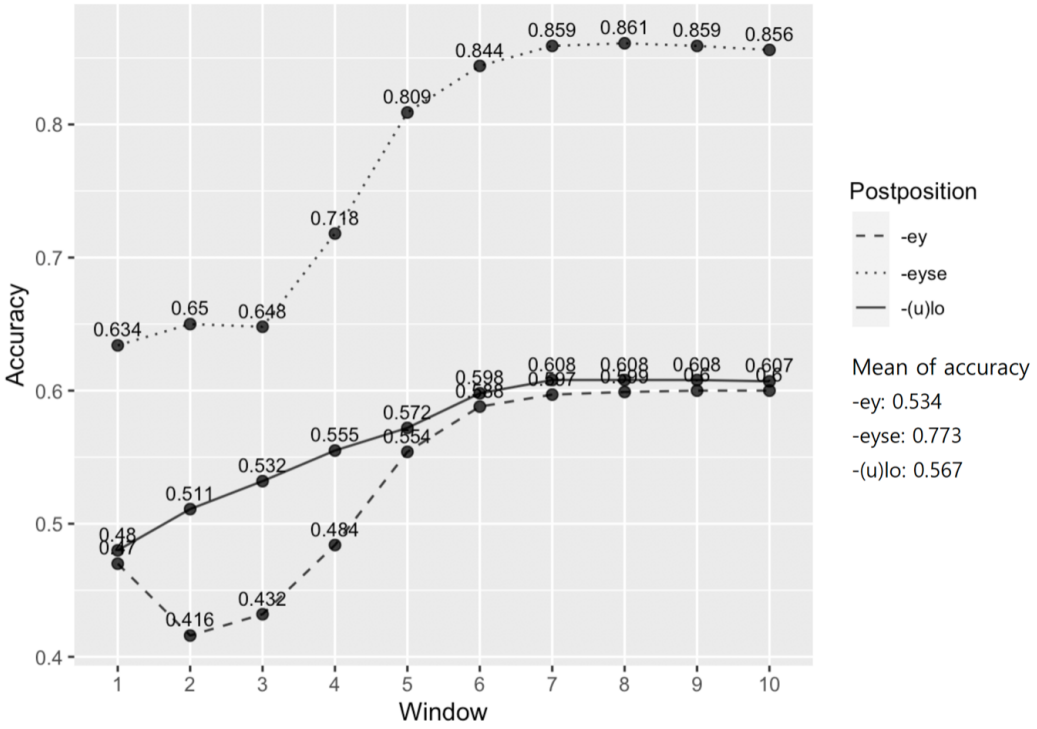

Figure 5.1: Classification accuracy by window size for the PPMI-SVD model

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	$\lvert t \rvert$	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).

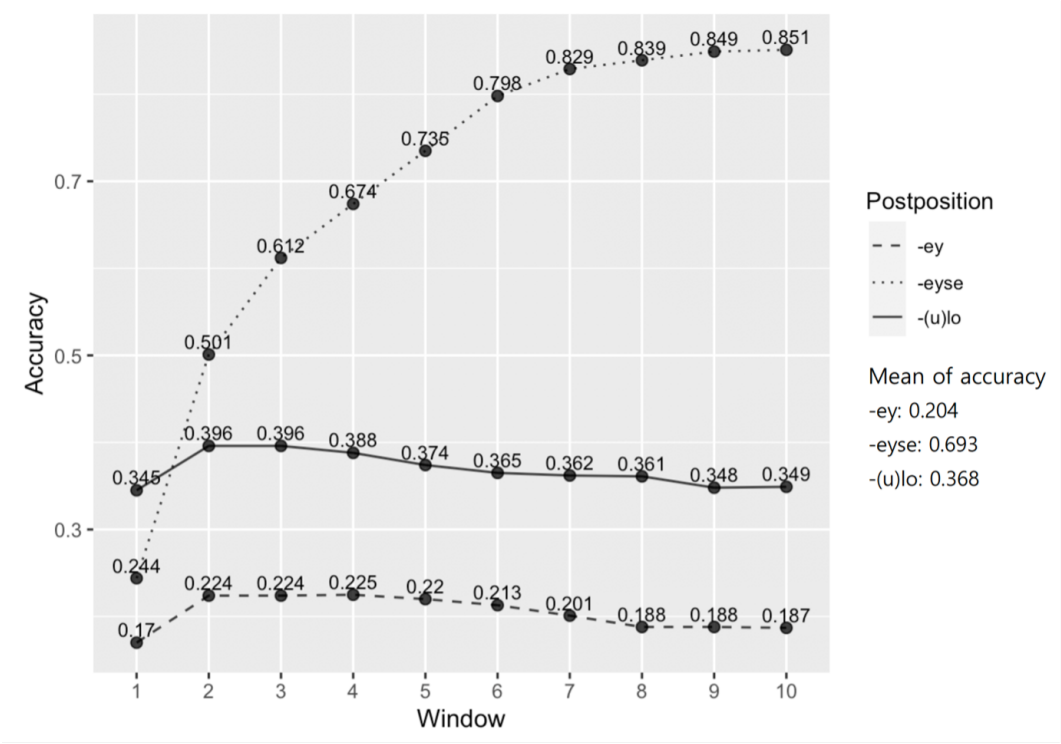

Figure 5.2: Classification accuracy by window size for the SGNS model

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.

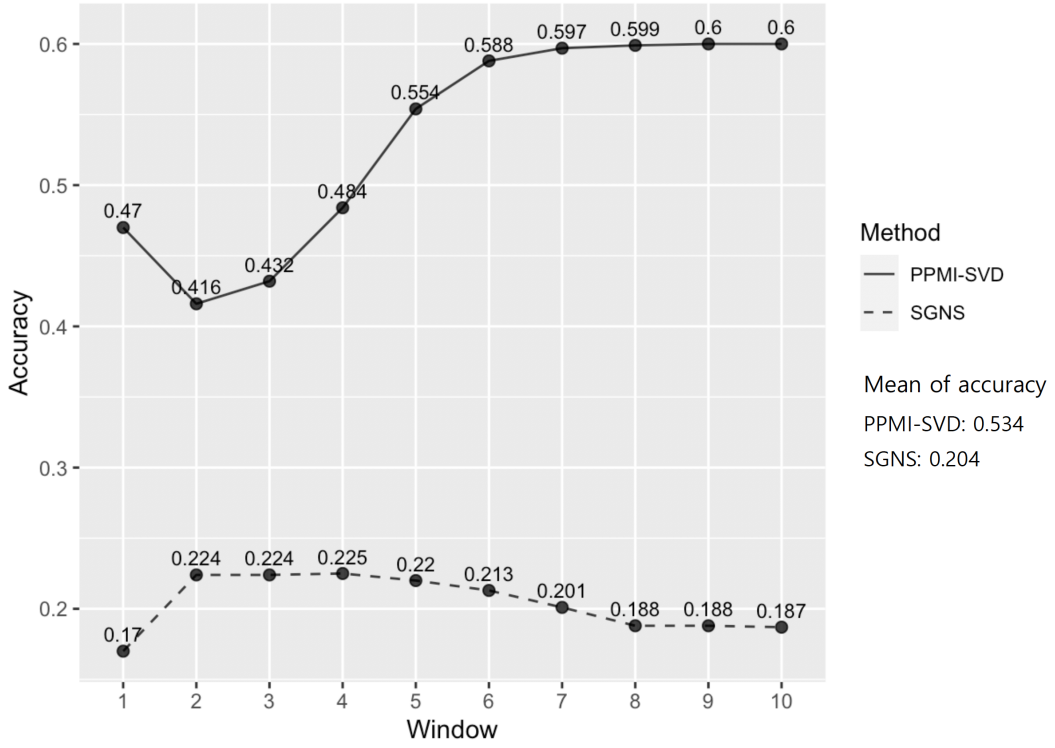

Figure 5.3: By-window-size accuracy for the two models: *-ey*

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.

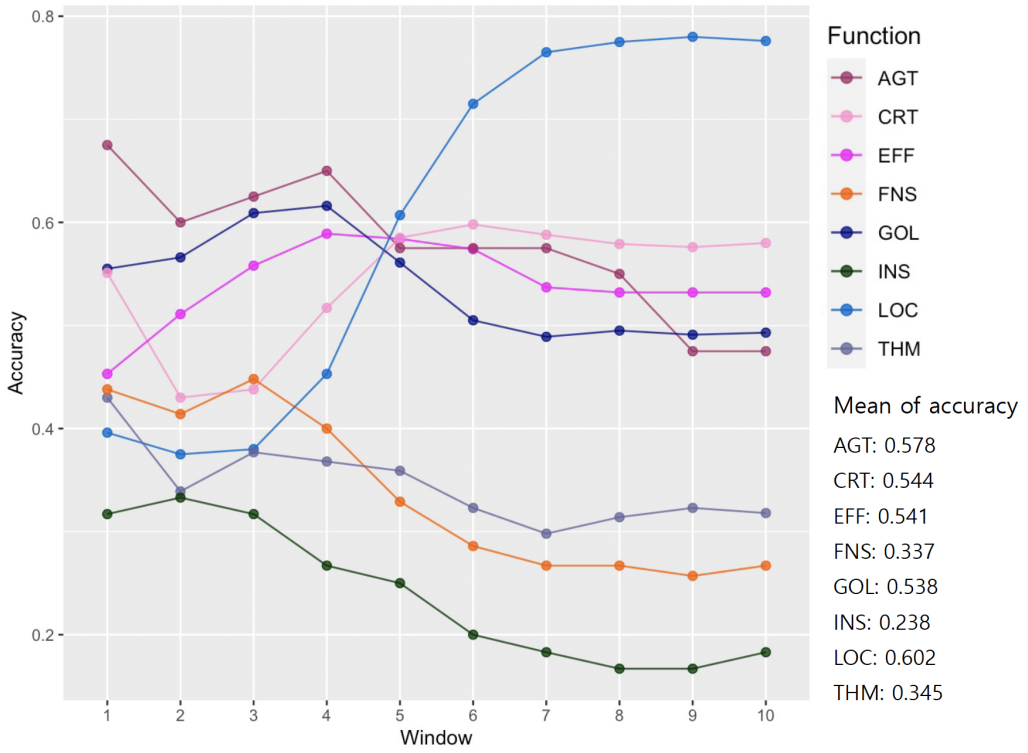

Figure 5.4: By-function accuracy curve for the PPMI-SVD model: *-ey*

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.

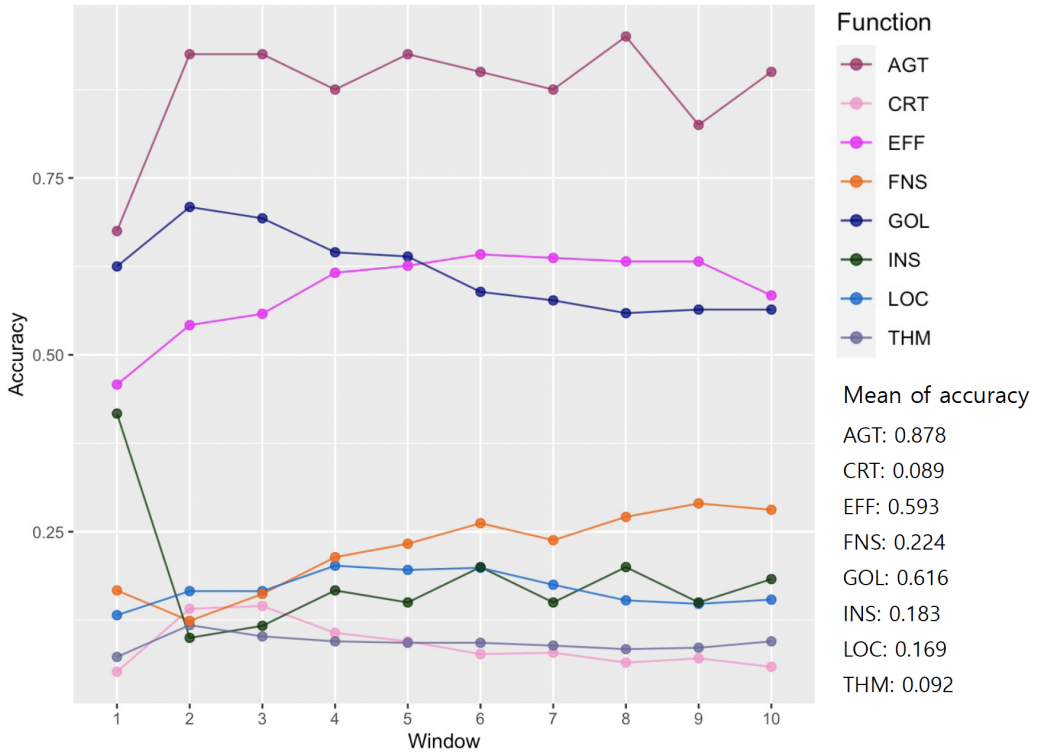

Figure 5.5: By-function accuracy curve for the SGNS model: *-ey*

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).

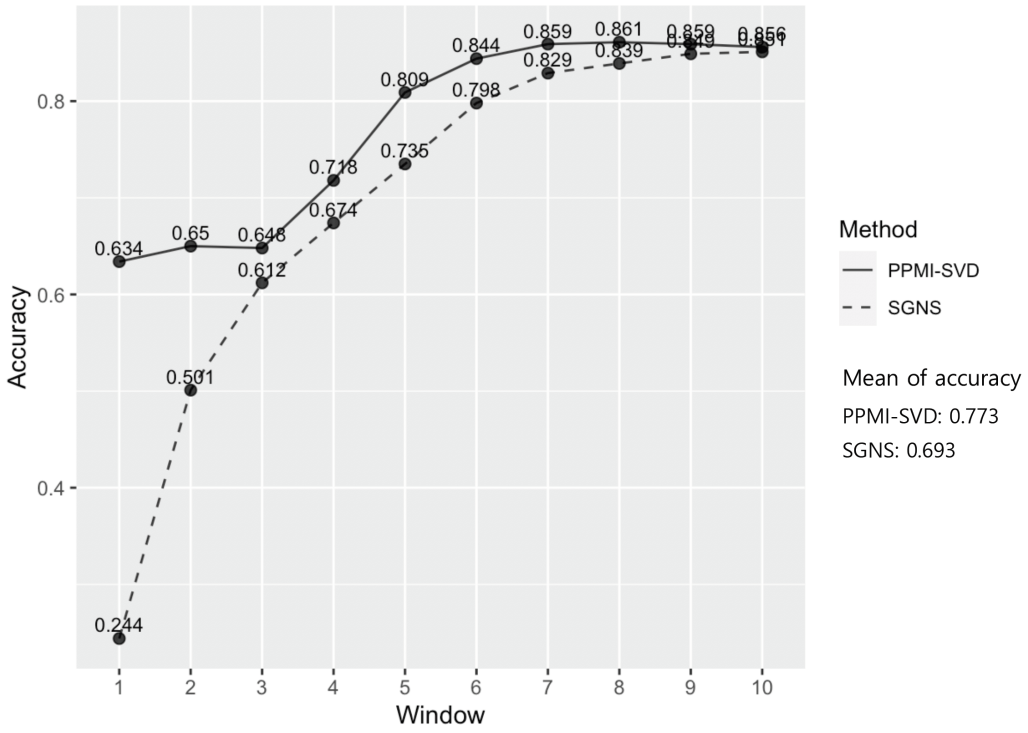

Figure 5.6: By-window-size accuracy for the two models: *-eyse*

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.

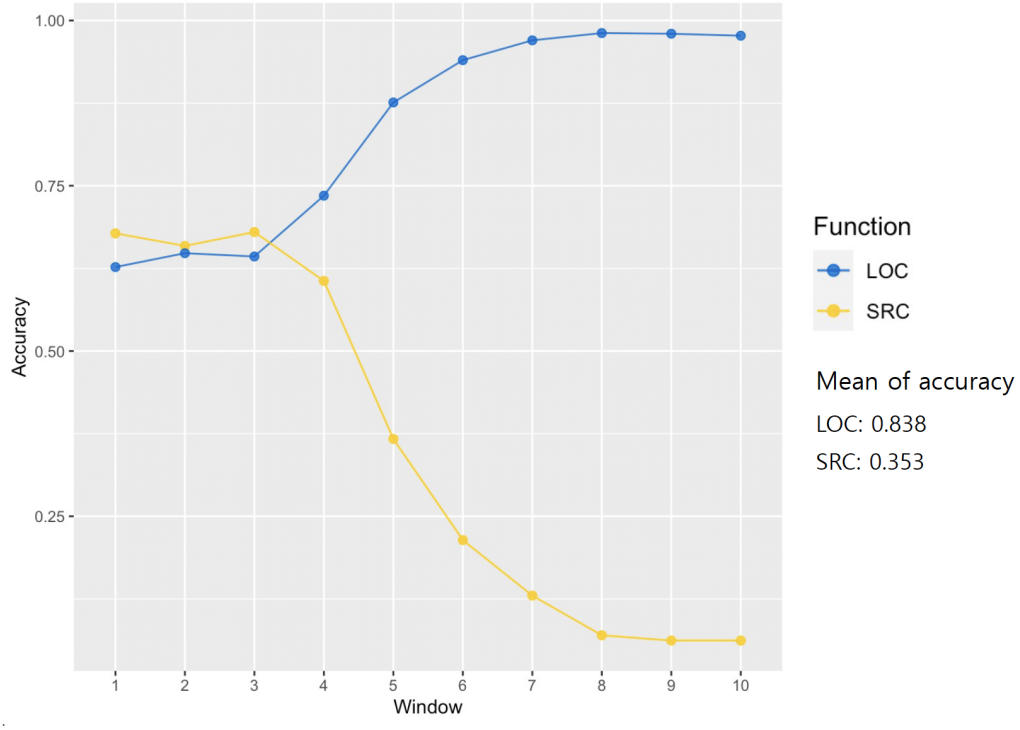

Figure 5.7: By-function accuracy curve for the PPMI-SVD model: *-eyse*
Note. Abbreviation: LOC = location; SRC = source

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.

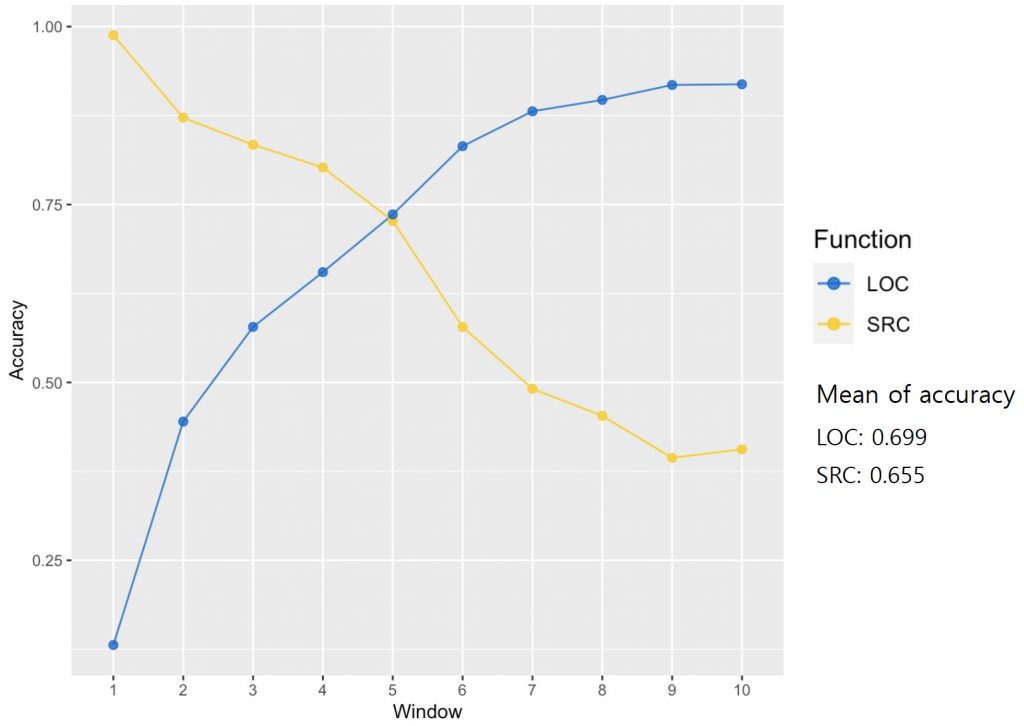

Figure 5.8: By-function accuracy curve for the SGNS model: *-eyse*

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).

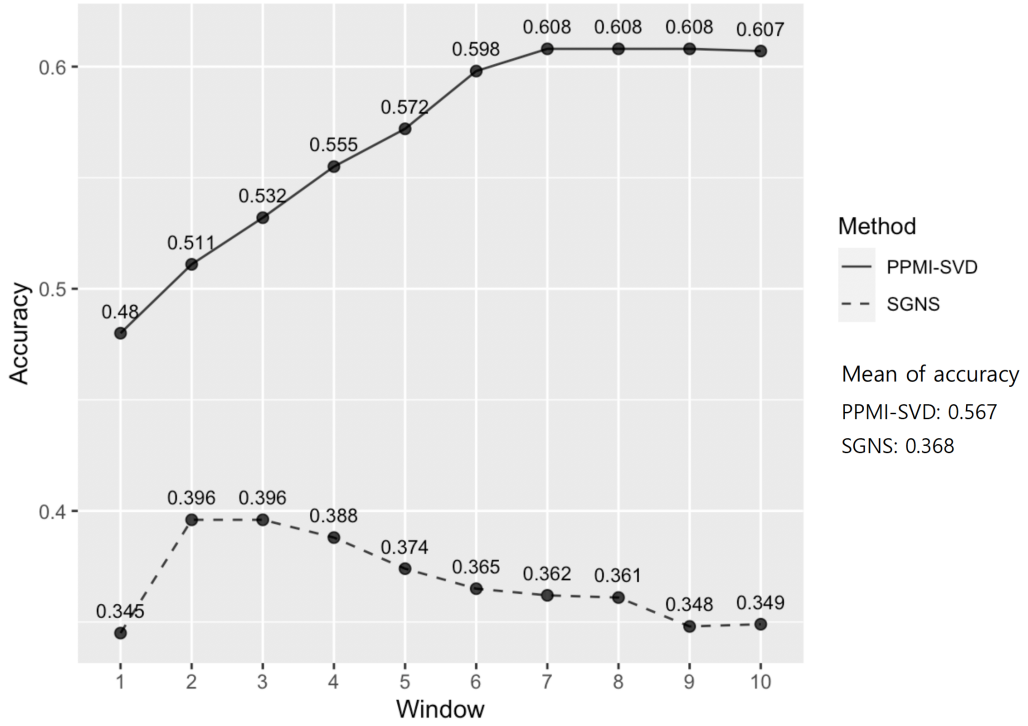

Figure 5.9: By-window-size accuracy for the two models: *-(u)lo*

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.

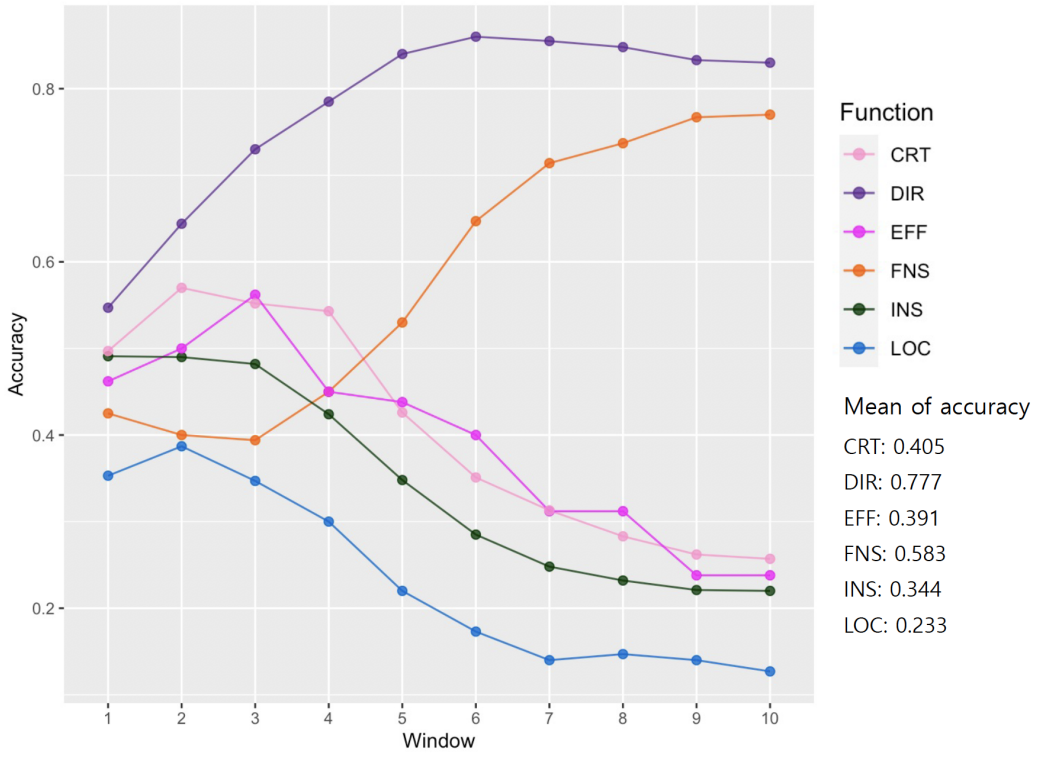

Figure 5.10: By-function accuracy curve for the PPMI-SVD model: *-(u)lo*

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.

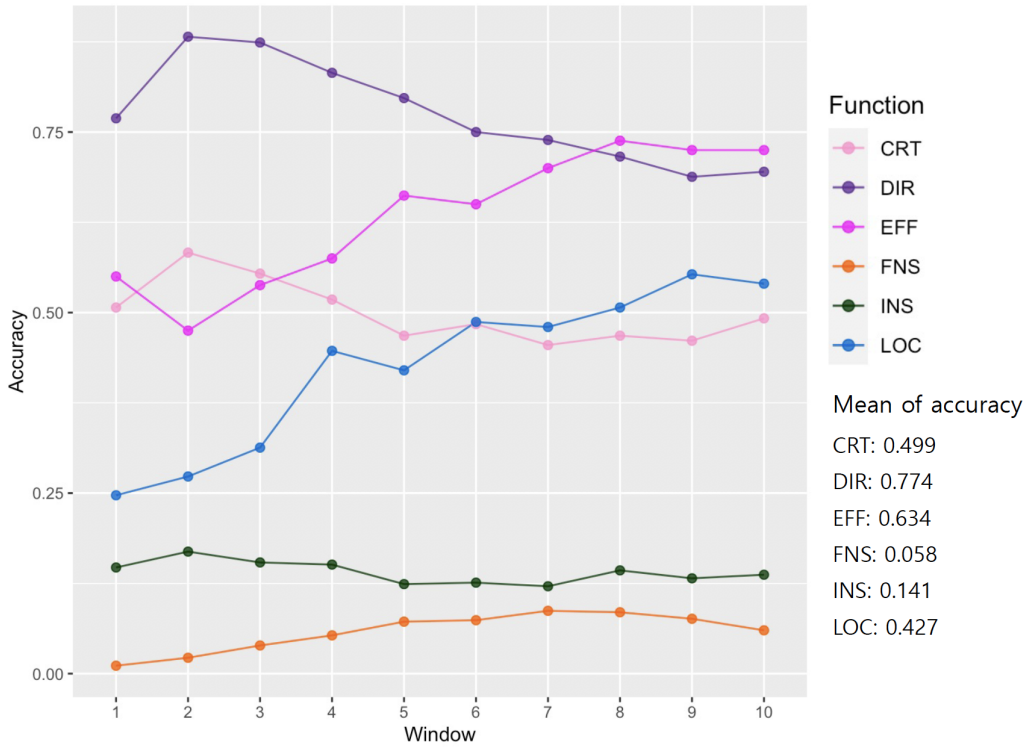

Figure 5.11: By-function accuracy curve for the SGNS model: *-(u)lo*

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 post-position. 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 (available at: PostEmbedding).

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

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.

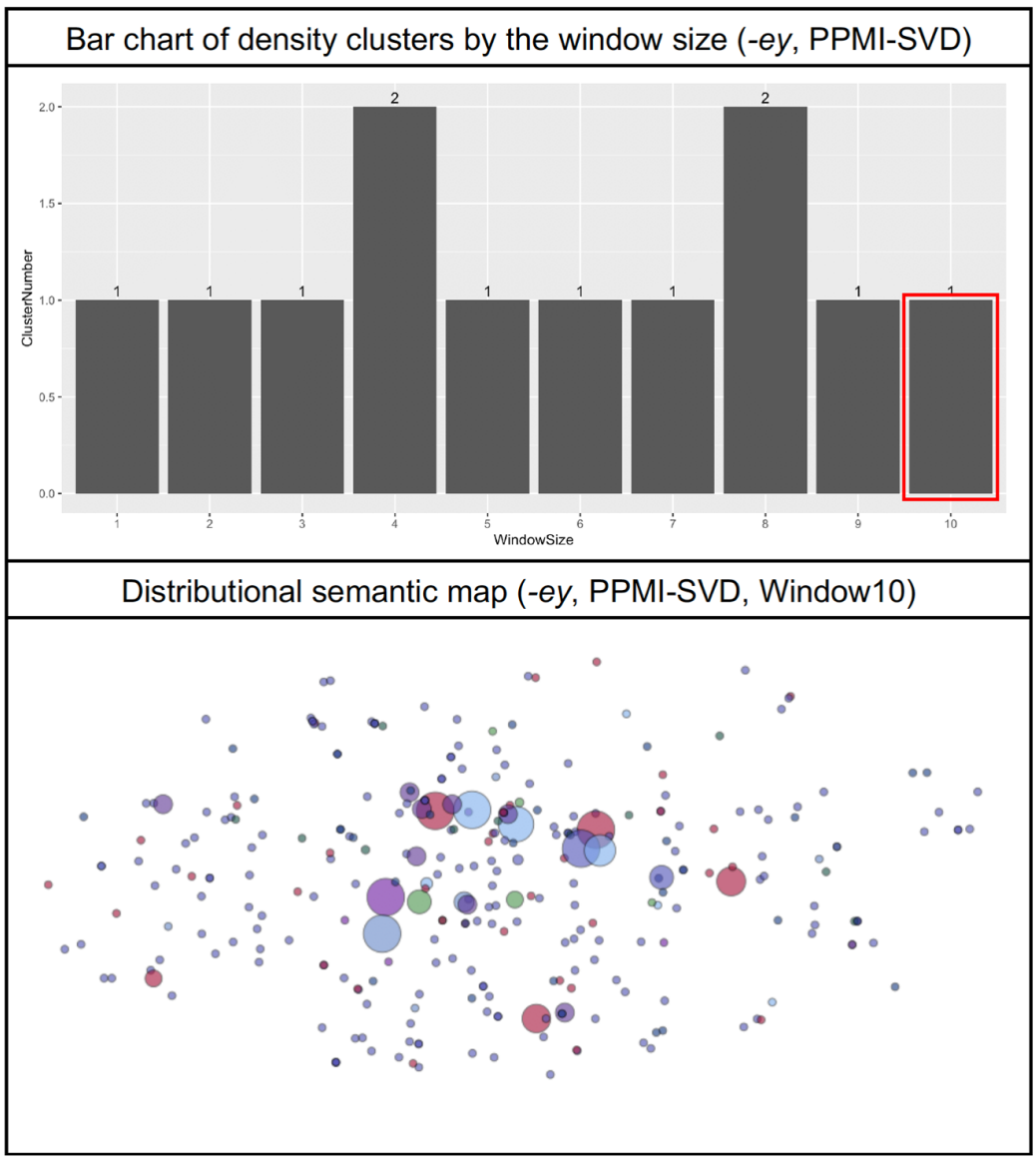

Figure 5.12: Bar chart of density cluster result and distributional semantic map for *-ey* (PPMI-SVD). Red in graph = the size of window showing the highest accuracy.

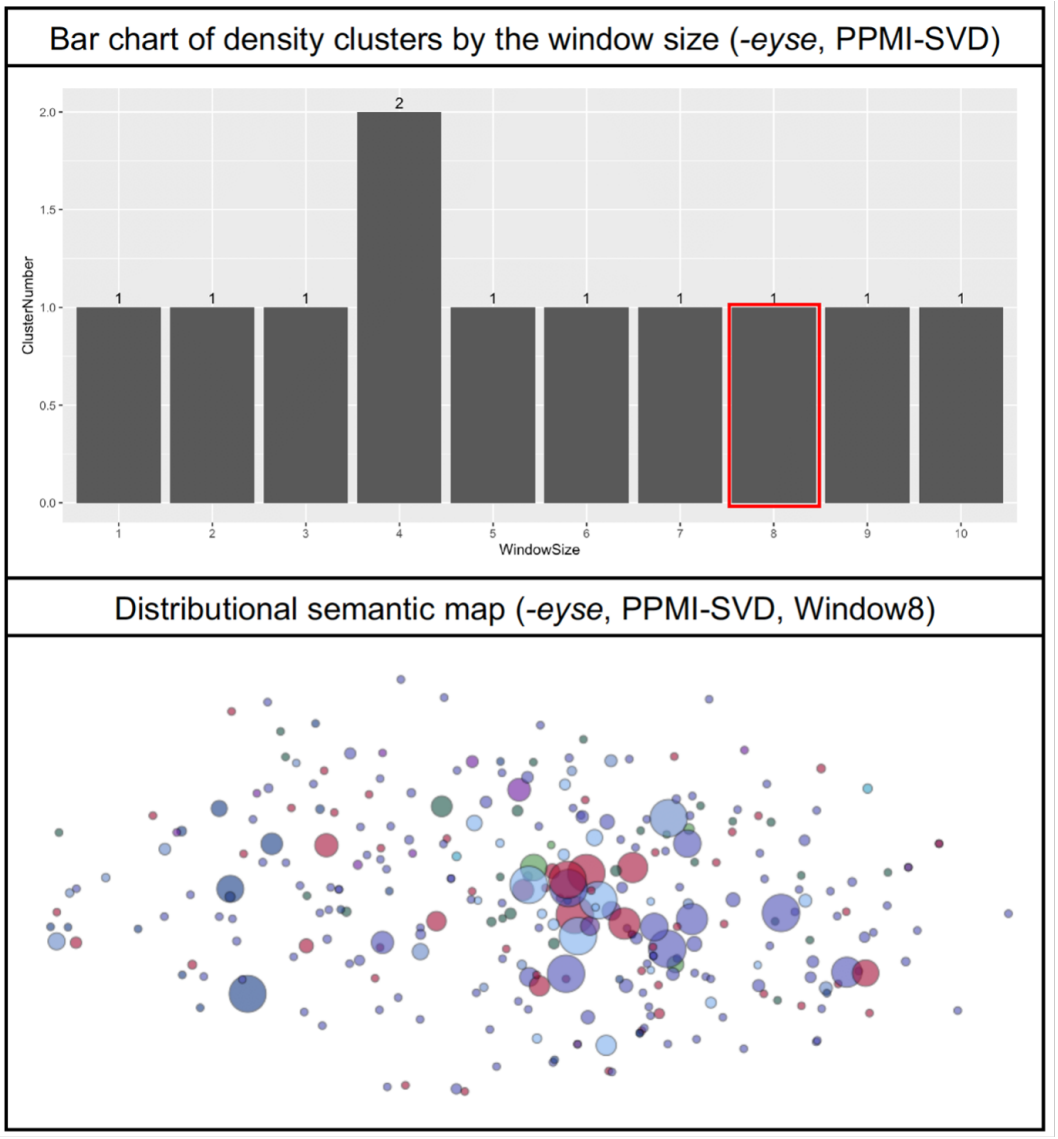

Figure 5.13: Bar chart of density cluster result and distributional semantic map for *-eyse* (PPMI-SVD). Red in graph = the size of window showing the highest accuracy.

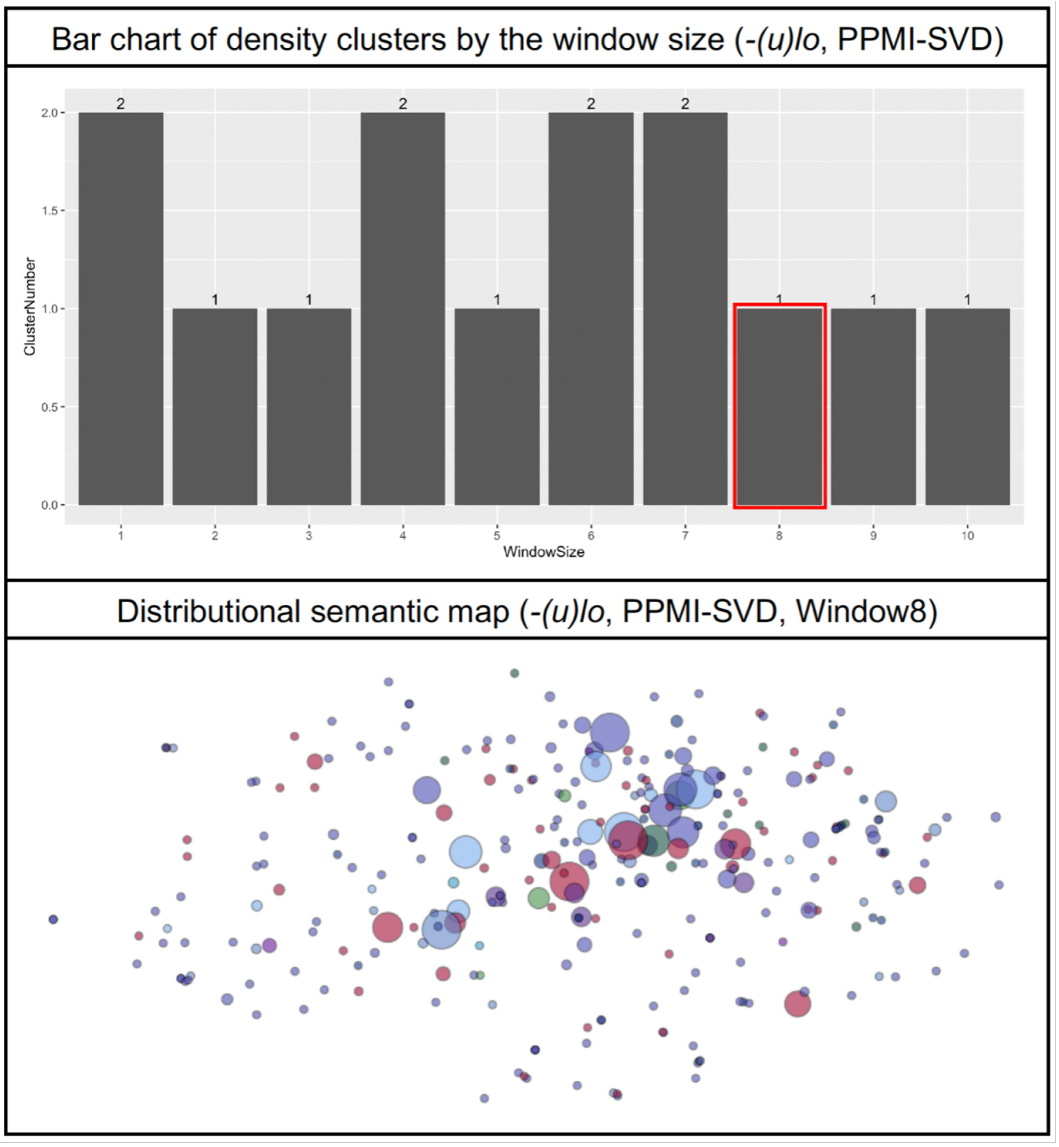

Figure 5.14: Bar chart of density cluster result and distributional semantic map for *-(u)lo* (PPMI-SVD). Red in graph = the size of window showing the highest accuracy.

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.

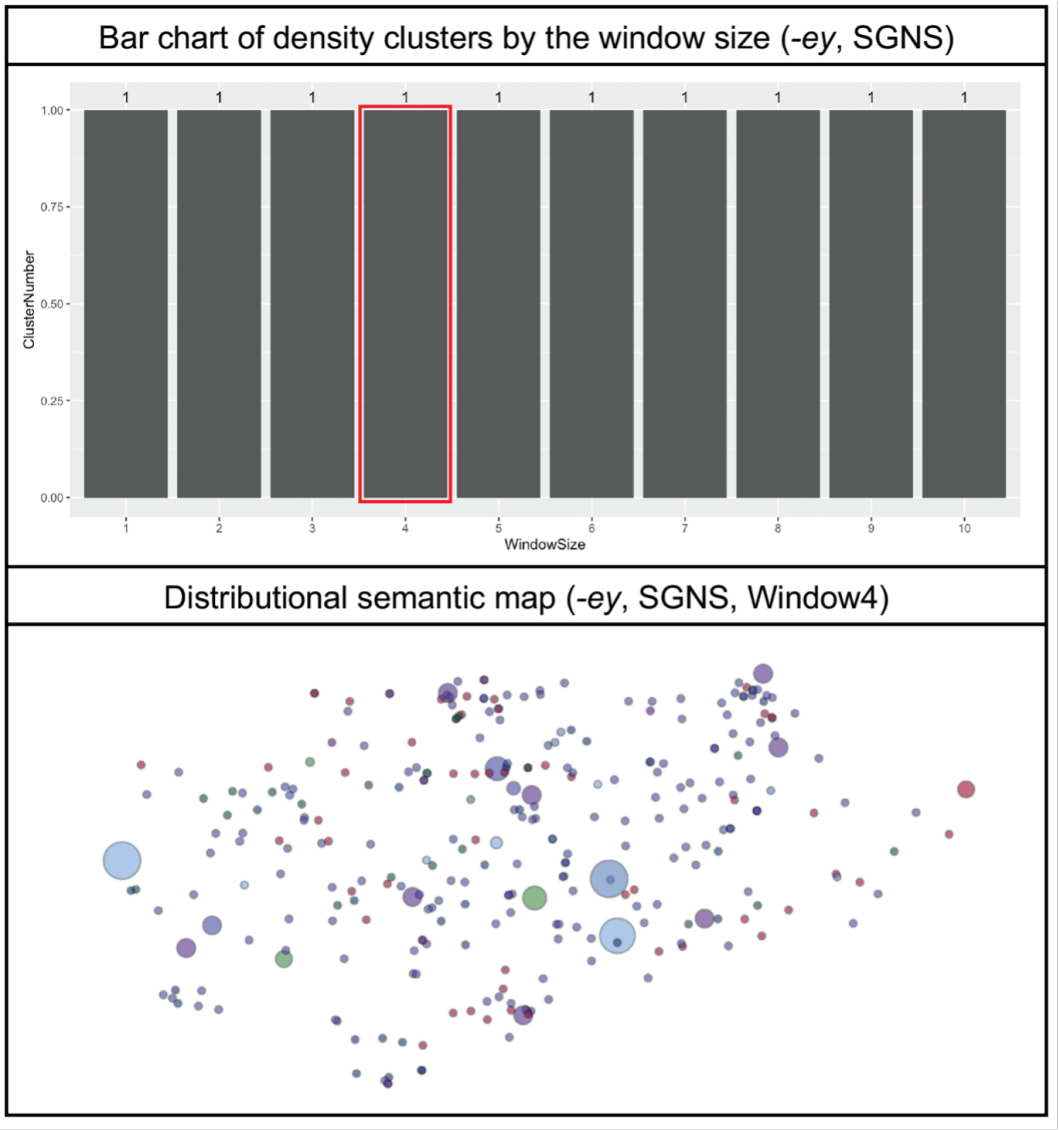

Figure 5.15: Bar chart of the density cluster result and distributional semantic map for *-ey* (SGNS). Red in graph = the size of window showing the highest accuracy.

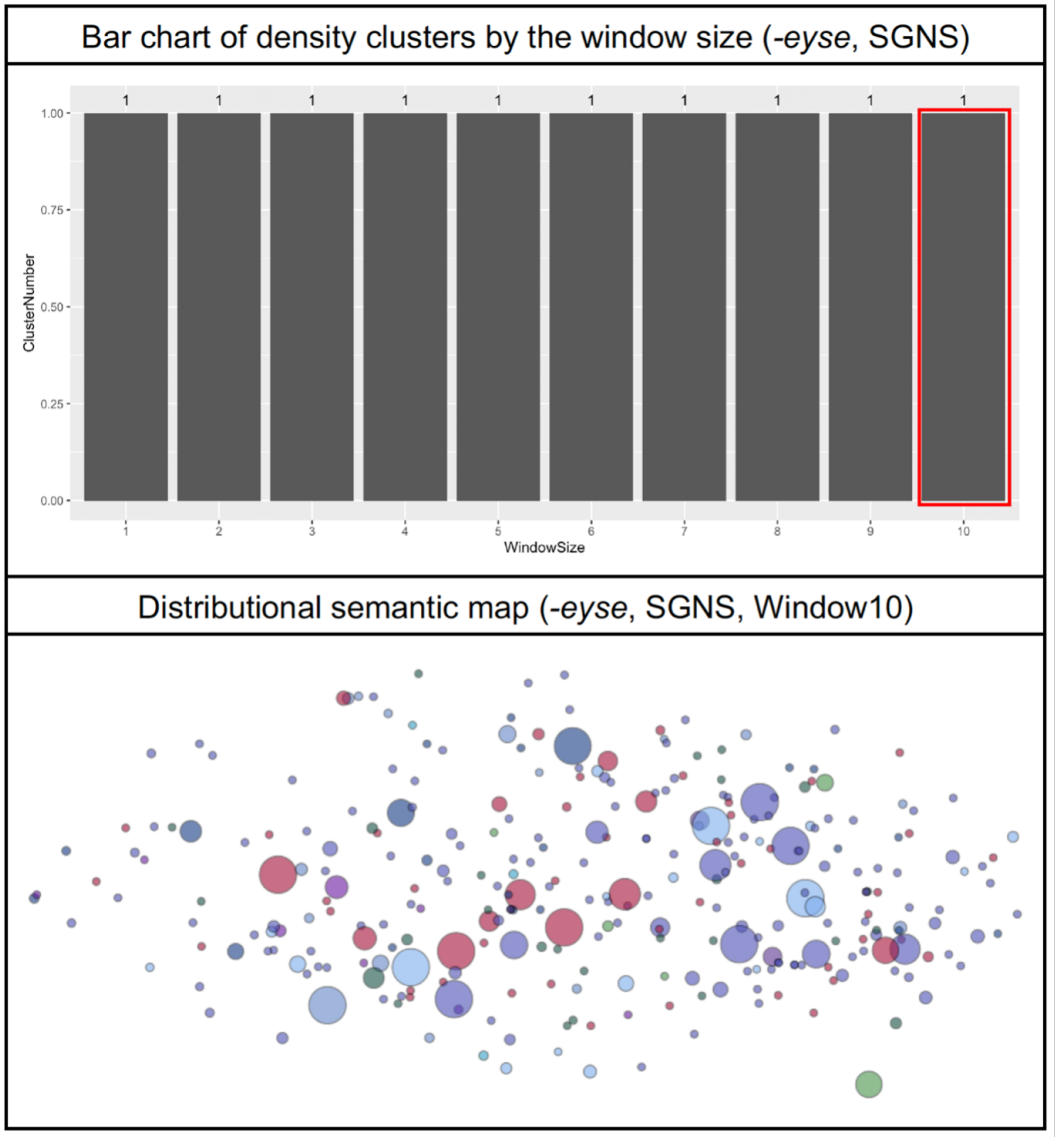

Figure 5.16: Bar chart of the density cluster result and distributional semantic map for *-eyse* (SGNS). Red in graph = the size of window showing the highest accuracy.

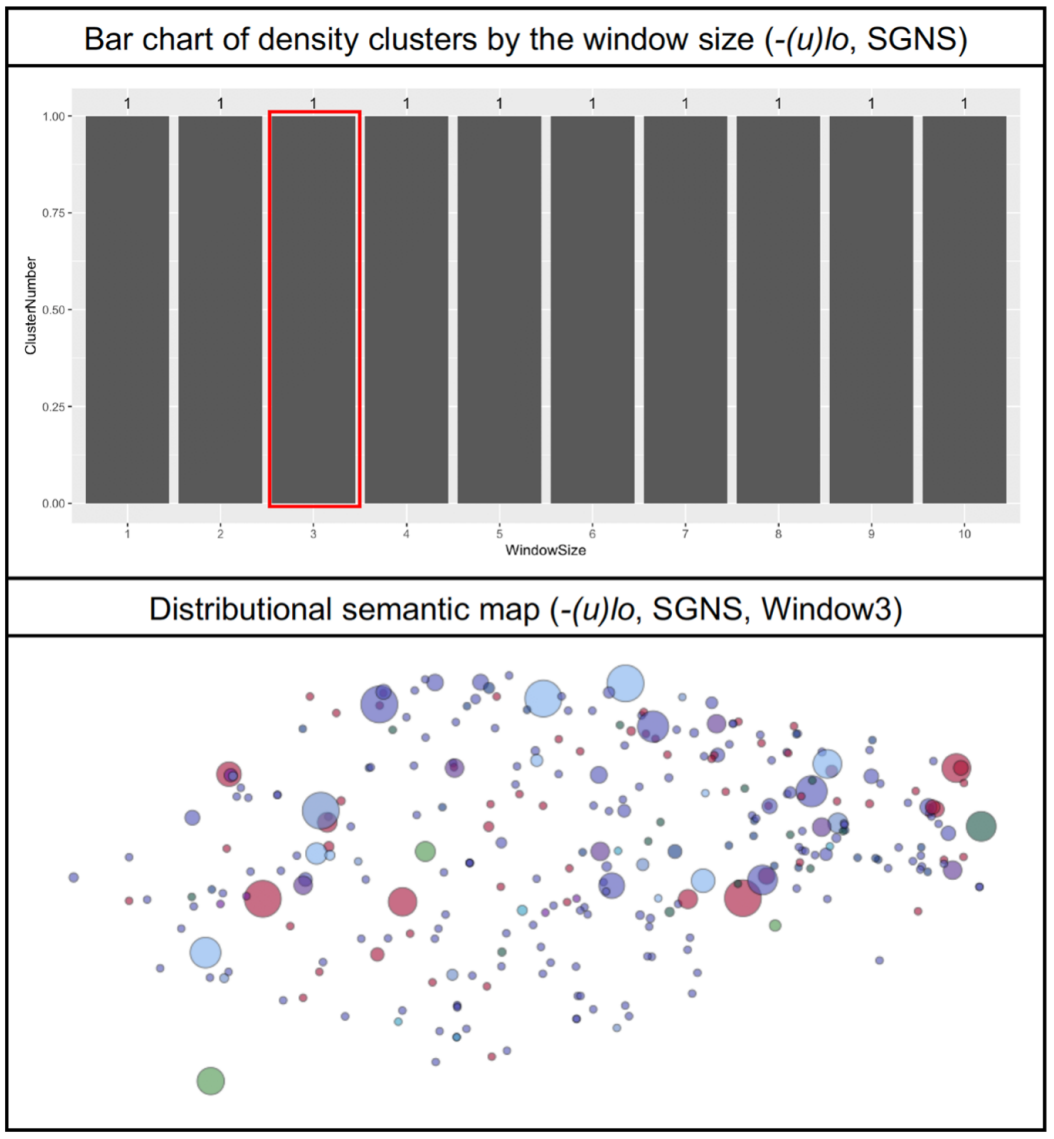

Figure 5.17: Bar chart of the density cluster result and distributional semantic map for *-(u)lo* (SGNS). Red in graph = the size of window showing the highest accuracy.

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.

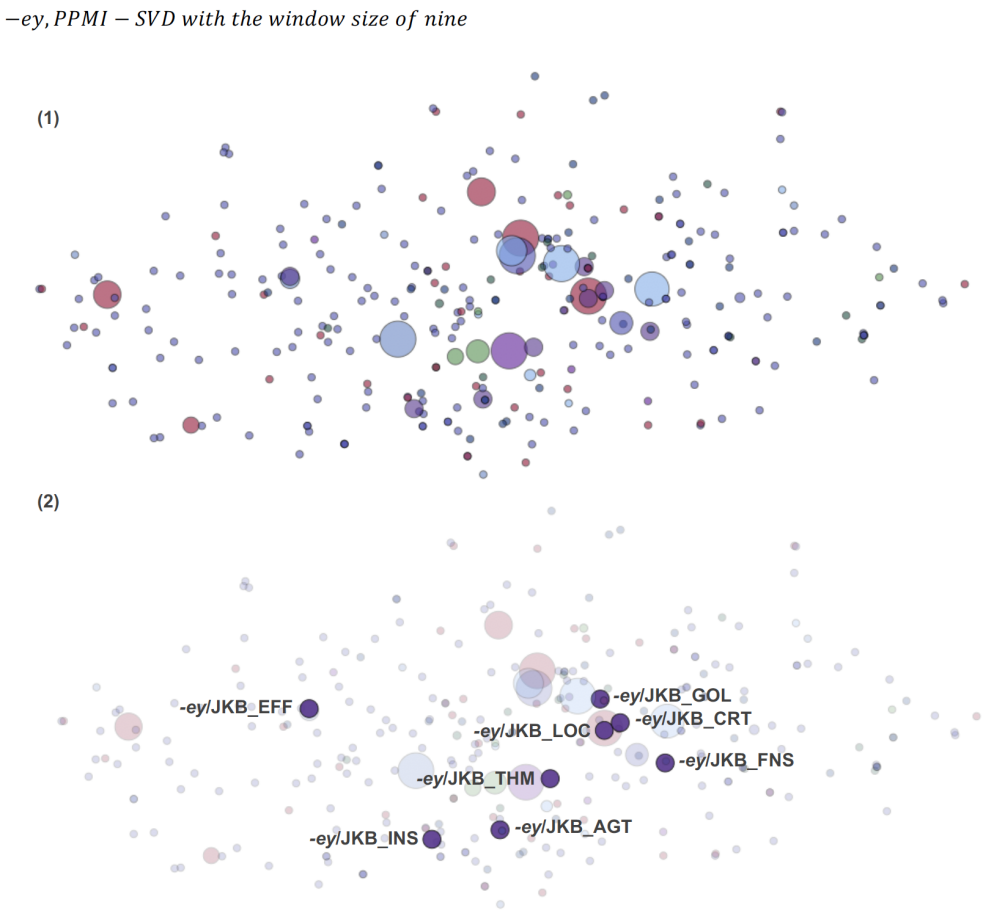

Figure 5.18: Distributional semantic map for *-ey* (PPMI-SVD; window size of nine)

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. 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.

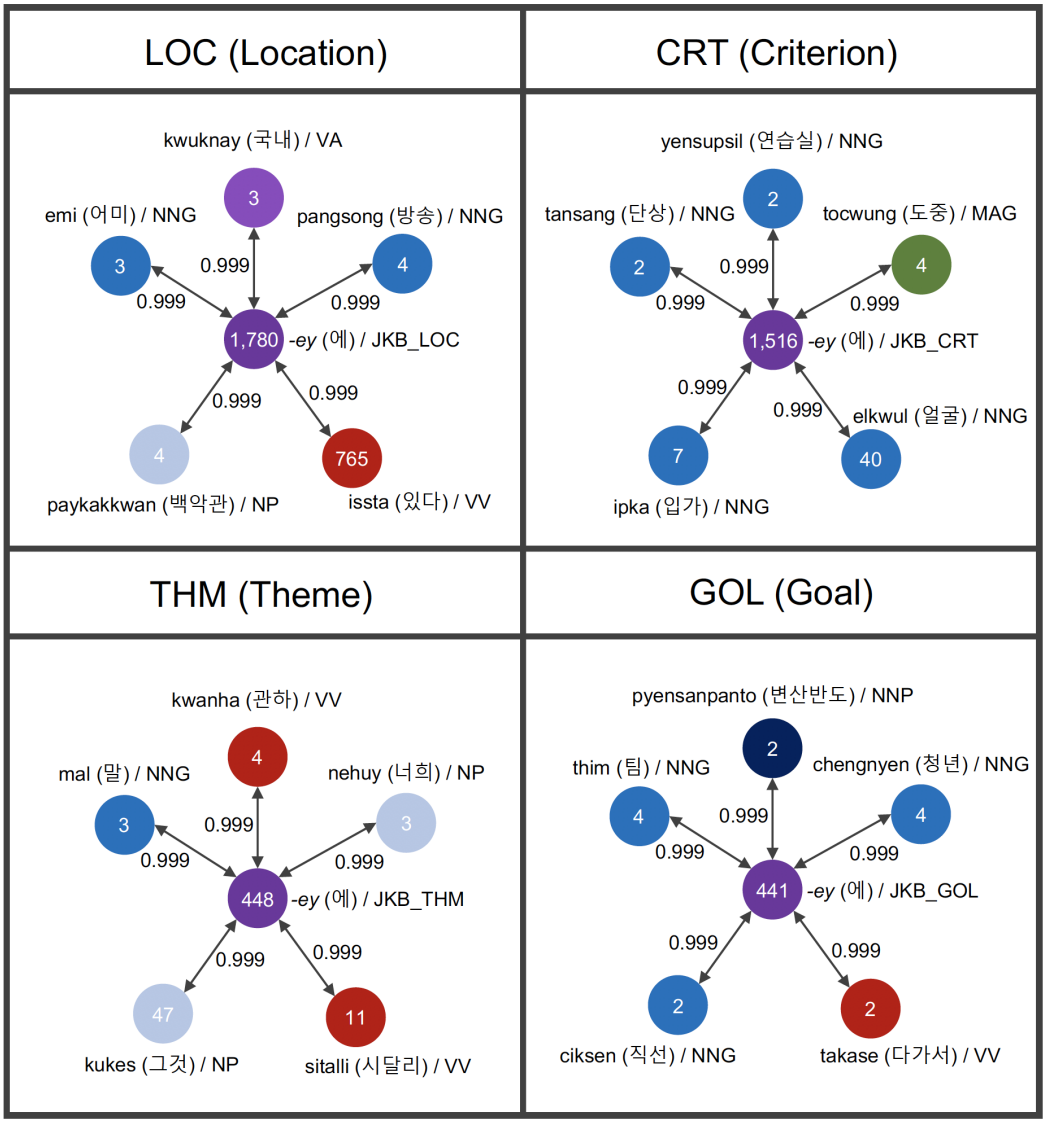

Figure 5.19: By-function co-occurring words for *-ey*: LOC, CRT, THM, and GOL

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.

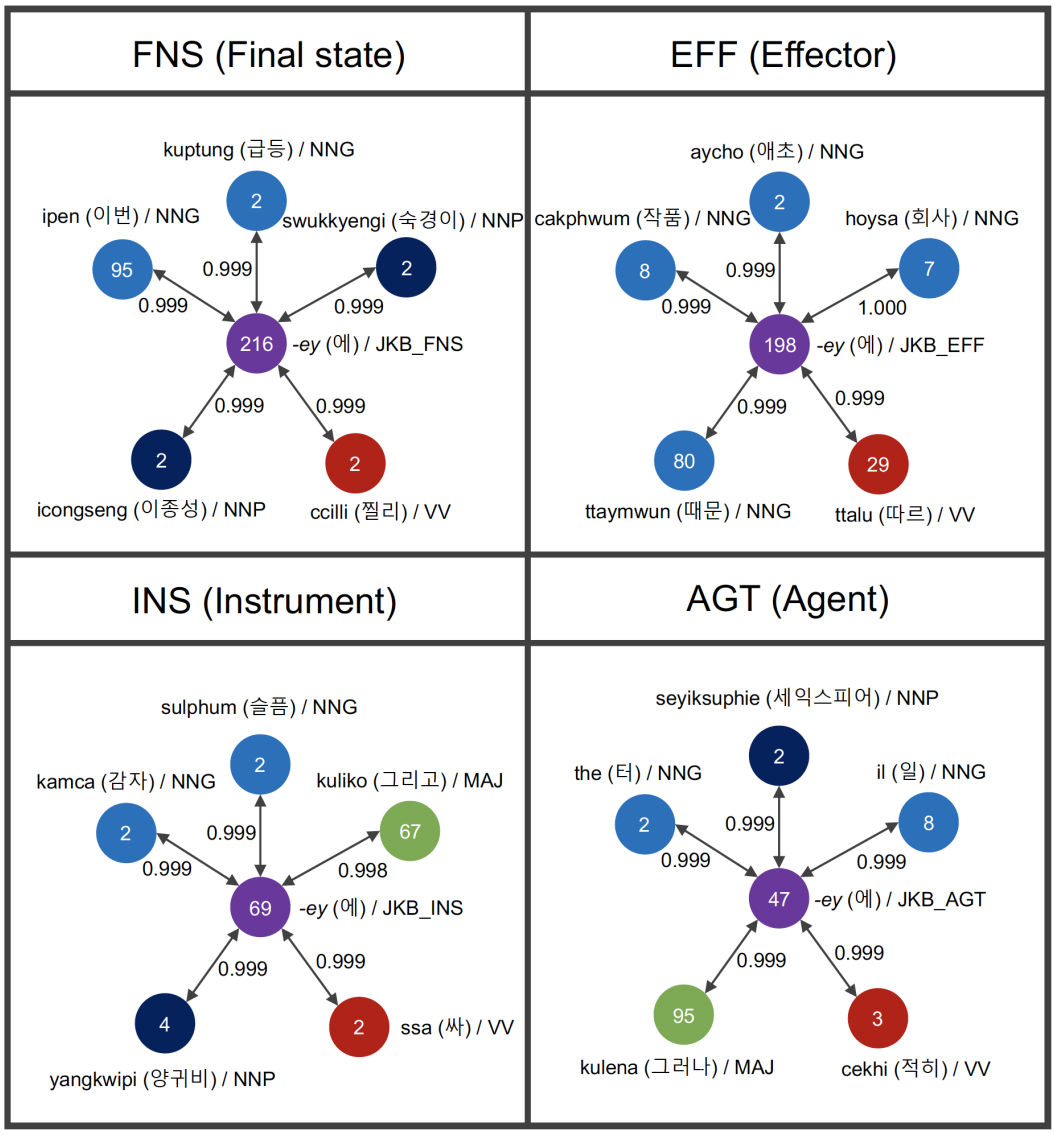

Figure 5.20: By-function co-occurring words for *-ey*: FNS, EFF, INS, and AGT

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.

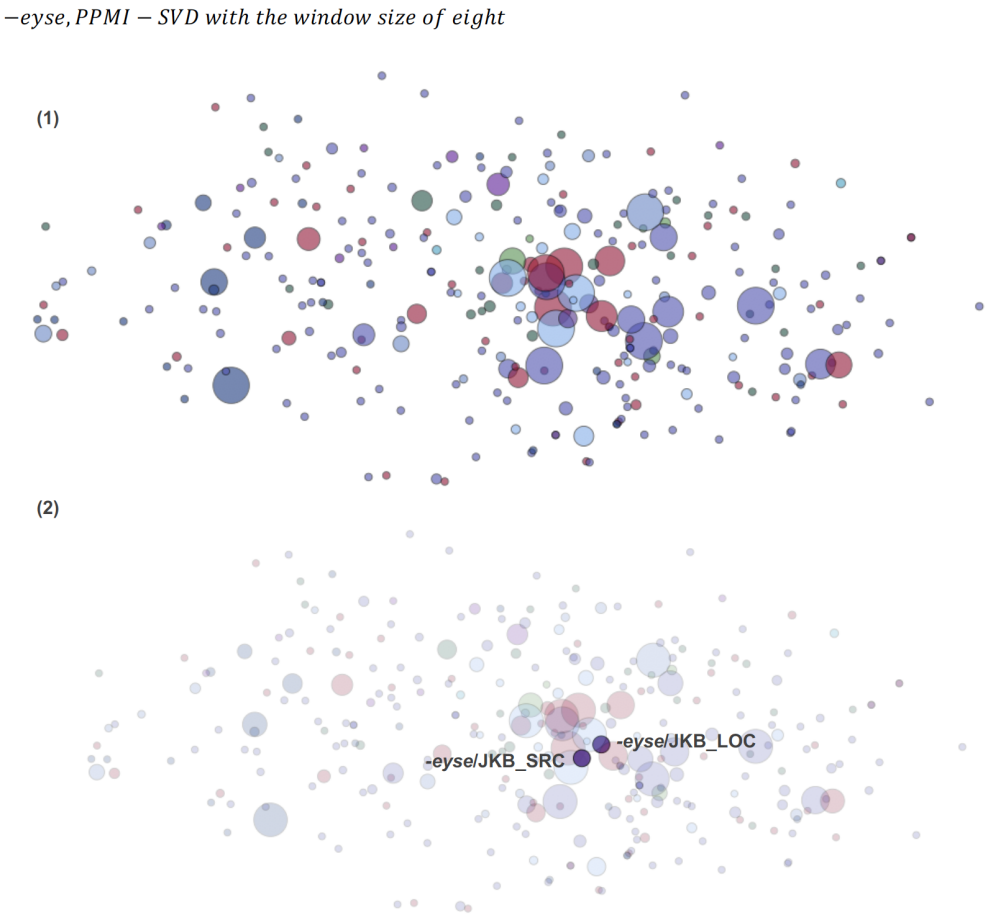

Figure 5.21: Distributional semantic map for *-eyse* (PPMI-SVD; window size of eight)

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*.

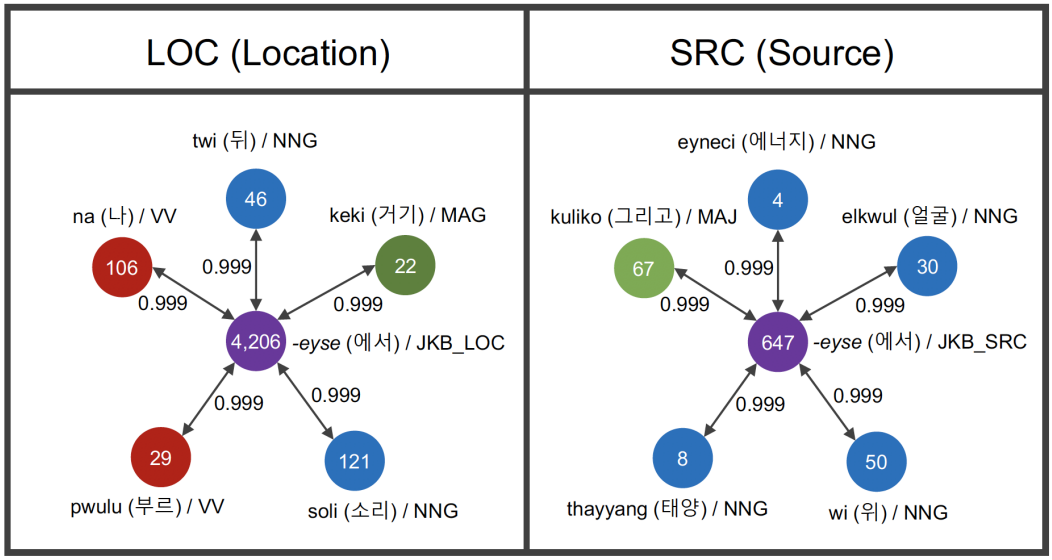

Figure 5.22: By-function co-occurring words for *-eyse*: LOC and SRC

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.

$-(u)lo, PPMI - SVD\ with\ the\ window\ size\ of\ nine$

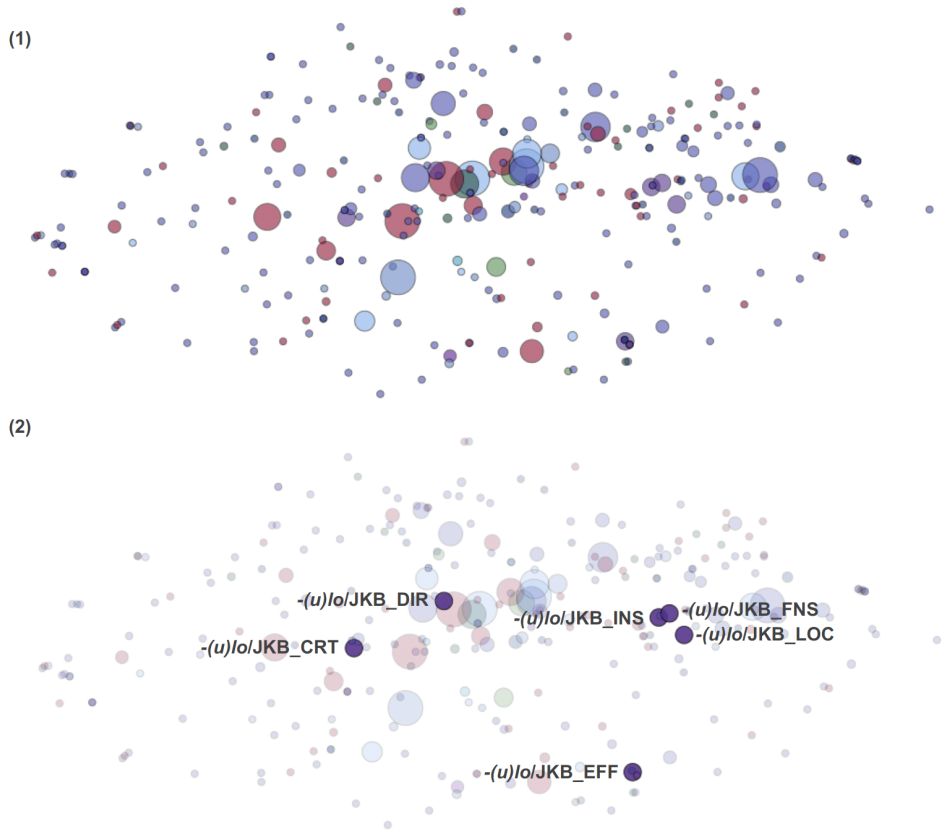

Figure 5.23: Distributional semantic map for *-(u)lo* (PPMI-SVD; window size of nine)

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 fre-

quently also.

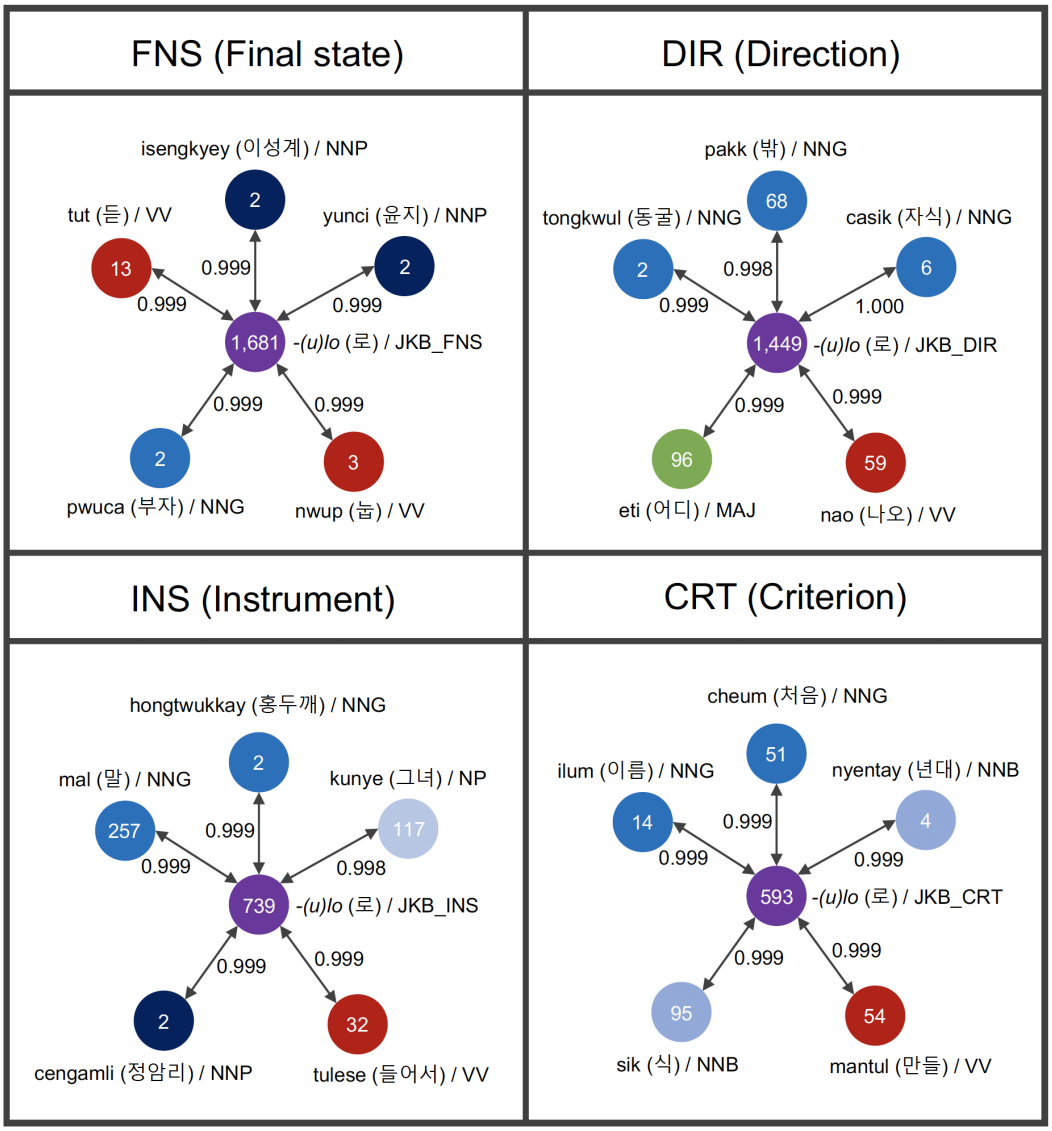

Figure 5.24: By-function co-occurring words for *-(u)lo*: FNS, EFF, INS, and CRT

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.

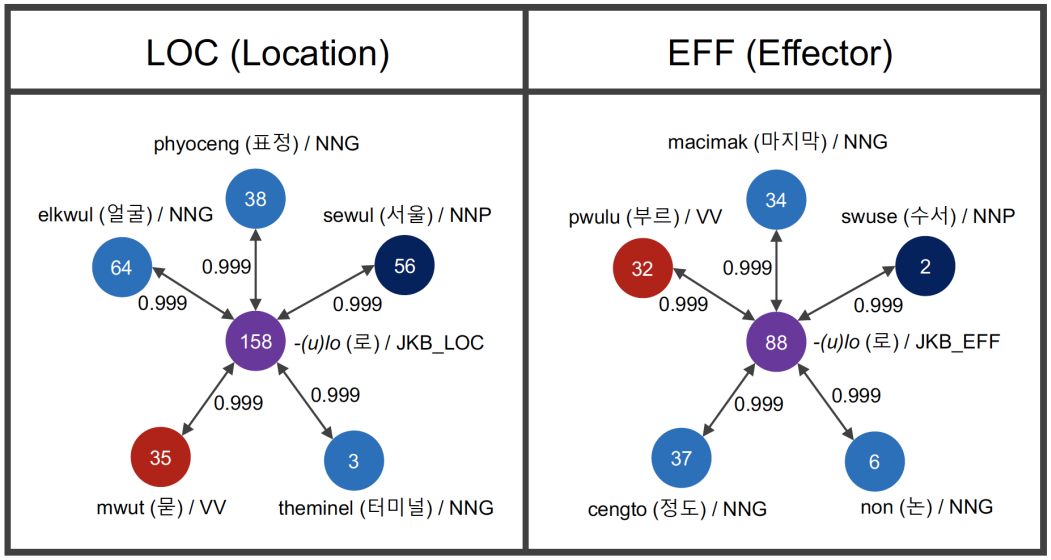

Figure 5.25: By-function co-occurring words for *-(u)lo*: LOC and EFF

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. Fur-

thermore, 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.

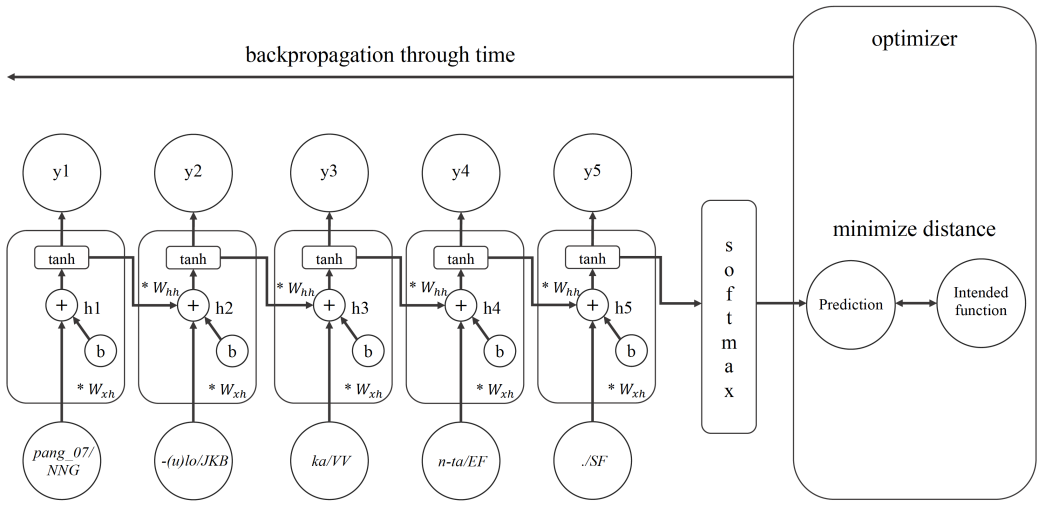

Figure 6.1: Workflow of the RNN model adapted from Heo (2018)

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).

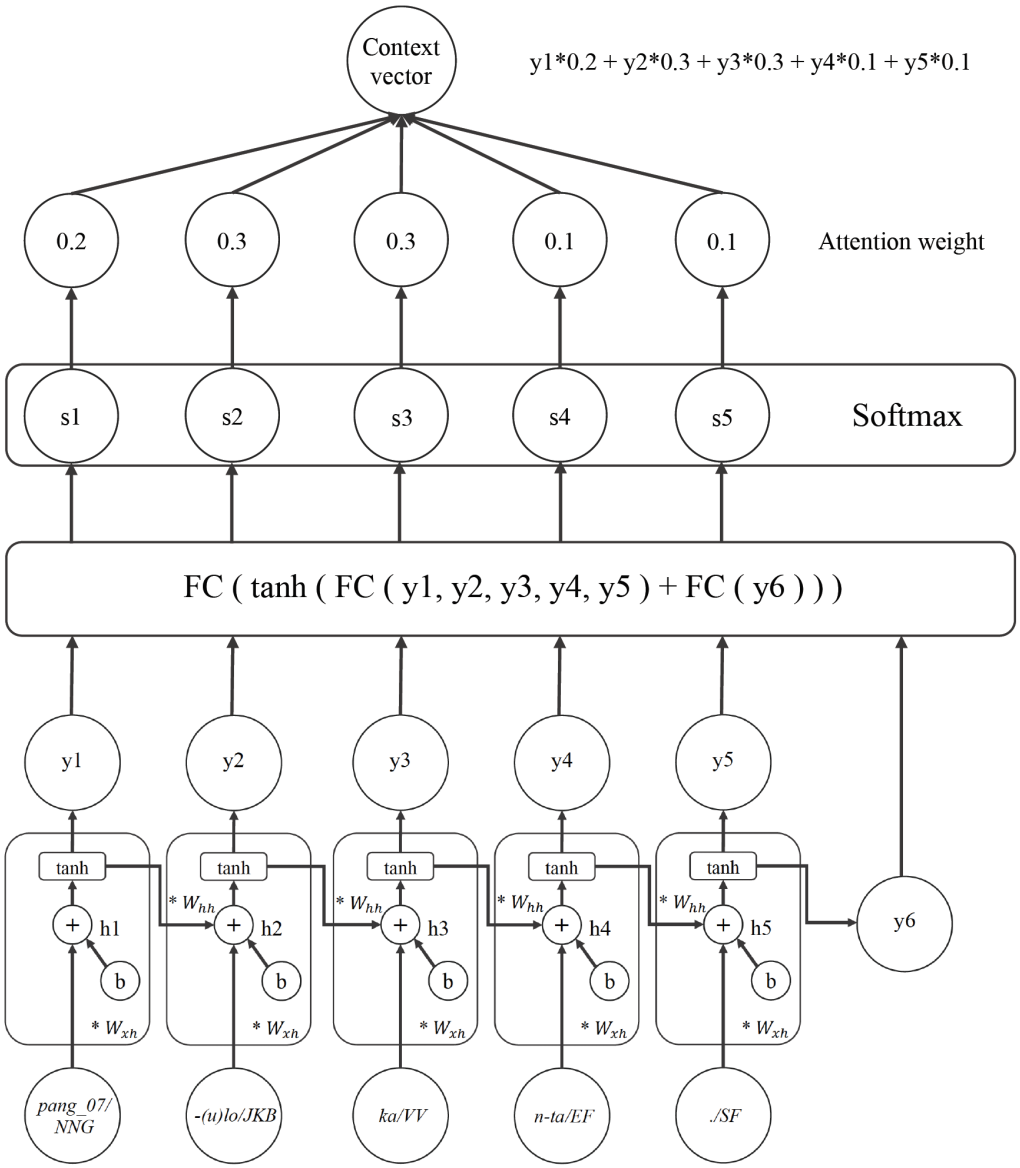

Figure 6.2: Workflow of the attention model adapted from Heo (2018)

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).

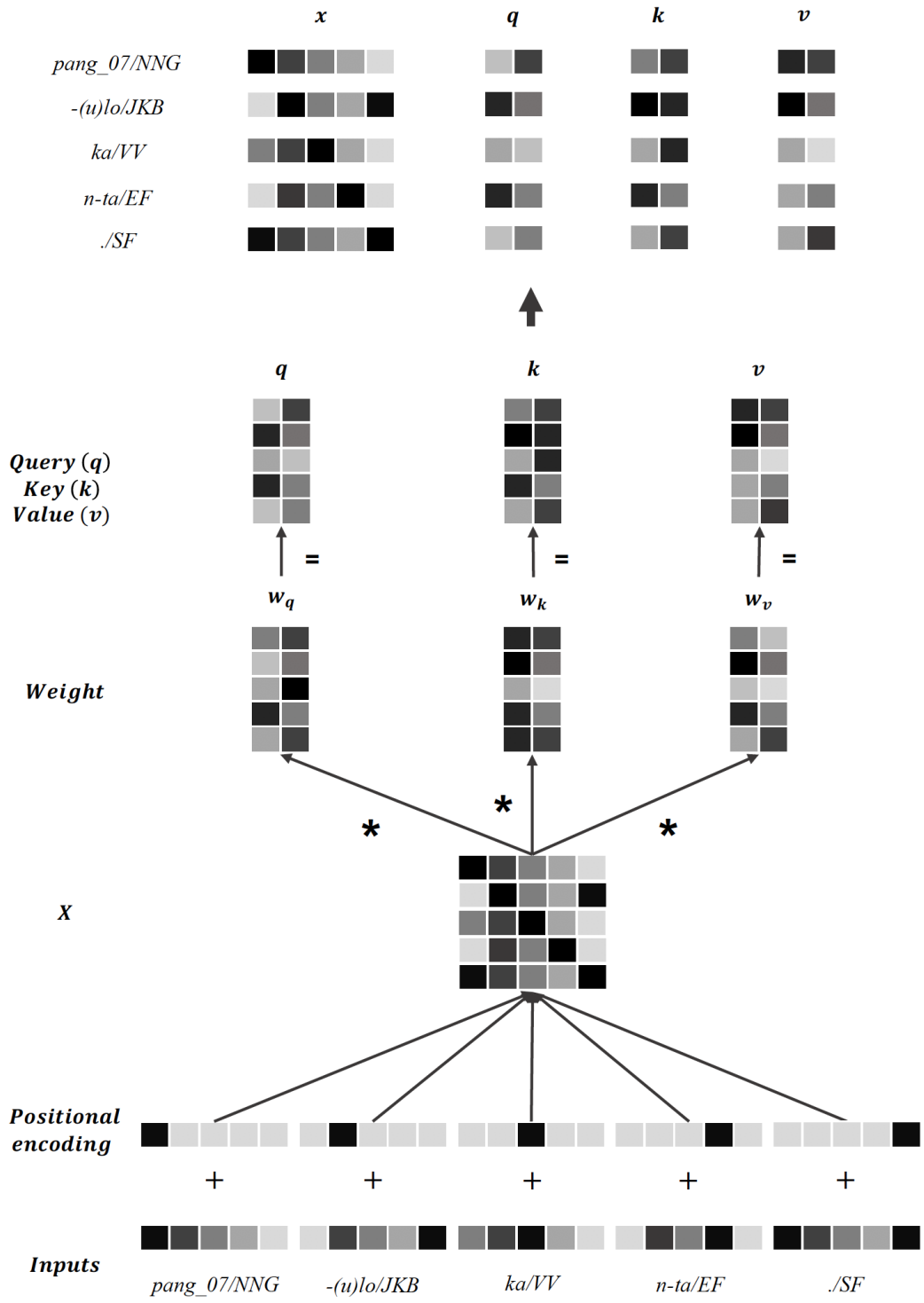

Figure 6.3: Workflow of the self-attention model

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.

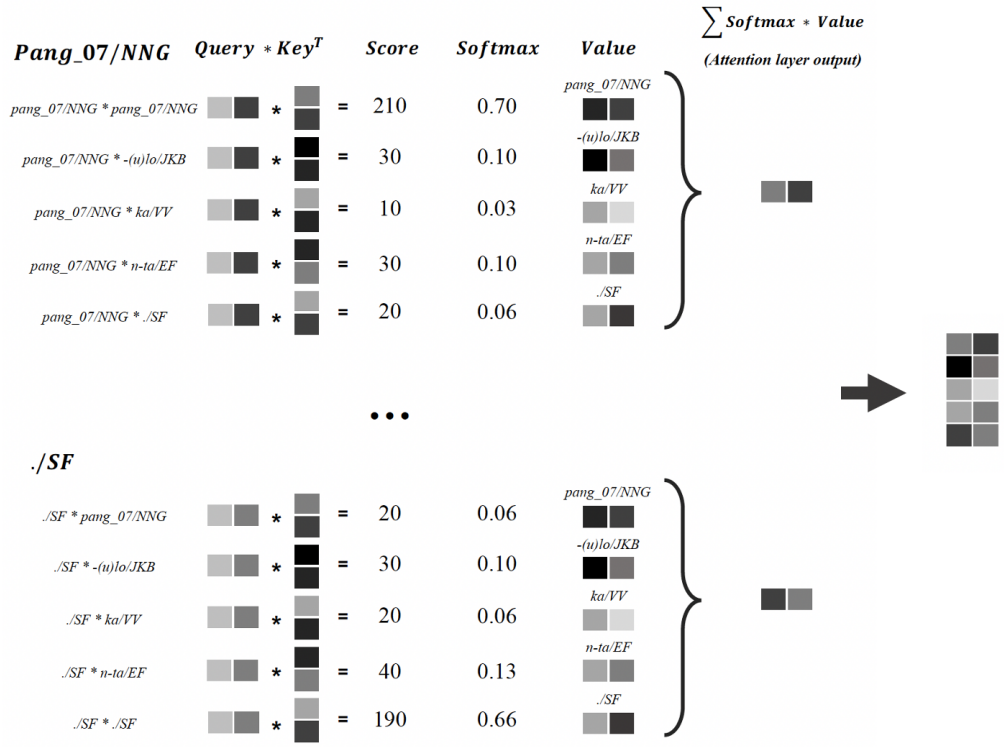

Figure 6.4: Calculation process of the self-attention model

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.

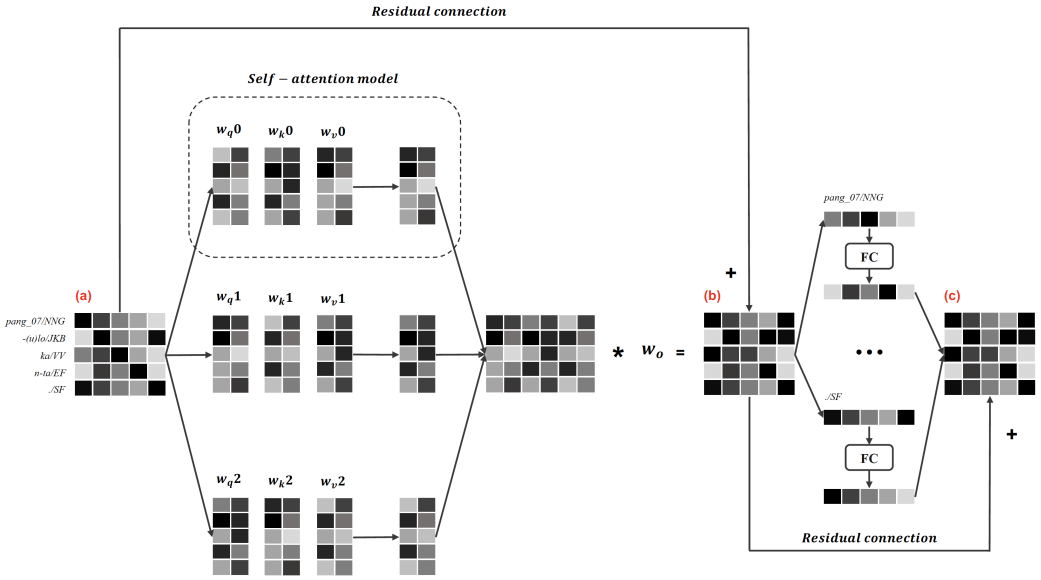

Figure 6.5: Workflow of the multi-head self-attention model (an encoder layer)

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 em-

phasis 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.

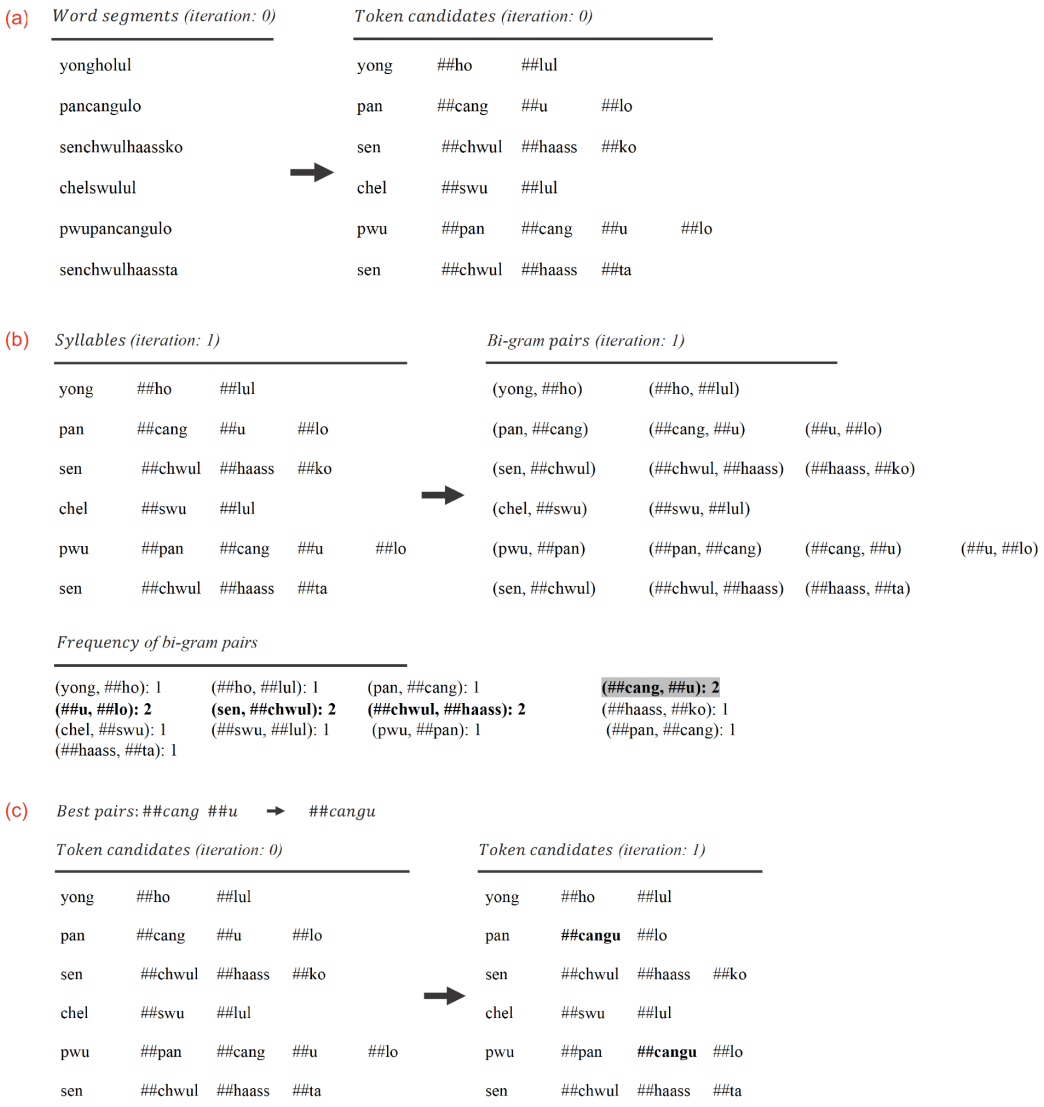

Figure 6.7: Workflow of the WordPiece tokenization

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-IND-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.

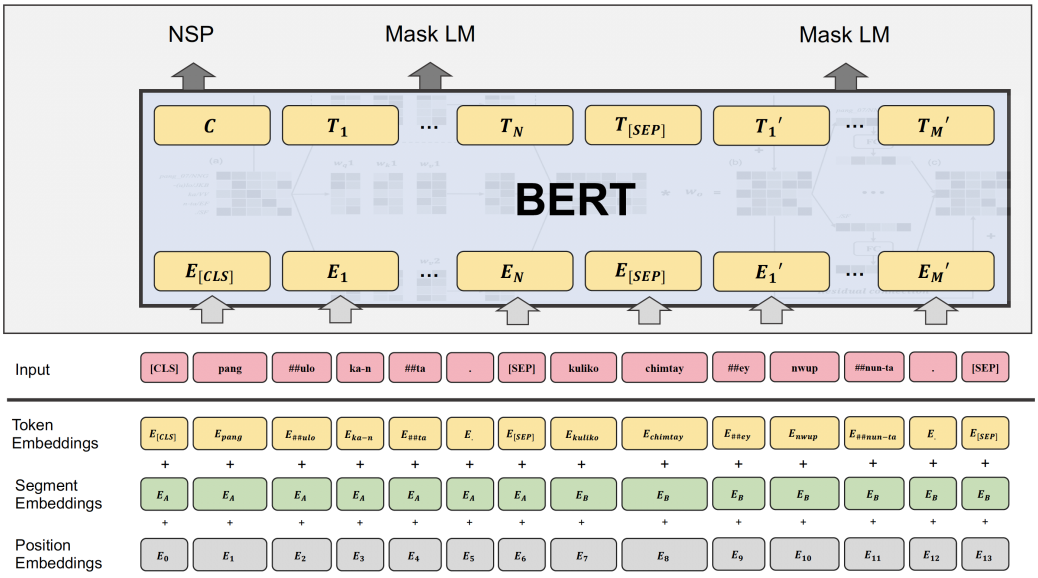

Figure 6.8: Three embedding types for BERT adapted from Devlin et al. (2018)

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.

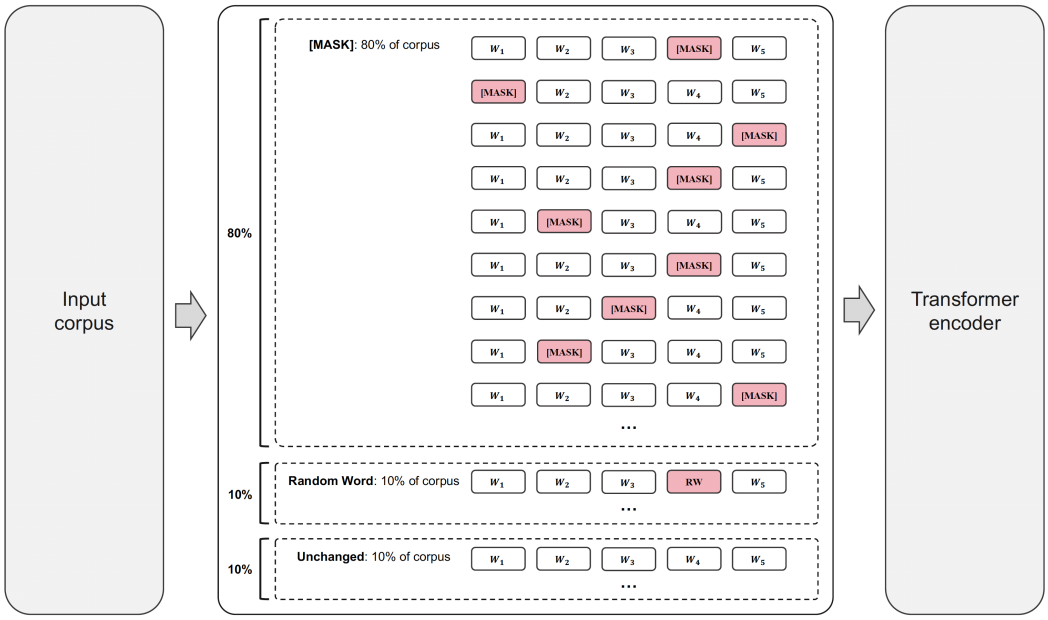

Figure 6.9: Workflow of the Masked Language Model (MLM)

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.

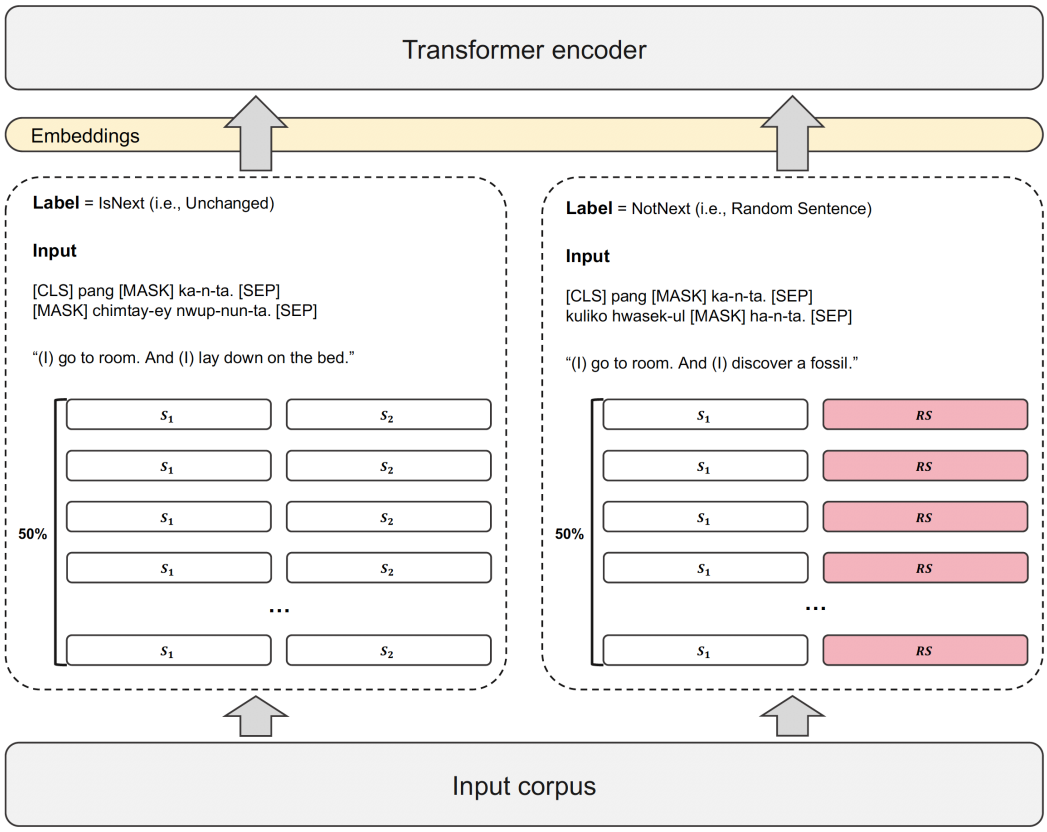

Figure 6.10: Workflow of the Next Sentence Prediction (NSP)

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.

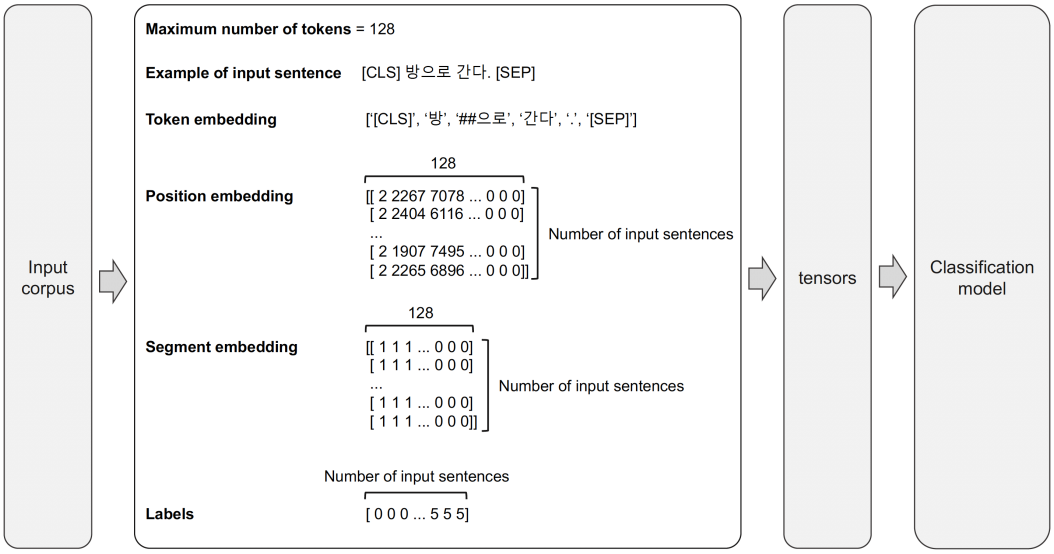

Figure 7.2: Input embeddings for the BERT classification model

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 obtained the two-dimension distribution data in each epoch as the outcome of

[1]The entire code for BERT training that I developed is available at: BERT

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.

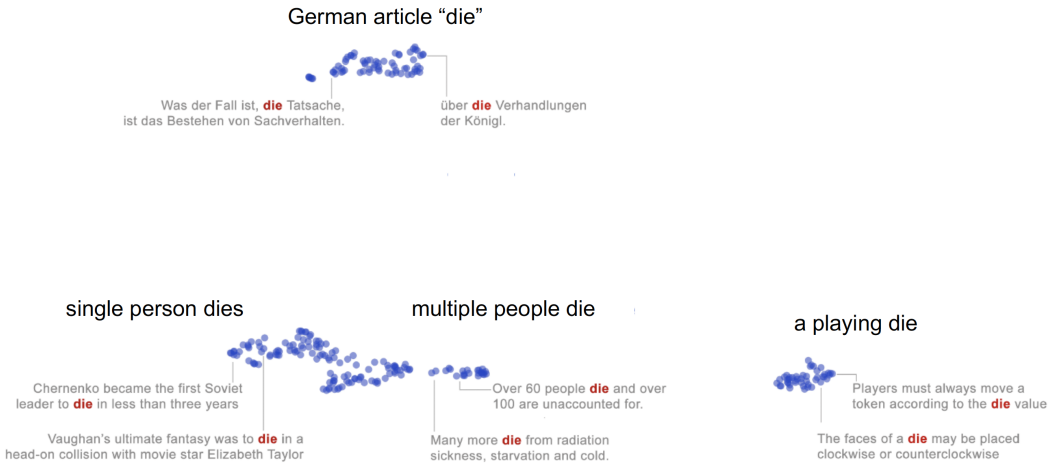

Figure 7.3: The visualization of the sentence-level embeddings for the word *'die'* in different contexts (adapted from Coenen et al. (2019))

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).

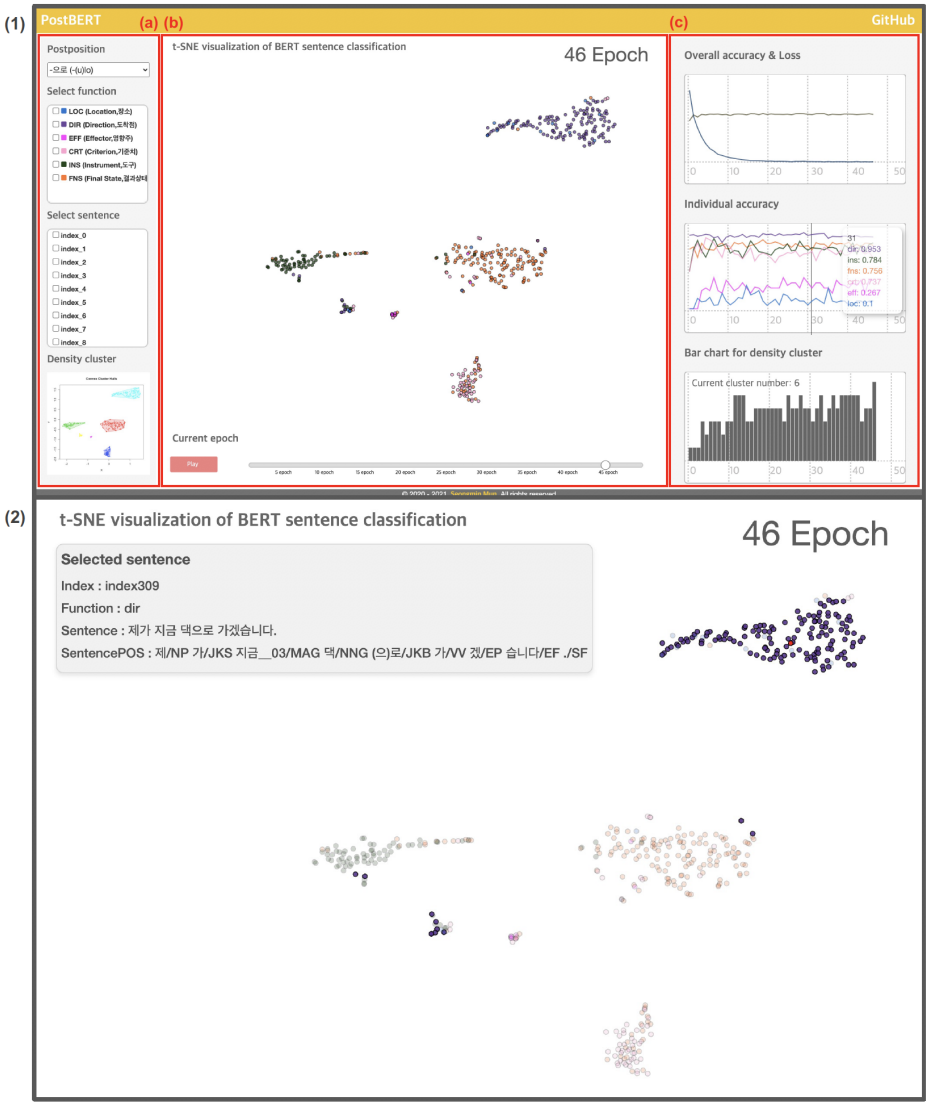

Figure 7.4: The visualization system: the overall interface (1) and the main view (2)

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 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, I 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.

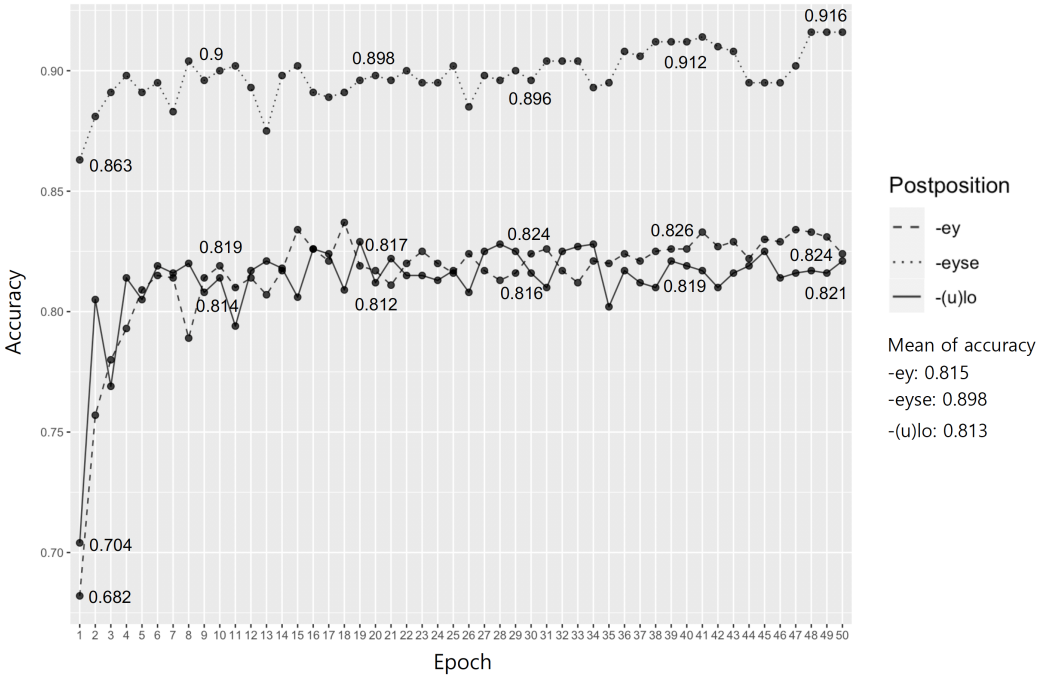

Figure 8.1: Classification accuracy by epoch and by postposition

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).

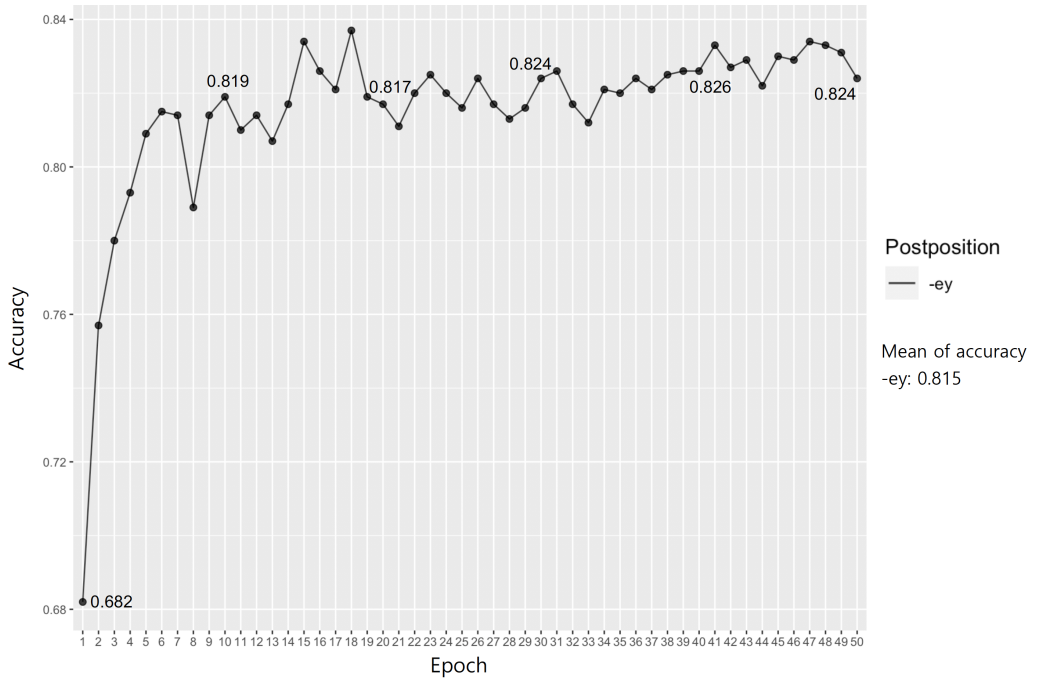

Figure 8.2: By-epoch accuracy for the BERT model: *-ey*

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).

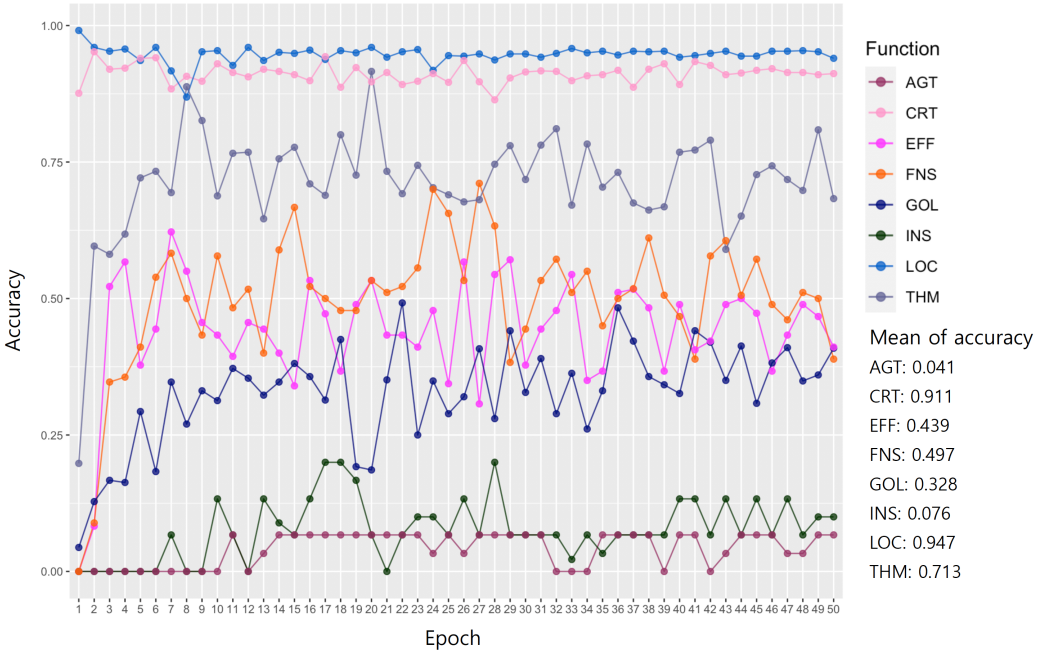

Figure 8.3: By-function accuracy curve for the BERT model: *-ey*

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.

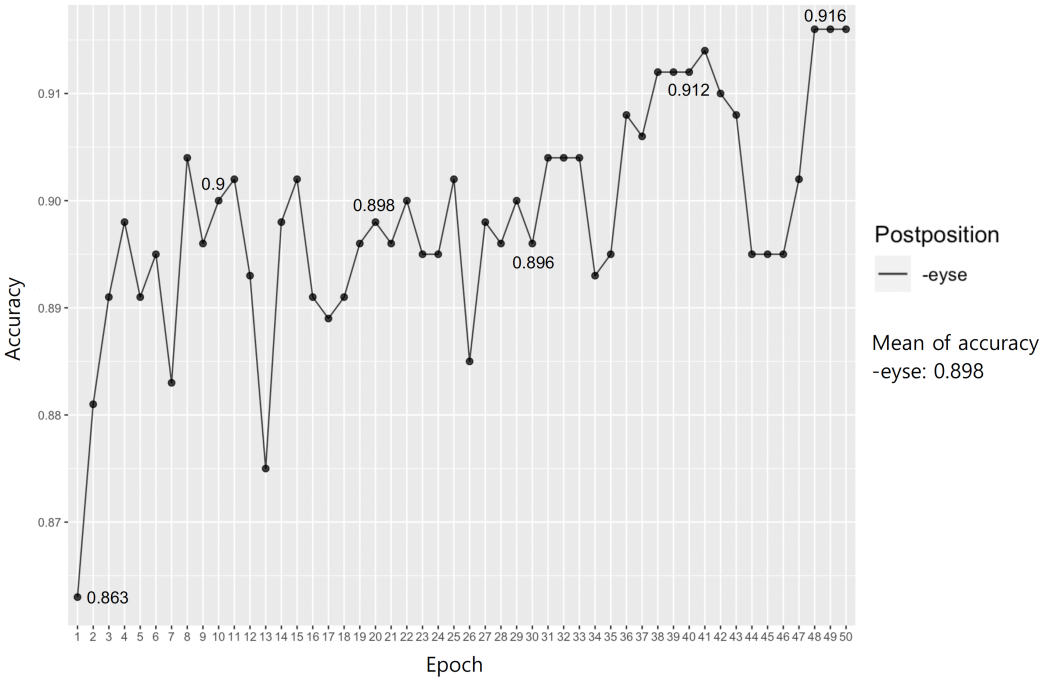

Figure 8.4: By-epoch accuracy for the BERT model: *-eyse*

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.

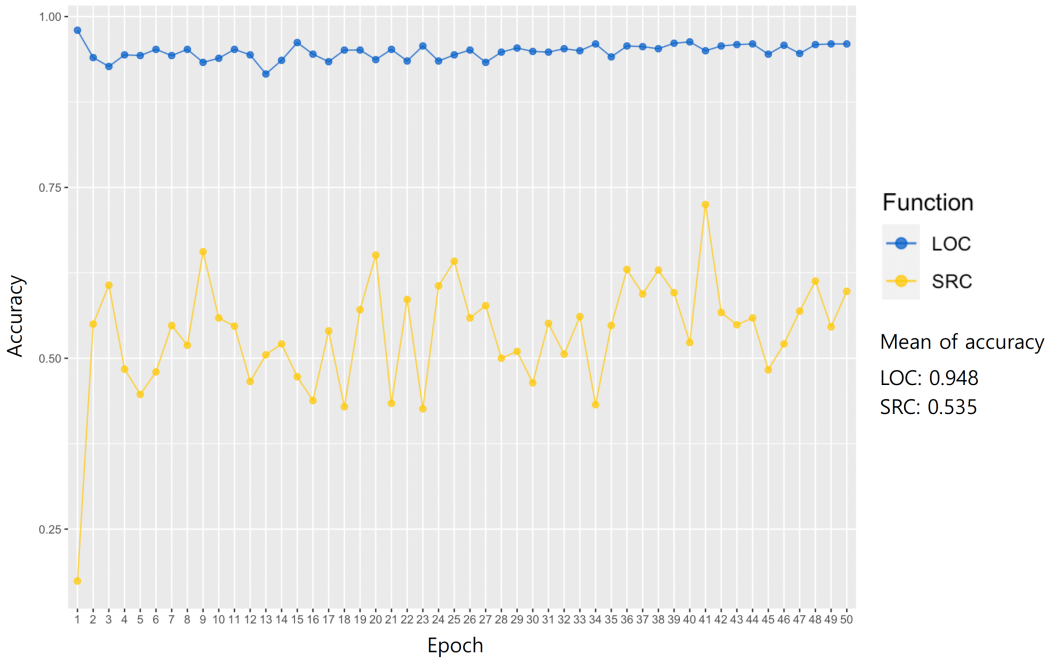

Figure 8.5: By-function accuracy curve for the BERT model: *-eyse*

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).

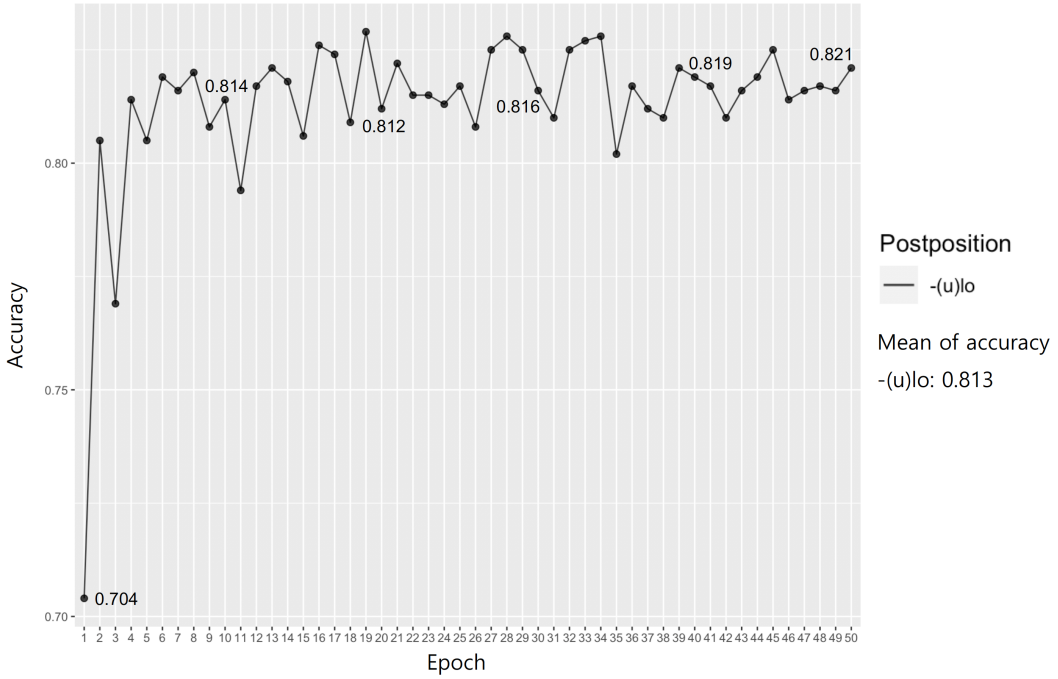

Figure 8.6: By-epoch accuracy for the BERT model: *-(u)lo*

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.

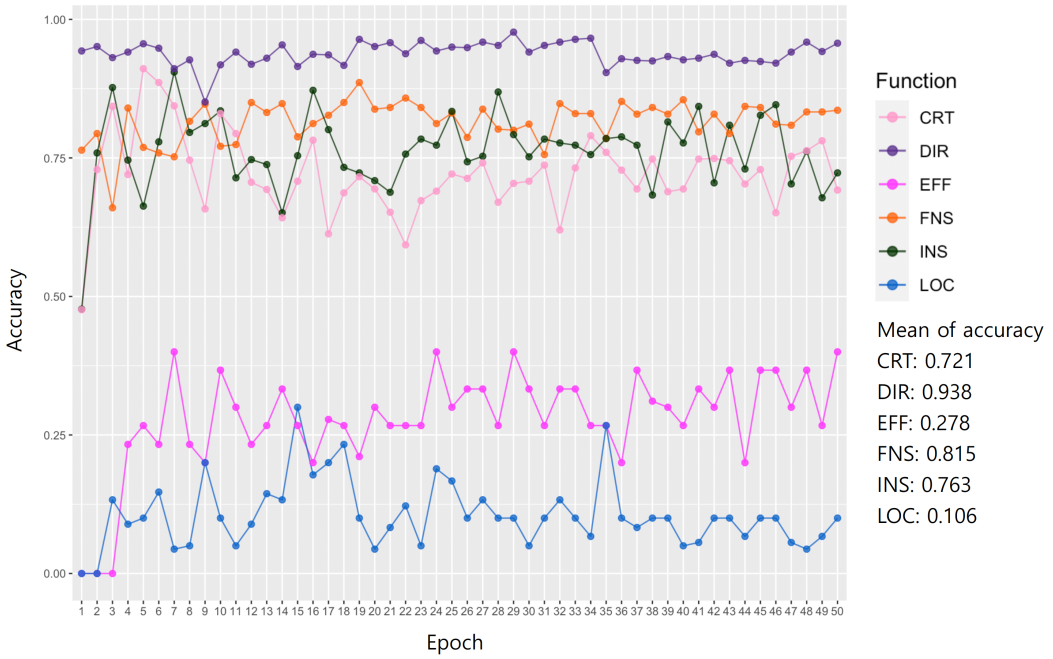

Figure 8.7: By-function accuracy curve for the BERT model: *-(u)lo*

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 by 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]The 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).

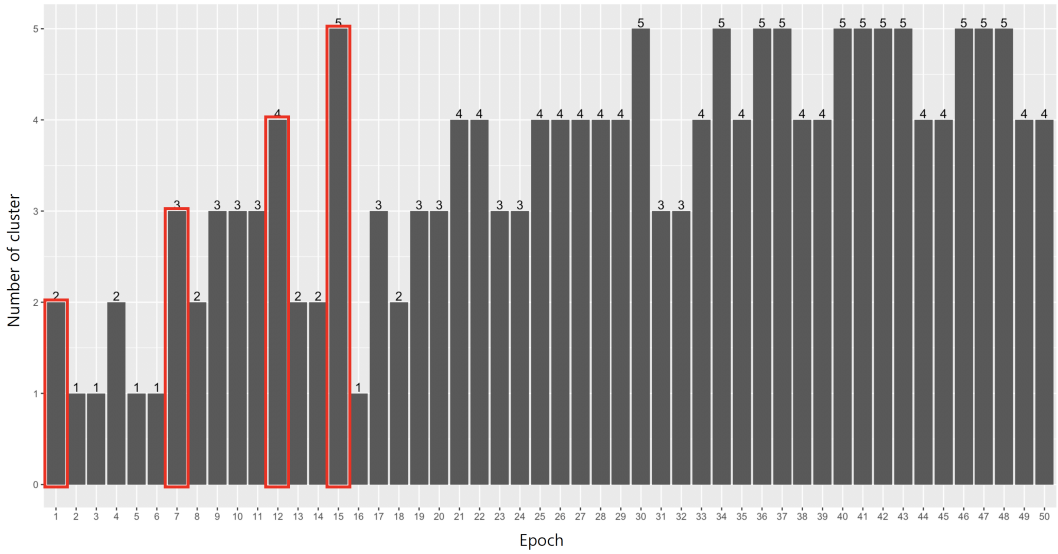

Figure 8.8: Number of density clusters in each epoch: *-ey*

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.

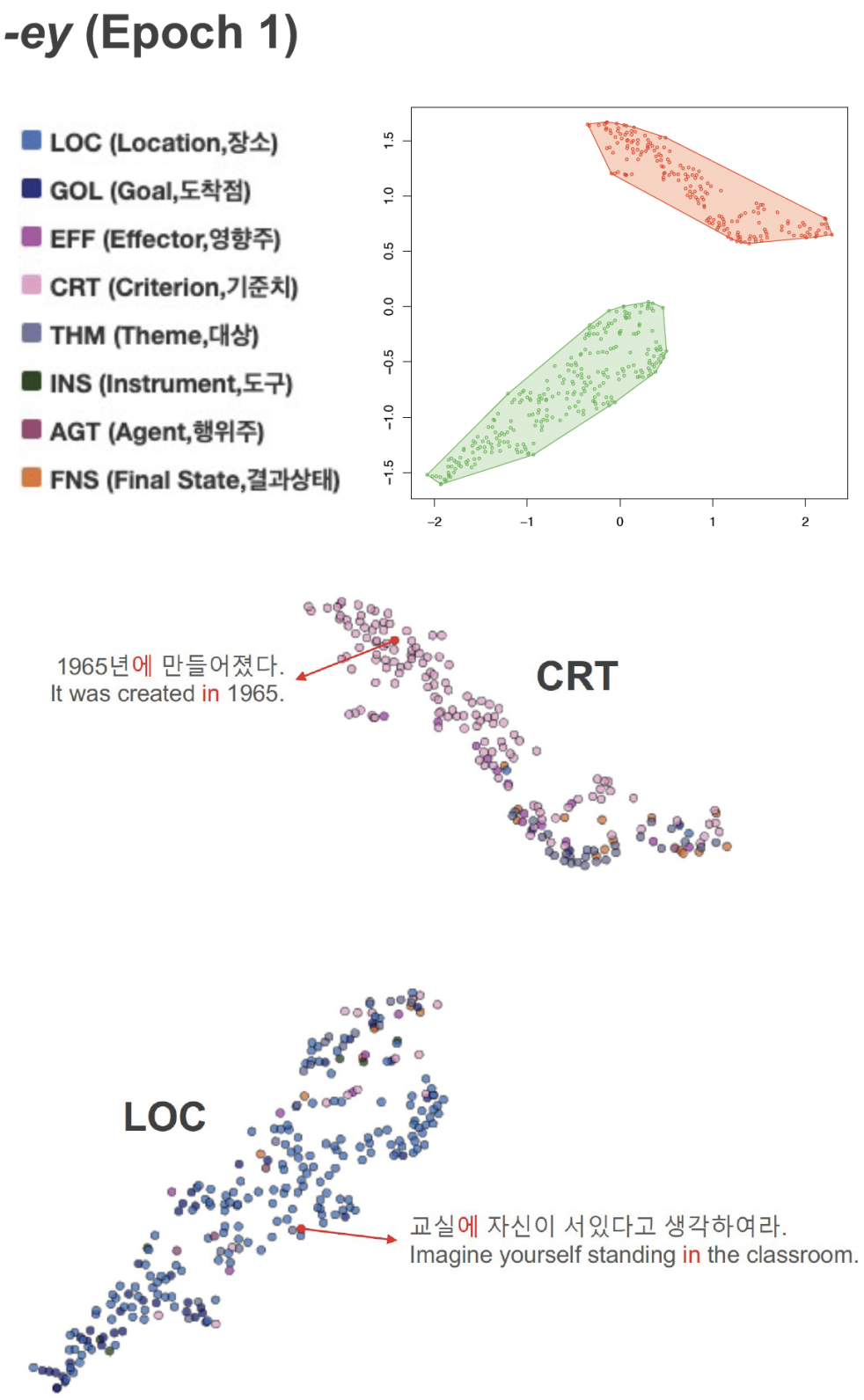

Figure 8.9: The distributional map for *-ey* in epoch one

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.

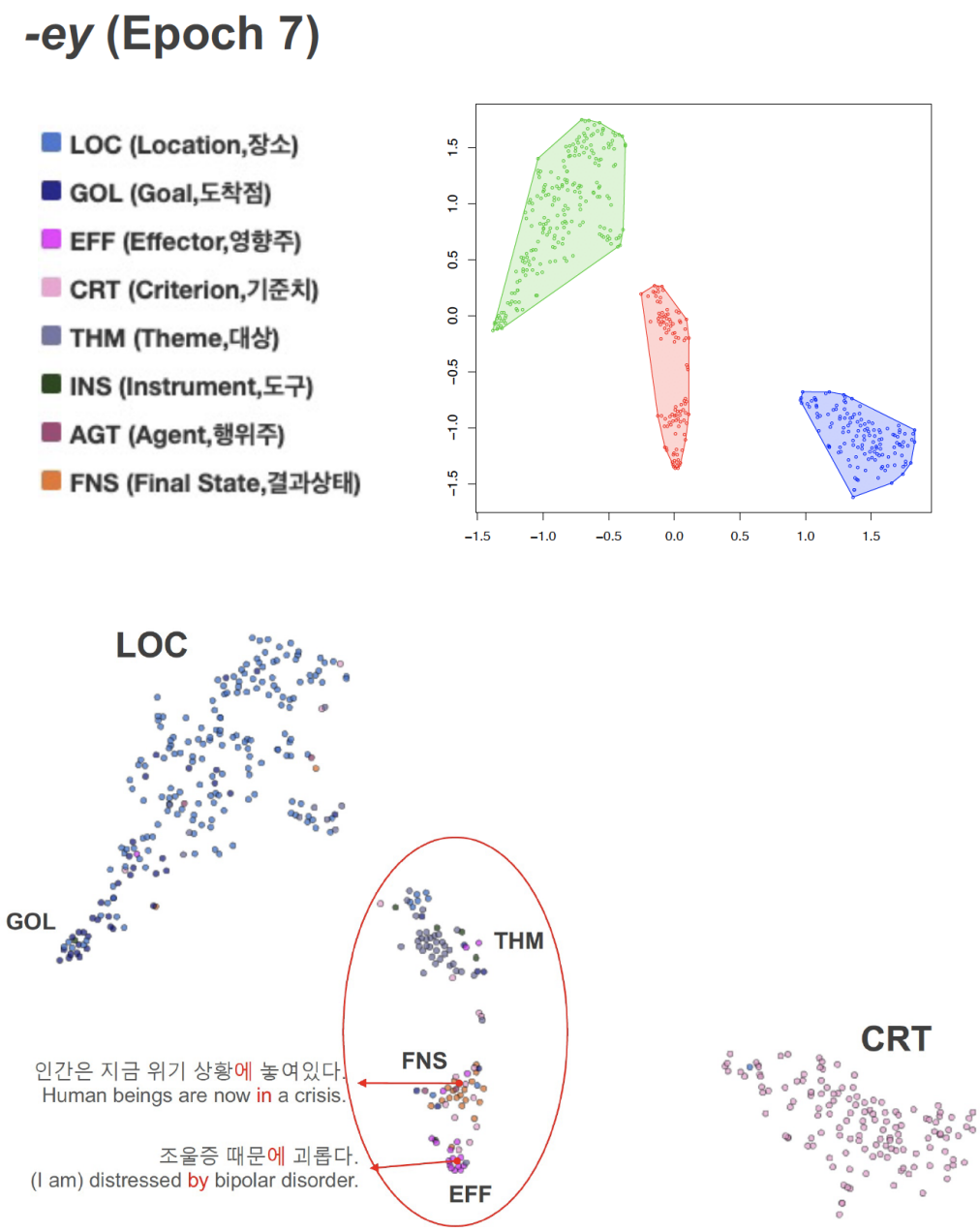

Figure 8.10: The distributional map for *-ey* in epoch seven

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.

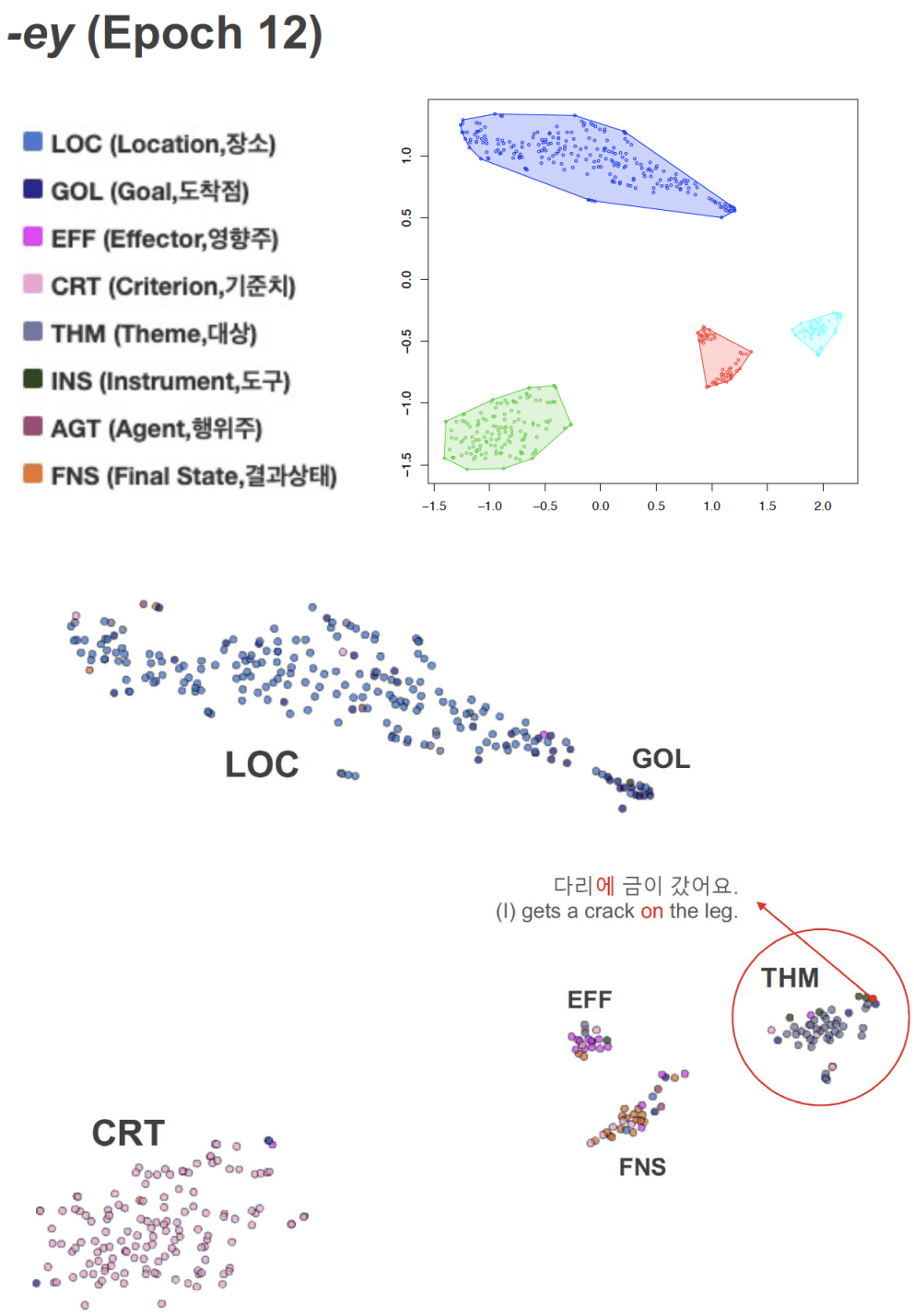

Figure 8.11: The distributional map for *-ey* in epoch 12

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.

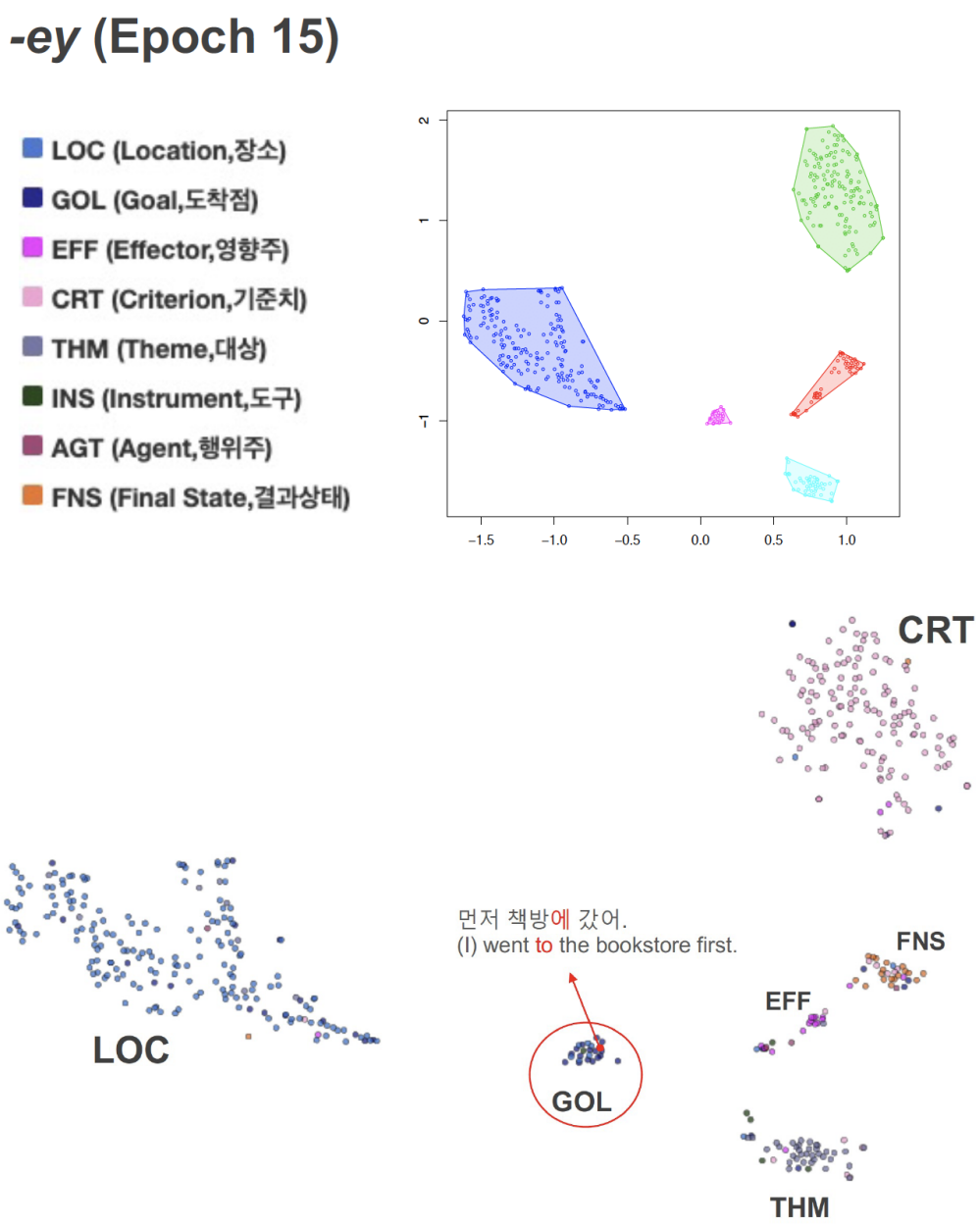

Figure 8.12: The distributional map for *-ey* in epoch 15

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).

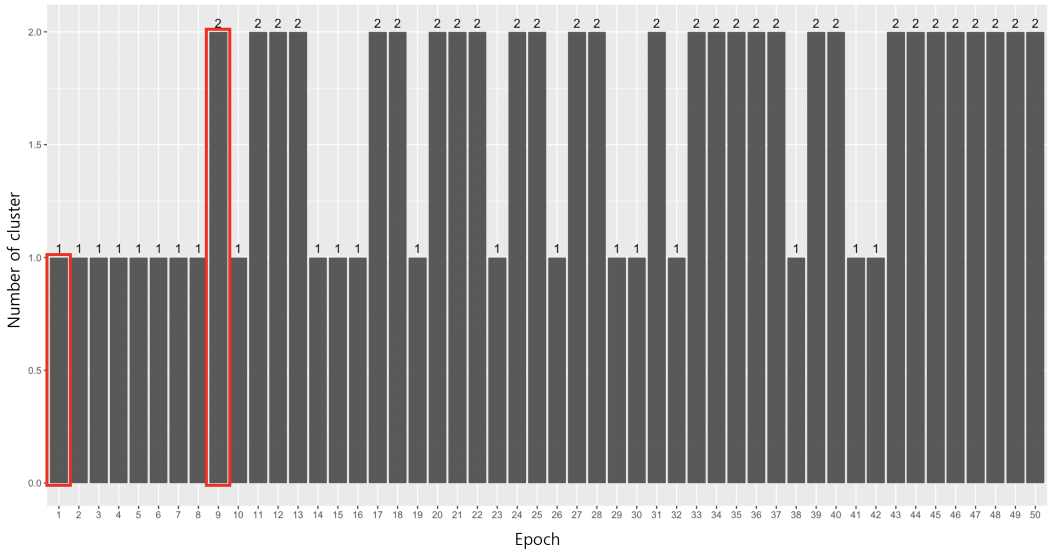

Figure 8.13: Number of density clusters in each epoch: *-eyse*

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.

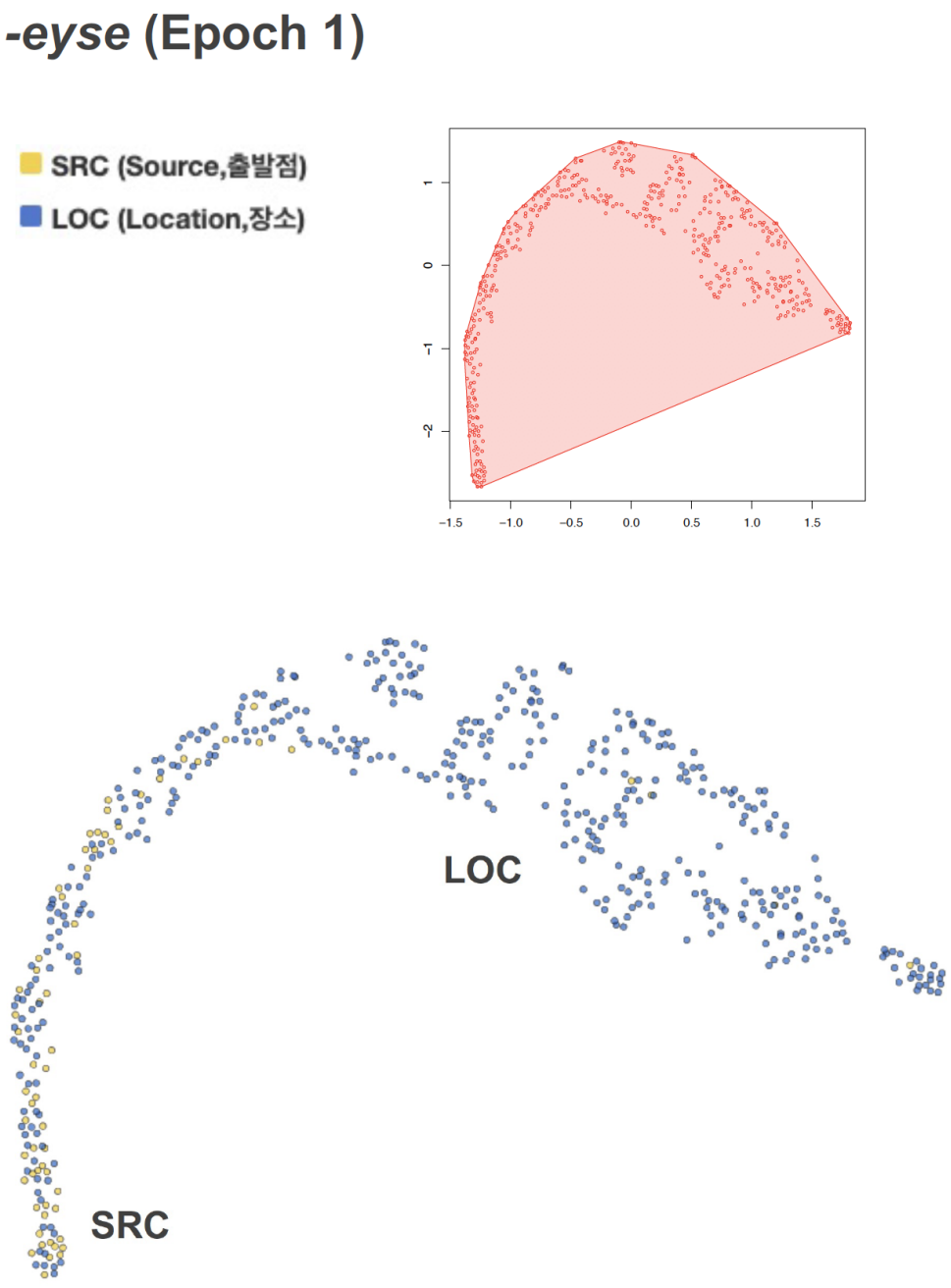

Figure 8.14: The distributional map for *-eyse* in epoch one

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.

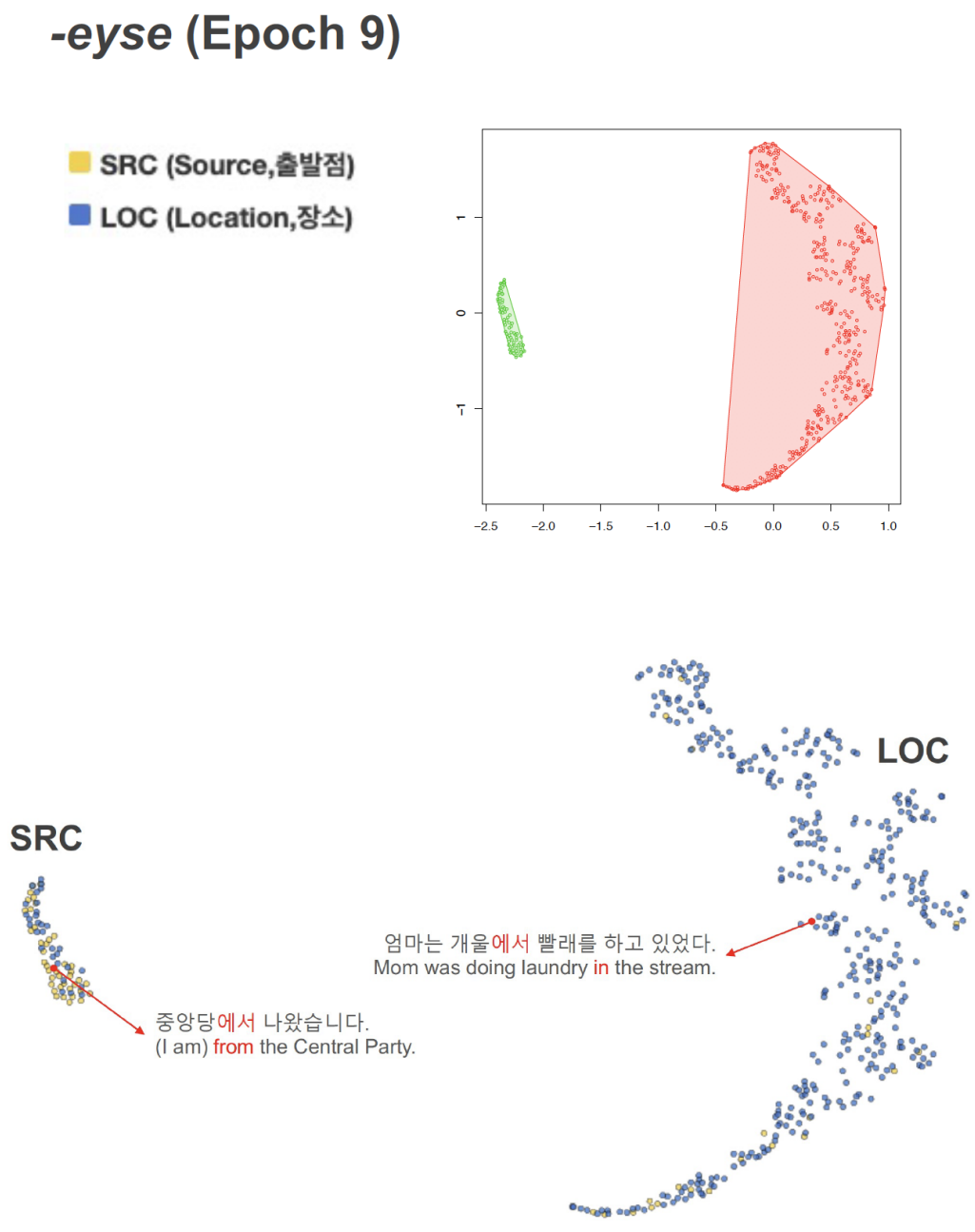

Figure 8.15: The distributional map for *-eyse* in epoch nine

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).

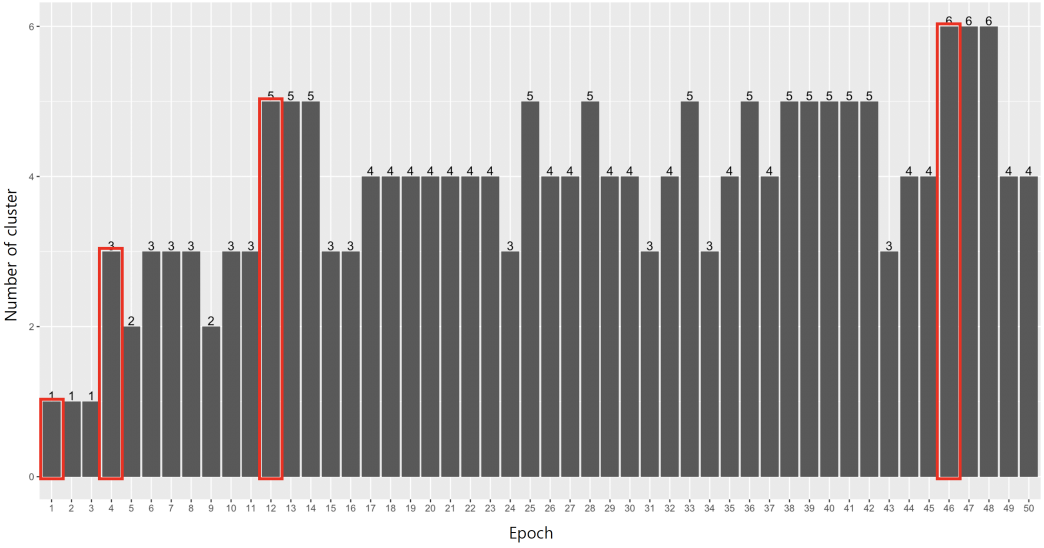

Figure 8.16: Number of density clusters in each epoch: *-(u)lo*

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.

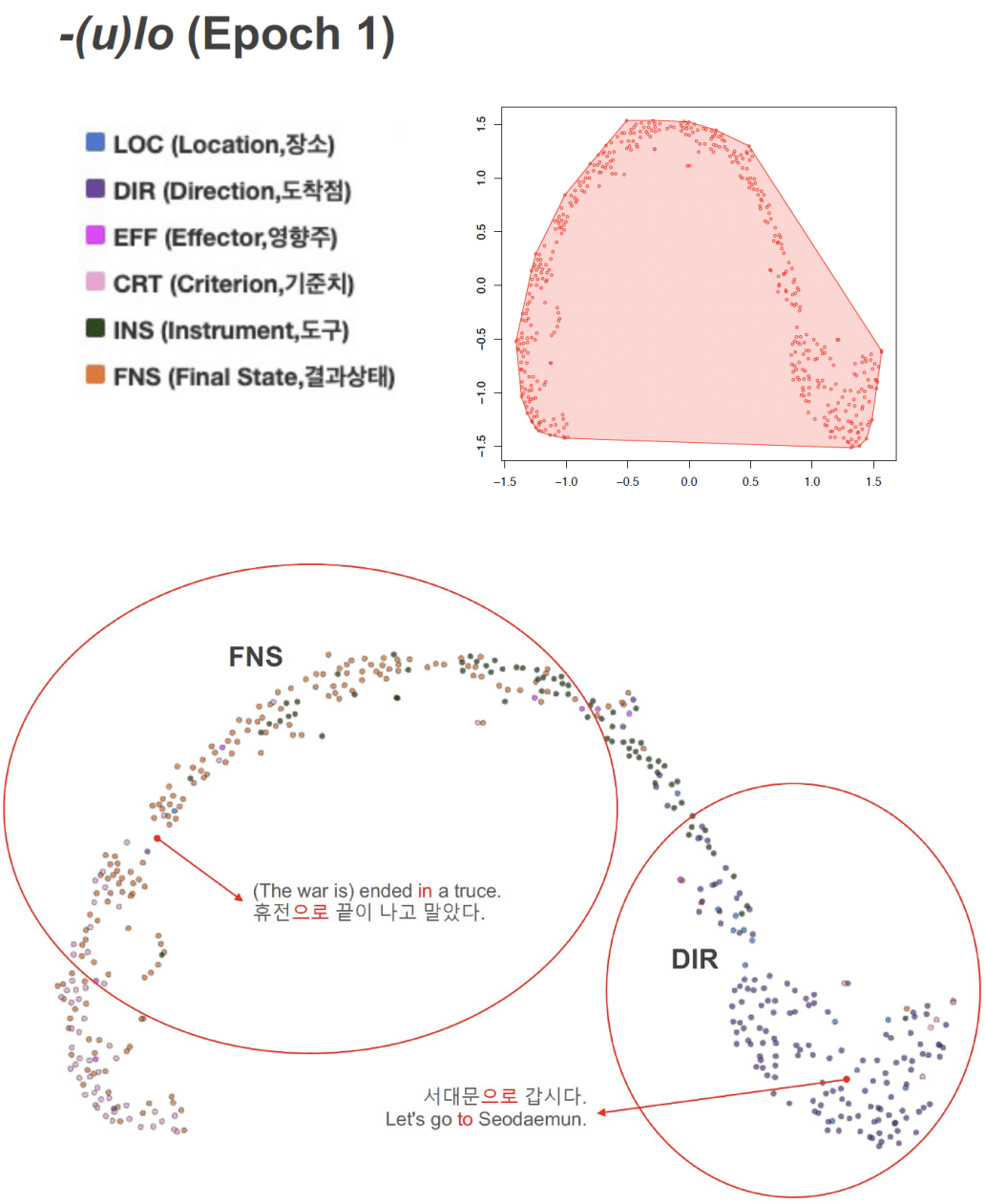

Figure 8.17: The distributional map for *-(u)lo* in epoch one

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).

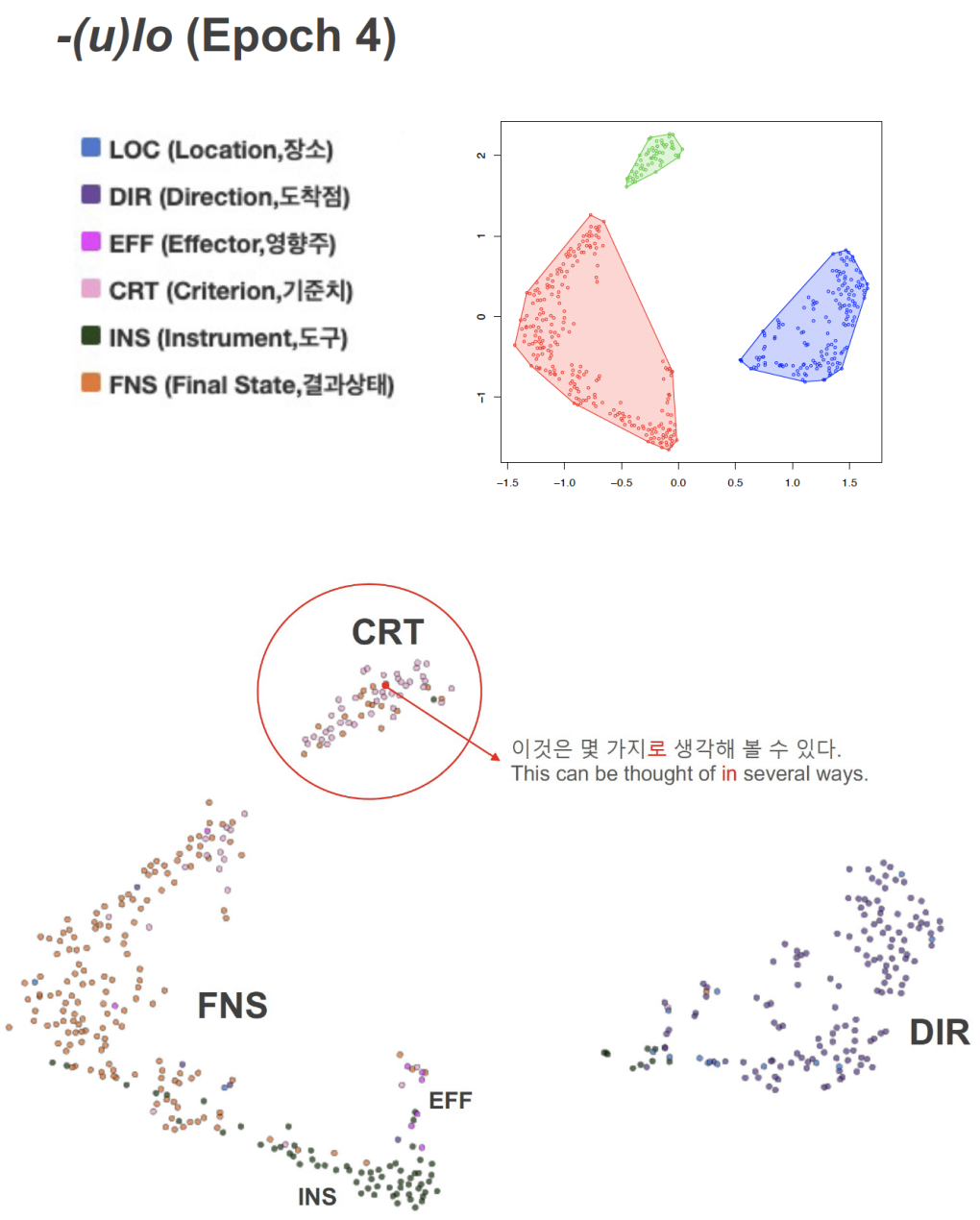

Figure 8.18: The distributional map for *-(u)lo* in epoch four

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.

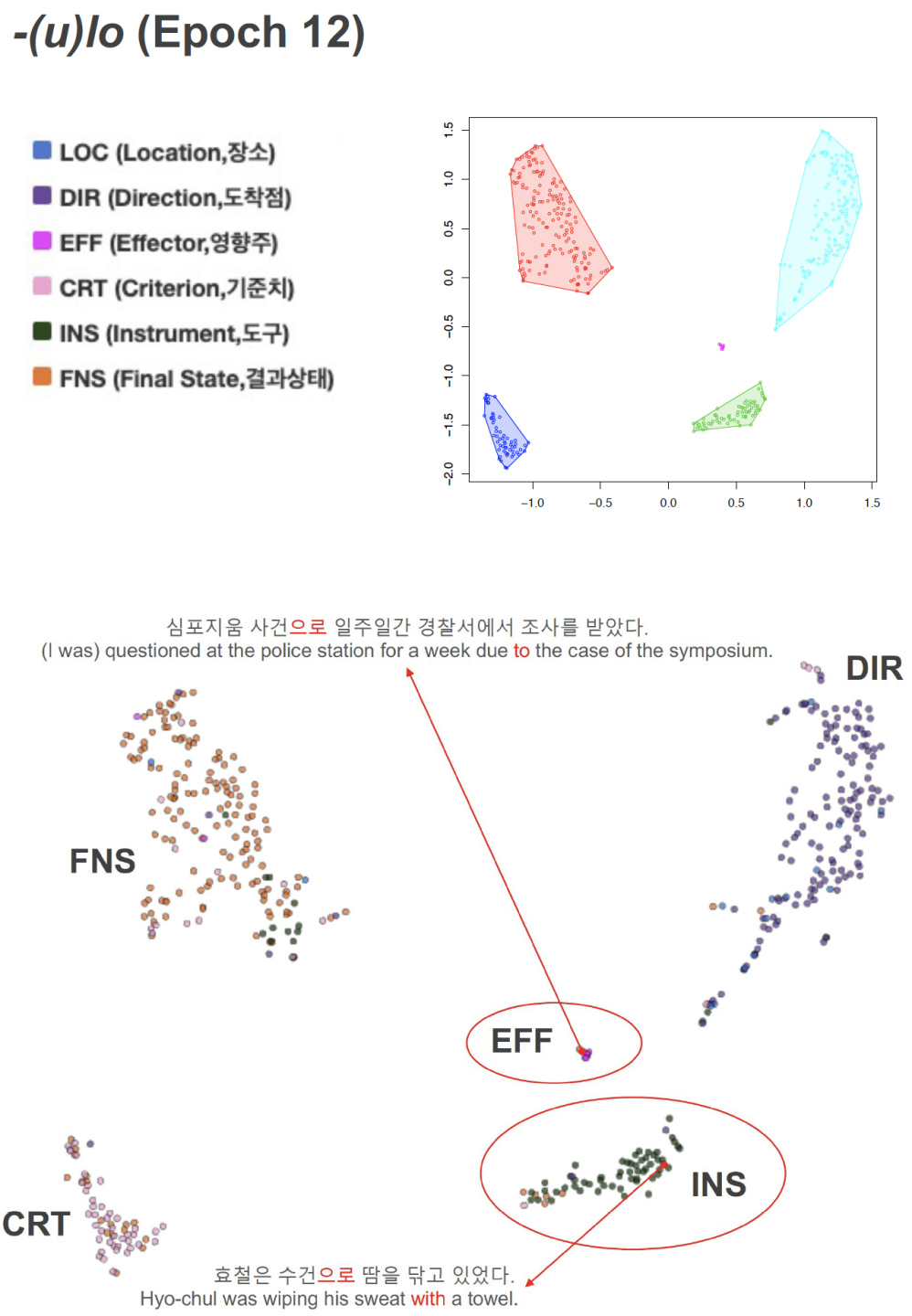

Figure 8.19: The distributional map for *-(u)lo* in epoch 12

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.

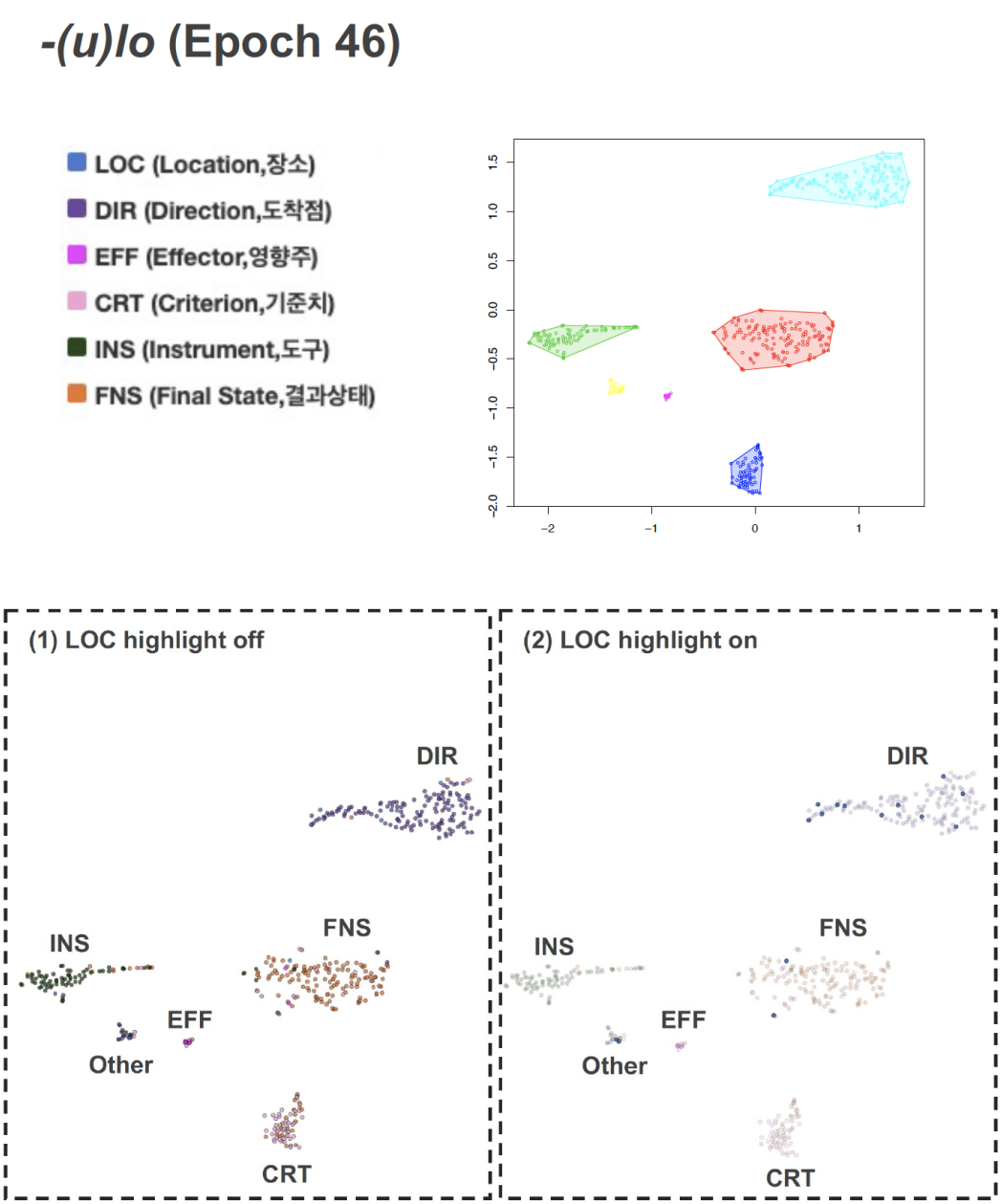

Figure 8.20: The distributional map for *-(u)lo* in epoch 46

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 re-stated 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 epochprogressed. 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 com-

bination 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 informa-

tion 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.

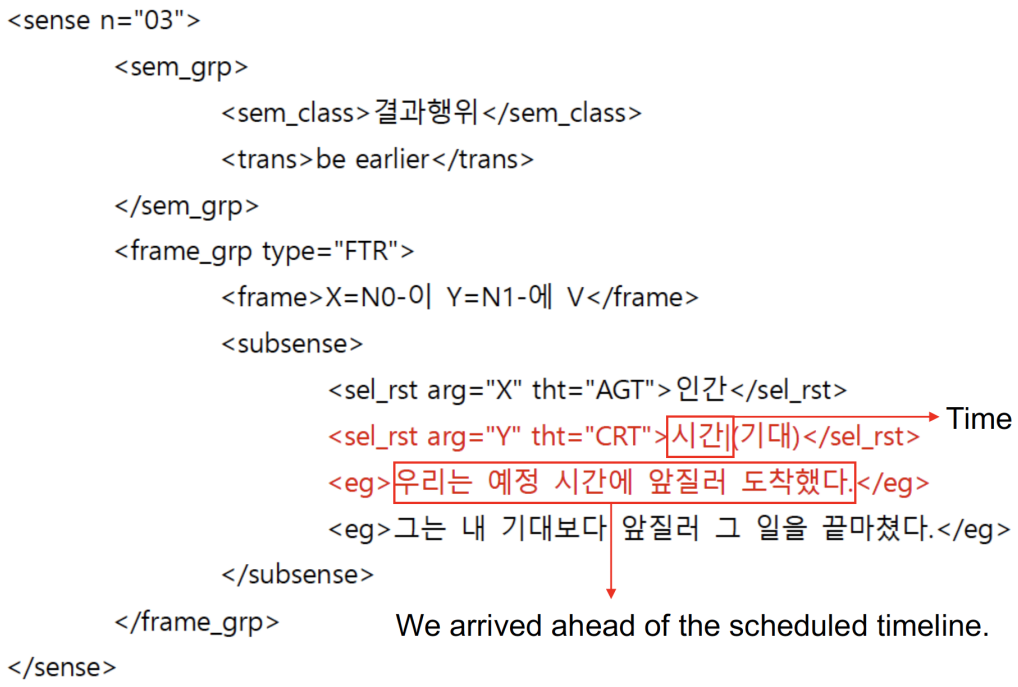

Figure 10.1: Example of an error extracted from the file *V-aphciluta* in the Sejong Electronic Dictionary

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 lan-

guage 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.

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

```
1
2 class PPMI_SVD_Algorithm:
3
4     def __init__ (self, fold, postposition, postposition_ko,
          window):
5         self.fold = fold
6         self.postposition = postposition
7         self.postposition_ko = postposition_ko
8         self.window = window
9
10    def PPMI_SVD_Calculation(self):
11
12        from collections import Counter
13        import itertools
```

```
14          import nltk
15          from nltk.corpus import stopwords
16          import numpy as np
17          import pandas as pd
18          from scipy import sparse
19          from scipy.sparse import linalg
20          from sklearn.preprocessing import normalize
21          from sklearn.metrics.pairwise import cosine_similarity
22
23          trainDir = "../../Data/Input/Fold/" + str(self.fold) +
                "Fold/"+ self.postposition +"_train_" + str(self.
                fold) + ".csv"
24
25          # data load
26          df = pd.read_csv(trainDir)
27          print(df.head())
28          headlines = df['Sentence'].tolist()
29          headlines = [[tok for tok in headline.split()] for
                headline in headlines]
30          # remove single word headlines
31          headlines = [hl for hl in headlines if len(hl) > 1]
32          # show results
33          print(headlines[0:20])
34
35          # calculate a unigram vocabulary
36          tok2indx = dict()
37          unigram_counts = Counter()
38          for ii, headline in enumerate(headlines):
39              if ii % 200000 == 0:
40                  print(f'finished {ii / len(headlines):.2%} of
                        headlines')
```

```
41              for token in headline:
42                  unigram_counts[token] += 1
43                  if token not in tok2indx:
44                      tok2indx[token] = len(tok2indx)
45          indx2tok = {indx: tok for tok, indx in tok2indx.items()
                }
46          print('done')
47          print('vocabulary size: {}'.format(len(unigram_counts))
                )
48          print('most common: {}'.format(unigram_counts.
                most_common(10)))
49
50          wordType = len(unigram_counts);
51
52          for j in range(1, self.window):
53              # Skipgrams
54              back_window = j
55              front_window = j
56              skipgram_counts = Counter()
57              for iheadline, headline in enumerate(headlines):
58                  for ifw, fw in enumerate(headline):
59                      icw_min = max(0, ifw - back_window)
60                      icw_max = min(len(headline) - 1, ifw +
                            front_window)
61                      icws = [ii for ii in range(icw_min, icw_max
                            + 1) if ii != ifw]
62                      for icw in icws:
63                          skipgram = (headline[ifw], headline[icw
                                ])
64                          skipgram_counts[skipgram] += 1
65                  if iheadline % 200000 == 0:
```

```
66                 print(f'finished {iheadline / len(headlines
                       ):.2%} of headlines')
67         print('done')
68         print('number of skipgrams: {}'.format(len(
               skipgram_counts)))
69         print('most common: {}'.format(skipgram_counts.
               most_common(10)))
70
71         # Word-Word Count Matrix
72         row_indxs = []
73         col_indxs = []
74         dat_values = []
75         ii = 0
76         for (tok1, tok2), sg_count in skipgram_counts.items
               ():
77             ii += 1
78             if ii % 1000000 == 0:
79                 print(f'finished {ii / len(skipgram_counts)
                       :.2%} of skipgrams')
80             tok1_indx = tok2indx[tok1]
81             tok2_indx = tok2indx[tok2]
82
83             row_indxs.append(tok1_indx)
84             col_indxs.append(tok2_indx)
85             dat_values.append(sg_count)
86
87         wwcnt_mat = sparse.csr_matrix((dat_values, (
               row_indxs, col_indxs)))
88         print('done')
89
90         # normalize each row using L2 norm
```

```
91              wwcnt_norm_mat = normalize(wwcnt_mat, norm='l2',
                    axis=1)
92
93              ##Word Similarity with Sparse Count Matrices
94              def ww_sim(word, mat, topn=len(tok2indx)):
95                  indx = tok2indx[word]
96                  if isinstance(mat, sparse.csr_matrix):
97                      v1 = mat.getrow(indx)
98                  else:
99                      v1 = mat[indx:indx + 1, :]
100                 sims = cosine_similarity(mat, v1).flatten()
101                 sindxs = np.argsort(-sims)
102                 sim_word_scores = [(indx2tok[sindx], sims[sindx
                        ]) for sindx in sindxs[0:topn]]
103                 return sim_word_scores
104
105             # Pointwise Mutual Information Matrices
106             num_skipgrams = wwcnt_mat.sum()
107             assert (sum(skipgram_counts.values()) ==
                    num_skipgrams)
108
109             # for creating sparce matrices
110             row_indxs = []
111             col_indxs = []
112
113             pmi_dat_values = []
114             ppmi_dat_values = []
115             spmi_dat_values = []
116             sppmi_dat_values = []
117
118             # smoothing
```

```
119             alpha = 0.75
120             nca_denom = np.sum(np.array(wwcnt_mat.sum(axis=0)).
                    flatten() ** alpha)
121             sum_over_words = np.array(wwcnt_mat.sum(axis=0)).
                    flatten()
122             sum_over_words_alpha = sum_over_words ** alpha
123             sum_over_contexts = np.array(wwcnt_mat.sum(axis=1))
                    .flatten()
124
125             ii = 0
126             for (tok1, tok2), sg_count in skipgram_counts.items
                    ():
127                 ii += 1
128                 if ii % 1000000 == 0:
129                     print(f'finished {ii / len(skipgram_counts)
                            :.2%} of skipgrams')
130                 tok1_indx = tok2indx[tok1]
131                 tok2_indx = tok2indx[tok2]
132
133                 nwc = sg_count
134                 Pwc = nwc / num_skipgrams
135                 nw = sum_over_contexts[tok1_indx]
136                 Pw = nw / num_skipgrams
137                 nc = sum_over_words[tok2_indx]
138                 Pc = nc / num_skipgrams
139
140                 nca = sum_over_words_alpha[tok2_indx]
141                 Pca = nca / nca_denom
142
143                 pmi = np.log2(Pwc / (Pw * Pc))
144                 ppmi = max(pmi, 0)
```

```
145
146                 spmi = np.log2(Pwc / (Pw * Pca))
147                 sppmi = max(spmi, 0)
148
149                 row_indxs.append(tok1_indx)
150                 col_indxs.append(tok2_indx)
151                 pmi_dat_values.append(pmi)
152                 ppmi_dat_values.append(ppmi)
153                 spmi_dat_values.append(spmi)
154                 sppmi_dat_values.append(sppmi)
155
156             pmi_mat = sparse.csr_matrix((pmi_dat_values, (
                    row_indxs, col_indxs)))
157             ppmi_mat = sparse.csr_matrix((ppmi_dat_values, (
                    row_indxs, col_indxs)))
158             spmi_mat = sparse.csr_matrix((spmi_dat_values, (
                    row_indxs, col_indxs)))
159             sppmi_mat = sparse.csr_matrix((sppmi_dat_values, (
                    row_indxs, col_indxs)))
160
161             print('done')
162
163             # Singular Value Decomposition
164             matrix_use = ppmi_mat
165
166             if wordType < 500:
167                 embedding_size = wordType - 1
168             else:
169                 embedding_size = 500
170
```

```
171             uu, ss, vv = linalg.svds(matrix_use, embedding_size
                    )
172             print('vocab size: {}'.format(len(unigram_counts)))
173             print('embedding size: {}'.format(embedding_size))
174             print('uu.shape: {}'.format(uu.shape))
175             print('ss.shape: {}'.format(ss.shape))
176             print('vv.shape: {}'.format(vv.shape))
177
178             unorm = uu / np.sqrt(np.sum(uu * uu, axis=1,
                    keepdims=True))
179             vnorm = vv / np.sqrt(np.sum(vv * vv, axis=0,
                    keepdims=True))
180             word_vecs = uu + vv.T
181             word_vecs_norm = word_vecs / np.sqrt(np.sum(
                    word_vecs * word_vecs, axis=1, keepdims=True))
182
183             print(word_vecs_norm)
184
185             from sklearn.manifold import TSNE
186             X_embedded = TSNE(n_components=2, random_state=0).
                    fit_transform(word_vecs_norm)
187
188             wordList = []
189             wordnum = 0
190             for typeeach in indx2tok:
191                 wordList.append(indx2tok[wordnum])
192                 wordnum += 1
193
194             tsne_df = pd.DataFrame({'X': X_embedded[:, 0], 'Y':
                     X_embedded[:, 1], 'Word': wordList})
```

```
195             tsne_df.to_csv("../../Data/Output/PPMI_SVD/" + self
                    .postposition + "/t-SNE/" + self.postposition +
                    "_tSNE_" + str(
196                         j) + ".csv")
197
198             TSNE_dic = {}
199
200             typenum = 0
201             for typeeach in indx2tok:
202                 TSNE_dic[indx2tok[typenum]] = [X_embedded[
                        typenum][0], X_embedded[typenum][1]]
203                 typenum = typenum + 1
204
205             functionEy = ["LOC", "GOL", "EFF", "CRT", "THM", "
                    INS", "AGT", "FNS"]
206             functionEyse = ["SRC", "LOC"]
207             functionLo = ["FNS", "INS", "DIR", "EFF", "CRT", "
                    LOC"]
208
209             if self.postposition == "Ey":
210                 for function in functionEy:
211                     word = self.postposition_ko + "/JKB" + "_"
                            + function
212                     from numpy import dot
213                     from numpy.linalg import norm
214                     import numpy as np
215                     def cos_sim(A, B):
216                         return dot(A, B) / (norm(A) * norm(B))
217                     target = np.array(TSNE_dic[word])
218                     outDir = "../../Data/Output/PPMI_SVD/" +
                            self.postposition + "/Similarity/" +
```

```
                        self.postposition + "_" + function + "
                        _Similarity_" + str(
219                     j) + ".csv"
220                 f = open(outDir, 'w')
221                 tsnenum = 0
222                 for typeeach in indx2tok:
223                     if indx2tok[tsnenum] != self.
                            postposition:
224                         source = np.array(TSNE_dic[indx2tok
                                [tsnenum]])
225                         normal_sim = (cos_sim(target,
                                source) + 1) / 2
226                         data = str(indx2tok[tsnenum]) + ","
                                 + str(normal_sim)
227                         f.write(data + "\n")
228                     tsnenum = tsnenum + 1
229                 f.close()
230
231             elif self.postposition == "Eyse":
232                 for function in functionEyse:
233                     word = self.postposition_ko + "/JKB" + "_"
                            + function
234                     from numpy import dot
235                     from numpy.linalg import norm
236                     import numpy as np
237                     def cos_sim(A, B):
238                         return dot(A, B) / (norm(A) * norm(B))
239                     target = np.array(TSNE_dic[word])
240                     outDir = "../../Data/Output/PPMI_SVD/" +
                            self.postposition + "/Similarity/" +
                            self.postposition + "_" + function + "
```

```
                    _Similarity_" + str(
241                     j) + ".csv"
242                 f = open(outDir, 'w')
243                 tsnenum = 0
244                 for typeeach in indx2tok:
245                     if indx2tok[tsnenum] != self.
                            postposition:
246                         source = np.array(TSNE_dic[indx2tok
                                [tsnenum]])
247                         normal_sim = (cos_sim(target,
                                source) + 1) / 2
248                         data = str(indx2tok[tsnenum]) + ","
                                 + str(normal_sim)
249                         f.write(data + "\n")
250                     tsnenum = tsnenum + 1
251                 f.close()
252
253             elif self.postposition == "Lo":
254                 for function in functionLo:
255                     word = self.postposition_ko + "/JKB" + "_"
                            + function
256                     from numpy import dot
257                     from numpy.linalg import norm
258                     import numpy as np
259                     def cos_sim(A, B):
260                         return dot(A, B) / (norm(A) * norm(B))
261                     target = np.array(TSNE_dic[word])
262                     outDir = "../../Data/Output/PPMI_SVD/" +
                            self.postposition + "/Similarity/" +
                            self.postposition + "_" + function + "
                            _Similarity_" + str(
```

```
263                     j) + ".csv"
264                 f = open(outDir, 'w')
265                 tsnenum = 0
266                 for typeeach in indx2tok:
267                     if indx2tok[tsnenum] != self.
                            postposition:
268                         source = np.array(TSNE_dic[indx2tok
                                [tsnenum]])
269                         normal_sim = (cos_sim(target,
                                source) + 1) / 2
270                         data = str(indx2tok[tsnenum]) + ","
                                 + str(normal_sim)
271                         f.write(data + "\n")
272                     tsnenum = tsnenum + 1
273                 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())
```

```
27          headlines = df['Sentence'].tolist()
28          headlines = [[tok for tok in headline.split()] for
                headline in headlines]
29          # remove single word headlines
30          headlines = [hl for hl in headlines if len(hl) > 1]
31          # show results
32          print(headlines[0:20])
33
34          # calculate a unigram vocabulary
35          tok2indx = dict()
36          unigram_counts = Counter()
37          for ii, headline in enumerate(headlines):
38              if ii % 200000 == 0:
39                  print(f'finished {ii / len(headlines):.2%} of
                        headlines')
40              for token in headline:
41                  unigram_counts[token] += 1
42                  if token not in tok2indx:
43                      tok2indx[token] = len(tok2indx)
44          indx2tok = {indx: tok for tok, indx in tok2indx.items()
                }
45          print('done')
46          print('vocabulary size: {}'.format(len(unigram_counts))
                )
47          print('most common: {}'.format(unigram_counts.
                most_common(10)))
48
49          wordType = len(unigram_counts);
50
51          for j in range(1, self.window):
52
```

```
53              from gensim.models import Word2Vec
54
55              if wordType < 500:
56                  embedding_size = wordType - 1
57              else:
58                  embedding_size = 500
59
60              embedding_model = Word2Vec(headlines, size=
                    embedding_size, window=j, min_count=0, workers=4
                    , iter=100, sg=1, negative=15, ns_exponent=0.75)
61
62              matrix = []
63              indxnum = 0
64              for typeeach in indx2tok:
65                  line = list(embedding_model[indx2tok[typeeach
                        ]])
66                  matrix.append(line)
67                  indxnum = indxnum + 1
68
69              embedded_matrix = np.array(matrix, dtype=np.float64
                    )
70
71              print(embedded_matrix)
72
73              from sklearn.manifold import TSNE
74              X_embedded = TSNE(n_components=2, random_state=0).
                    fit_transform(embedded_matrix)
75
76              wordList = []
77              wordnum = 0
78              for typeeach in indx2tok:
```

```
79                 wordList.append(indx2tok[wordnum])
80                 wordnum += 1
81
82             tsne_df = pd.DataFrame({'X': X_embedded[:, 0], 'Y':
                   X_embedded[:, 1], 'Word': wordList})
83             tsne_df.to_csv(
84                 "../../Data/Output/SGNS/" + self.postposition +
                       "/t-SNE/" + self.postposition + "_tSNE_" +
                      str(
85                       j) + ".csv")
86
87             # Word Similarity with Sparse Count Matrices
88             def ww_sim(word, mat, topn=len(tok2indx)):
89                 indx = tok2indx[word]
90                 if isinstance(mat, sparse.csr_matrix):
91                     v1 = mat.getrow(indx)
92                 else:
93                     v1 = mat[indx:indx + 1, :]
94                 sims = cosine_similarity(mat, v1).flatten()
95                 sindxs = np.argsort(-sims)
96                 sim_word_scores = [(indx2tok[sindx], sims[sindx
                       ]) for sindx in sindxs[0:topn]]
97                 return sim_word_scores
98
99             # Word Similarity
100            def word_sim_report(word, sim_mat):
101                output = {}
102                sim_word_scores = ww_sim(word, embedded_matrix)
103                for sim_word, sim_score in sim_word_scores:
104                    output[sim_word] = ((sim_score + 1) / 2)
105                return output
```

```
106
107             TSNE_dic = {}
108
109             typenum = 0
110             for typeeach in indx2tok:
111                 TSNE_dic[indx2tok[typenum]] = [X_embedded[
                        typenum][0], X_embedded[typenum][1]]
112                 typenum = typenum + 1
113
114             functionEy = ["LOC", "GOL", "EFF", "CRT", "THM", "
                    INS", "AGT", "FNS"]
115             functionEyse = ["SRC", "LOC"]
116             functionLo = ["FNS", "INS", "DIR", "EFF", "CRT", "
                    LOC"]
117
118             if self.postposition == "Ey":
119                 for function in functionEy:
120                     word = self.postposition_ko + "/JKB" + "_"
                            + function
121                     from numpy import dot
122                     from numpy.linalg import norm
123                     import numpy as np
124                     def cos_sim(A, B):
125                         return dot(A, B) / (norm(A) * norm(B))
126                     target = np.array(TSNE_dic[word])
127                     outDir = "../../Data/Output/SGNS/" + self.
                            postposition + "/Similarity/" + self.
                            postposition + "_" + function + "
                            _Similarity_" + str(
128                         j) + ".csv"
129                     f = open(outDir, 'w')
```

```
130                     tsnenum = 0
131                     for typeeach in indx2tok:
132                         if indx2tok[tsnenum] != self.
                                postposition:
133                             source = np.array(TSNE_dic[indx2tok
                                    [tsnenum]])
134                             normal_sim = (cos_sim(target,
                                    source) + 1) / 2
135                             data = str(indx2tok[tsnenum]) + ","
                                     + str(normal_sim)
136                             f.write(data + "\n")
137                         tsnenum = tsnenum + 1
138                     f.close()
139
140
141
142
143             elif self.postposition == "Eyse":
144                 for function in functionEyse:
145                     word = self.postposition_ko + "/JKB" + "_"
                            + function
146                     from numpy import dot
147                     from numpy.linalg import norm
148                     import numpy as np
149                     def cos_sim(A, B):
150                         return dot(A, B) / (norm(A) * norm(B))
151                     target = np.array(TSNE_dic[word])
152                     outDir = "../../Data/Output/SGNS/" + self.
                            postposition + "/Similarity/" + self.
                            postposition + "_" + function + "
                            _Similarity_" + str(
```

```
153                         j) + ".csv"
154                     f = open(outDir, 'w')
155                     tsnenum = 0
156                     for typeeach in indx2tok:
157                         if indx2tok[tsnenum] != self.
                                postposition:
158                             source = np.array(TSNE_dic[indx2tok
                                    [tsnenum]])
159                             normal_sim = (cos_sim(target,
                                    source) + 1) / 2
160                             data = str(indx2tok[tsnenum]) + ","
                                     + str(normal_sim)
161                             f.write(data + "\n")
162                         tsnenum = tsnenum + 1
163                     f.close()
164
165             elif self.postposition == "Lo":
166                 for function in functionLo:
167                     word = self.postposition_ko + "/JKB" + "_"
                            + function
168                     from numpy import dot
169                     from numpy.linalg import norm
170                     import numpy as np
171                     def cos_sim(A, B):
172                         return dot(A, B) / (norm(A) * norm(B))
173                     target = np.array(TSNE_dic[word])
174                     outDir = "../../Data/Output/SGNS/" + self.
                            postposition + "/Similarity/" + self.
                            postposition + "_" + function + "
                            _Similarity_" + str(
175                         j) + ".csv"
```

```
176                     f = open(outDir, 'w')
177                     tsnenum = 0
178                     for typeeach in indx2tok:
179                         if indx2tok[tsnenum] != self.
                                postposition:
180                             source = np.array(TSNE_dic[indx2tok
                                    [tsnenum]])
181                             normal_sim = (cos_sim(target,
                                    source) + 1) / 2
182                             data = str(indx2tok[tsnenum]) + ","
                                     + str(normal_sim)
183                             f.write(data + "\n")
184                         tsnenum = tsnenum + 1
185                     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)):
```

```
26                 functionDic[words[k]] = sims[k]
27             functionDicDic[function] = functionDic
28
29         testDir = "../../Data/Input/Fold/" + str(self.fold) + "
               Fold/" + self.postposition + "_test_" + str(self.
               fold) + ".csv"
30
31         df = pd.read_csv(testDir)
32         headlines = df['Sentence'].tolist()
33
34         countDic = {}
35         frequencyDic = {}
36
37         countDic["Total"] = 0
38         frequencyDic["Total"] = 0
39
40         for function in functions:
41             countDic[function] = 0
42             frequencyDic[function] = 0
43
44         for sentence in headlines:
45             originClass = ""
46             token = sentence.split(" ")
47             for eachToken in token:
48                 if (self.postposition_ko + "/") in eachToken
                       and "JKB_" in eachToken:
49                     originClass = eachToken.replace((self.
                           postposition_ko + "/"), "").replace("
                           JKB_", "")
50
51             classifiedFunc = {}
```

```
52
53              funcScore = {}
54              matchNum = 0
55
56              for eachToken in token:
57                  if functionDicDic.get("LOC").get(eachToken.
                        strip()) == None or ((self.postposition_ko +
                         "/") in eachToken and "JKB_" in eachToken):
58                      pass
59                  else:
60                      matchNum = matchNum + 1
61                      for function in functions:
62                          if funcScore.get(function) == None:
63                              funcScore[function] =
                                    functionDicDic.get(function).get
                                    (eachToken.strip())
64                          else:
65                              funcScore[function] = funcScore.get
                                    (function) + functionDicDic.get(
                                    function).get(eachToken.strip())
66
67              for function in functions:
68                  classifiedFunc[function] = funcScore.get(
                        function) / matchNum
69
70              dic_max = max(classifiedFunc.values())
71
72              for x, y in classifiedFunc.items():
73                  if y == dic_max:
74                      for function in functions:
75                          if originClass == function and
```

```
                            originClass == x:
76                              countDic[function] = countDic.get(
                                    function) + 1
77                      if originClass == x:
78                          countDic["Total"] = countDic.get("Total
                                ") + 1
79
80              frequencyDic["Total"] = frequencyDic.get("Total") +
                    1
81
82              for function in functions:
83                  if originClass == function:
84                      frequencyDic[function] = frequencyDic.get(
                            function) + 1
85
86          finalResult = {}
87
88          totalAccuracy = countDic.get("Total") / frequencyDic.
                get("Total")
89          finalResult["Total"] = totalAccuracy
90
91          for function in functions:
92              funcAccuracy = countDic.get(function) /
                    frequencyDic.get(function)
93              finalResult[function] = funcAccuracy
94
95          averageAccuracy = 0
96          for function in functions:
97              averageAccuracy = averageAccuracy+finalResult.get(
                    function)
98
```

```
99          finalResult["TotalAverage"] = averageAccuracy / len(
                functions)
100
101         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 *BertForSequenceClassification*

```
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':
```

```
14          print('Found GPU at: {}'.format(device_name))
15      else:
16          raise SystemError('GPU device not found')
17
18      import torch
19      # If there's a GPU available...
20      if torch.cuda.is_available():
21          # Tell PyTorch to use the GPU.
22          device = torch.device("cuda")
23          print('There are %d GPU(s) available.' % torch.cuda
                .device_count())
24          print('We will use the GPU:', torch.cuda.
                get_device_name(0))
25      # If not...
26      else:
27          print('No GPU available, using the CPU instead.')
28          device = torch.device("cpu")
29
30      !pip install transformers
31
32      # Mount Google Drive to this Notebook instance.
33      from google.colab import drive
34      drive.mount('/content/drive')
35
36      import tensorflow as tf
37      import torch
38      from transformers import BertTokenizer
39      from transformers import BertForSequenceClassification,
            AdamW, BertConfig
40      from transformers import
            get_linear_schedule_with_warmup
```

```
41          from torch.utils.data import TensorDataset, DataLoader,
                RandomSampler, SequentialSampler
42          from keras.preprocessing.sequence import pad_sequences
43          from sklearn.model_selection import train_test_split
44          import pandas as pd
45          import numpy as np
46          import random
47          import time
48          import datetime
49
50          fileDir = "drive/My Drive/BERT/SM/KoBERT/Postposition/
                Data/test_"+self.postposition+".csv"
51          fr = open(fileDir, 'r')
52          contents= fr.readlines()
53          fr.close()
54
55          test = pd.DataFrame(columns=('index', 'Label', '
                Sentence'))
56          i = 0
57          index = ""
58          label = ""
59          sentence = ""
60          for content in contents:
61              if i == 0:
62                  pass
63              else:
64                  infos = content.split(",")
65                  index = infos[0]
66                  label = int(infos[1])
67                  sentence = infos[2].replace("\n","")
68                  test.loc[i] = [index, label, sentence]
```

```
69              i = i + 1
70
71          fileDir = "drive/My Drive/BERT/SM/KoBERT/Postposition/
                Data/train_"+self.postposition+".csv"
72          fr = open(fileDir, 'r')
73          contents= fr.readlines()
74          fr.close()
75          train = pd.DataFrame(columns=('index', 'Label', '
                Sentence'))
76          i = 0
77          index = ""
78          label = ""
79          sentence = ""
80          for content in contents:
81              if i == 0:
82                  pass
83              else:
84                  infos = content.split(",")
85                  index = infos[0]
86                  label = int(infos[1])
87                  sentence = infos[2].replace("\n","")
88                  train.loc[i] = [index, label, sentence]
89              i = i + 1
90
91          sentences = train['Sentence']
92          sentences = ["[CLS] " + str(sentence) + " [SEP]" for
                sentence in sentences]
93
94          labels = train['Label'].values
95          labels_re = []
96          for label in labels:
```

```
97            labels_re.append(label)
98        labels = labels_re
99
100       tokenizer = KoBertTokenizer.from_pretrained('monologg/
              kobert')
101       tokenized_texts = [tokenizer.tokenize(sent) for sent in
               sentences]
102
103       MAX_LEN = 128
104       input_ids = [tokenizer.convert_tokens_to_ids(x) for x
              in tokenized_texts]
105       input_ids = pad_sequences(input_ids, maxlen=MAX_LEN,
              dtype="long", truncating="post", padding="post")
106
107       attention_masks = []
108       for seq in input_ids:
109           seq_mask = [float(i>0) for i in seq]
110           attention_masks.append(seq_mask)
111
112       train_inputs, validation_inputs, train_labels,
              validation_labels = train_test_split(input_ids,
              labels,random_state=2018,test_size=0.1)
113       train_masks, validation_masks, _, _ = train_test_split(
              attention_masks,input_ids,random_state=2018,
              test_size=0.1)
114       train_inputs = torch.tensor(train_inputs)
115       train_labels = torch.tensor(train_labels)
116       train_masks = torch.tensor(train_masks)
117       validation_inputs = torch.tensor(validation_inputs)
118       validation_labels = torch.tensor(validation_labels)
119       validation_masks = torch.tensor(validation_masks)
```

```
120
121         batch_size = 32
122
123         train_data = TensorDataset(train_inputs, train_masks,
                train_labels)
124         train_sampler = RandomSampler(train_data)
125         train_dataloader = DataLoader(train_data, sampler=
                train_sampler, batch_size=batch_size)
126         validation_data = TensorDataset(validation_inputs,
                validation_masks, validation_labels)
127         validation_sampler = SequentialSampler(validation_data)
128         validation_dataloader = DataLoader(validation_data,
                sampler=validation_sampler, batch_size=batch_size)
129
130         sentences = test['Sentence']
131         sentences = ["[CLS] " + str(sentence) + " [SEP]" for
                sentence in sentences]
132
133         labels = test['Label'].values
134         labels_re = []
135         for label in labels:
136           labels_re.append(label)
137         labels = labels_re
138
139         tokenizer = KoBertTokenizer.from_pretrained('monologg/
                kobert')
140         tokenized_texts = [tokenizer.tokenize(sent) for sent in
                 sentences]
141
142         MAX_LEN = 128
```

```
143         input_ids = [tokenizer.convert_tokens_to_ids(x) for x
                in tokenized_texts]
144         input_ids = pad_sequences(input_ids, maxlen=MAX_LEN,
                dtype="long", truncating="post", padding="post")
145
146         attention_masks = []
147
148         for seq in input_ids:
149             seq_mask = [float(i>0) for i in seq]
150             attention_masks.append(seq_mask)
151         test_inputs = torch.tensor(input_ids)
152         test_labels = torch.tensor(labels)
153         test_masks = torch.tensor(attention_masks)
154
155         batch_size = 32
156
157         test_data = TensorDataset(test_inputs, test_masks,
                test_labels)
158         test_sampler = RandomSampler(test_data)
159         test_dataloader = DataLoader(test_data, sampler=
                test_sampler, batch_size=batch_size)
160         test_dataloader
161
162         model = BertForSequenceClassification.from_pretrained("
                monologg/kobert", num_labels=self.lablesNum)
163         model.cuda()
164
165         def format_time(elapsed):
166             elapsed_rounded = int(round((elapsed)))
167             return str(datetime.timedelta(seconds=
                    elapsed_rounded))
```

```
168
169         optimizer = AdamW(model.parameters(),lr = 2e-5,eps = 1e
                -8)
170         epochs = 50
171         total_steps = len(train_dataloader) * epochs
172         scheduler = get_linear_schedule_with_warmup(optimizer,
                num_warmup_steps = 0,num_training_steps =
                total_steps)
173
174         seed_val = 42
175         random.seed(seed_val)
176         np.random.seed(seed_val)
177         torch.manual_seed(seed_val)
178         torch.cuda.manual_seed_all(seed_val)
179
180         if self.postposition == "Ey":
181
182             def flat_accuracy(preds, labels):
183                 pred_flat = np.argmax(preds, axis=1).flatten()
184                 labels_flat = labels.flatten()
185                 return np.sum(pred_flat == labels_flat) / len(
                        labels_flat)
186
187             def FNS_flat_accuracy(preds, labels):
188                 pred_flat = np.argmax(preds, axis=1).flatten()
189                 labels_flat = labels.flatten()
190                 match_num = 0
191                 func_num = 0
192                 for i in range(0,len(pred_flat)):
193                   if (pred_flat[i] == labels_flat[i]) and (
                        labels_flat[i] == 0):
```

```
194                     match_num += 1
195                 if labels_flat[i] == 0:
196                     func_num += 1
197             if match_num == 0 or func_num == 0:
198                 return 0
199             else:
200                 return match_num / func_num
201
202         def INS_flat_accuracy(preds, labels):
203             pred_flat = np.argmax(preds, axis=1).flatten()
204             labels_flat = labels.flatten()
205             match_num = 0
206             func_num = 0
207             for i in range(0,len(pred_flat)):
208                 if (pred_flat[i] == labels_flat[i]) and (
                        labels_flat[i] == 1):
209                     match_num += 1
210                 if labels_flat[i] == 1:
211                     func_num += 1
212             if match_num == 0 or func_num == 0:
213                 return 0
214             else:
215                 return match_num / func_num
216
217         def GOL_flat_accuracy(preds, labels):
218             pred_flat = np.argmax(preds, axis=1).flatten()
219             labels_flat = labels.flatten()
220             match_num = 0
221             func_num = 0
222             for i in range(0,len(pred_flat)):
```

```
223                 if (pred_flat[i] == labels_flat[i]) and (
                        labels_flat[i] == 2):
224                   match_num += 1
225                 if labels_flat[i] == 2:
226                   func_num += 1
227               if match_num == 0 or func_num == 0:
228                 return 0
229               else:
230                 return match_num / func_num
231
232             def EFF_flat_accuracy(preds, labels):
233               pred_flat = np.argmax(preds, axis=1).flatten()
234               labels_flat = labels.flatten()
235               match_num = 0
236               func_num = 0
237               for i in range(0,len(pred_flat)):
238                 if (pred_flat[i] == labels_flat[i]) and (
                        labels_flat[i] == 3):
239                   match_num += 1
240                 if labels_flat[i] == 3:
241                   func_num += 1
242               if match_num == 0 or func_num == 0:
243                 return 0
244               else:
245                 return match_num / func_num
246
247             def CRT_flat_accuracy(preds, labels):
248               pred_flat = np.argmax(preds, axis=1).flatten()
249               labels_flat = labels.flatten()
250               match_num = 0
251               func_num = 0
```

```
252                 for i in range(0,len(pred_flat)):
253                   if (pred_flat[i] == labels_flat[i]) and (
                          labels_flat[i] == 4):
254                     match_num += 1
255                   if labels_flat[i] == 4:
256                     func_num += 1
257                 if match_num == 0 or func_num == 0:
258                   return 0
259                 else:
260                   return match_num / func_num
261
262             def LOC_flat_accuracy(preds, labels):
263                 pred_flat = np.argmax(preds, axis=1).flatten()
264                 labels_flat = labels.flatten()
265                 match_num = 0
266                 func_num = 0
267                 for i in range(0,len(pred_flat)):
268                   if (pred_flat[i] == labels_flat[i]) and (
                          labels_flat[i] == 5):
269                     match_num += 1
270                   if labels_flat[i] == 5:
271                     func_num += 1
272                 if match_num == 0 or func_num == 0:
273                   return 0
274                 else:
275                   return match_num / func_num
276
277             def AGT_flat_accuracy(preds, labels):
278                 pred_flat = np.argmax(preds, axis=1).flatten()
279                 labels_flat = labels.flatten()
280                 match_num = 0
```

```
281                 func_num = 0
282                 for i in range(0,len(pred_flat)):
283                   if (pred_flat[i] == labels_flat[i]) and (
                          labels_flat[i] == 6):
284                     match_num += 1
285                   if labels_flat[i] == 6:
286                     func_num += 1
287                 if match_num == 0 or func_num == 0:
288                   return 0
289                 else:
290                   return match_num / func_num
291
292               def THM_flat_accuracy(preds, labels):
293                 pred_flat = np.argmax(preds, axis=1).flatten()
294                 labels_flat = labels.flatten()
295                 match_num = 0
296                 func_num = 0
297                 for i in range(0,len(pred_flat)):
298                   #print(pred_flat[i]," / ",labels_flat[i])
299                   if (pred_flat[i] == labels_flat[i]) and (
                          labels_flat[i] == 7):
300                     match_num += 1
301                   if labels_flat[i] == 7:
302                     func_num += 1
303                 if match_num == 0 or func_num == 0:
304                   return 0
305                 else:
306                   return match_num / func_num
307
308               model.zero_grad()
309
```

```
310             final_info = {}
311
312             for epoch_i in range(0, epochs):
313
314                 print("")
315                 print('======== Epoch {:} / {:} ========'.
                        format(epoch_i + 1, epochs))
316                 print('Training...')
317
318                 t0 = time.time()
319                 total_loss = 0
320                 model.train()
321
322                 for step, batch in enumerate(train_dataloader):
323                     if step % 500 == 0 and not step == 0:
324                         elapsed = format_time(time.time() - t0)
325                         print('  Batch {:>5,}  of  {:>5,}.
                            Elapsed: {:}.'.format(step, len(
                            train_dataloader), elapsed))
326
327                     batch = tuple(t.to(device) for t in batch)
328                     b_input_ids, b_input_mask, b_labels = batch
329                     outputs = model(b_input_ids,
330                                     token_type_ids=None,
331                                     attention_mask=b_input_mask
                                        ,
332                                     labels=b_labels)
333
334                     loss = outputs[0]
335                     total_loss += loss.item()
336                     loss.backward()
```

```
337
338                     torch.nn.utils.clip_grad_norm_(model.
                            parameters(), 1.0)
339                     optimizer.step()
340                     scheduler.step()
341                     model.zero_grad()
342
343                 avg_train_loss = total_loss / len(
                        train_dataloader)
344
345                 print("")
346                 print("  Average training loss: {0:.2f}".format
                        (avg_train_loss))
347                 print("  Training epcoh took: {:}".format(
                        format_time(time.time() - t0)))
348
349                 print("")
350                 print("Running Validation...")
351
352                 t0 = time.time()
353
354                 model.eval()
355
356                 eval_loss, eval_accuracy = 0, 0
357                 nb_eval_steps, nb_eval_examples = 0, 0
358                 FNS_nb_eval_steps, FNS_eval_accuracy = 0, 0
359                 INS_nb_eval_steps, INS_eval_accuracy = 0, 0
360                 GOL_nb_eval_steps, GOL_eval_accuracy = 0, 0
361                 EFF_nb_eval_steps, EFF_eval_accuracy = 0, 0
362                 CRT_nb_eval_steps, CRT_eval_accuracy = 0, 0
363                 LOC_nb_eval_steps, LOC_eval_accuracy = 0, 0
```

```
364                 AGT_nb_eval_steps, AGT_eval_accuracy = 0, 0
365                 THM_nb_eval_steps, THM_eval_accuracy = 0, 0
366
367                 epoch_info = {}
368
369                 for batch in test_dataloader:
370                     batch = tuple(t.to(device) for t in batch)
371                     b_input_ids, b_input_mask, b_labels = batch
372                     with torch.no_grad():
373                         outputs = model(b_input_ids,
374                                         token_type_ids=None,
375                                         attention_mask=
                                            b_input_mask)
376
377                     logits = outputs[0]
378                     logits = logits.detach().cpu().numpy()
379                     label_ids = b_labels.to('cpu').numpy()
380
381                     tmp_eval_accuracy = flat_accuracy(logits,
                        label_ids)
382                     eval_accuracy += tmp_eval_accuracy
383                     nb_eval_steps += 1
384
385                     FNS_tmp_eval_accuracy = FNS_flat_accuracy(
                        logits, label_ids)
386                     FNS_eval_accuracy += FNS_tmp_eval_accuracy
387                     FNS_nb_eval_steps += 1
388
389                     INS_tmp_eval_accuracy = INS_flat_accuracy(
                        logits, label_ids)
390                     INS_eval_accuracy += INS_tmp_eval_accuracy
```

```
391                     INS_nb_eval_steps += 1
392
393                     GOL_tmp_eval_accuracy = GOL_flat_accuracy(
                            logits, label_ids)
394                     GOL_eval_accuracy += GOL_tmp_eval_accuracy
395                     GOL_nb_eval_steps += 1
396
397                     EFF_tmp_eval_accuracy = EFF_flat_accuracy(
                            logits, label_ids)
398                     EFF_eval_accuracy += EFF_tmp_eval_accuracy
399                     EFF_nb_eval_steps += 1
400
401                     CRT_tmp_eval_accuracy = CRT_flat_accuracy(
                            logits, label_ids)
402                     CRT_eval_accuracy += CRT_tmp_eval_accuracy
403                     CRT_nb_eval_steps += 1
404
405                     LOC_tmp_eval_accuracy = LOC_flat_accuracy(
                            logits, label_ids)
406                     LOC_eval_accuracy += LOC_tmp_eval_accuracy
407                     LOC_nb_eval_steps += 1
408
409                     AGT_tmp_eval_accuracy = AGT_flat_accuracy(
                            logits, label_ids)
410                     AGT_eval_accuracy += AGT_tmp_eval_accuracy
411                     AGT_nb_eval_steps += 1
412
413                     THM_tmp_eval_accuracy = THM_flat_accuracy(
                            logits, label_ids)
414                     THM_eval_accuracy += THM_tmp_eval_accuracy
415                     THM_nb_eval_steps += 1
```

```
416
417                 print("  Accuracy: {0:.2f}".format(
                        eval_accuracy/nb_eval_steps))
418                 print("  Validation took: {:}".format(
                        format_time(time.time() - t0)))
419                 print("")
420                 print("  Detail accuracy  ")
421                 print("  FNS_Accuracy: {0:.2f}".format(
                        FNS_eval_accuracy/FNS_nb_eval_steps))
422                 print("  INS_Accuracy: {0:.2f}".format(
                        INS_eval_accuracy/INS_nb_eval_steps))
423                 print("  GOL_Accuracy: {0:.2f}".format(
                        GOL_eval_accuracy/GOL_nb_eval_steps))
424                 print("  EFF_Accuracy: {0:.2f}".format(
                        EFF_eval_accuracy/EFF_nb_eval_steps))
425                 print("  CRT_Accuracy: {0:.2f}".format(
                        CRT_eval_accuracy/CRT_nb_eval_steps))
426                 print("  LOC_Accuracy: {0:.2f}".format(
                        LOC_eval_accuracy/LOC_nb_eval_steps))
427                 print("  AGT_Accuracy: {0:.2f}".format(
                        AGT_eval_accuracy/AGT_nb_eval_steps))
428                 print("  THM_Accuracy: {0:.2f}".format(
                        THM_eval_accuracy/THM_nb_eval_steps))
429
430                 epoch_info["Total"] = round(eval_accuracy/
                        nb_eval_steps,3)
431                 epoch_info["Loss"] = round(avg_train_loss,3)
432                 epoch_info["FNS"] = round(FNS_eval_accuracy/
                        FNS_nb_eval_steps,3)
433                 epoch_info["INS"] = round(INS_eval_accuracy/
                        INS_nb_eval_steps,3)
```

```
434                 epoch_info["GOL"] = round(GOL_eval_accuracy/
                        GOL_nb_eval_steps,3)
435                 epoch_info["EFF"] = round(EFF_eval_accuracy/
                        EFF_nb_eval_steps,3)
436                 epoch_info["CRT"] = round(CRT_eval_accuracy/
                        CRT_nb_eval_steps,3)
437                 epoch_info["LOC"] = round(LOC_eval_accuracy/
                        LOC_nb_eval_steps,3)
438                 epoch_info["AGT"] = round(AGT_eval_accuracy/
                        AGT_nb_eval_steps,3)
439                 epoch_info["THM"] = round(THM_eval_accuracy/
                        THM_nb_eval_steps,3)
440
441                 final_info["epoch"+str(epoch_i)] = epoch_info
442
443                 model.eval()
444                 test_input_ids = []
445                 test_input_mask = []
446                 test_labels = []
447
448                 num = 0
449                 for step, batch in enumerate(test_data):
450                   batch = tuple(t.to(device) for t in batch)
451
452                   b_input_ids, b_input_mask, b_labels = batch
453                   input_ids_arr = []
454                   input_mask_arr = []
455
456                   for i in range(0,len(b_input_ids)):
457                     input_ids_arr.append(int(b_input_ids[i]))
458                     input_mask_arr.append(int(b_input_mask[i]))
```

```
459
460                 test_input_ids.append(input_ids_arr)
461                 test_input_mask.append(input_mask_arr)
462                 test_labels.append(int(b_labels))
463
464             test_input_ids = torch.tensor(test_input_ids)
465             test_input_mask = torch.tensor(test_input_mask)
466             test_labels = test_labels
467
468             test_input_ids = test_input_ids.to(device)
469             test_input_mask = test_input_mask.to(device)
470
471             with torch.no_grad():
472                 outputs = model(test_input_ids,
473                                 token_type_ids=None,
474                                 attention_mask=
                                     test_input_mask)
475
476             sentence_vecs_sum = outputs[0]
477
478             sentence_array = []
479             for i in range(0,len(sentence_vecs_sum)):
480               each_array = []
481               for j in range(0,len(sentence_vecs_sum[i])):
482                 each_array.append(float(sentence_vecs_sum[i
                      ][j]))
483               sentence_array.append(each_array)
484
485             initial_df = pd.DataFrame(sentence_array)
486
487             from sklearn.manifold import TSNE
```

```
488                 tsne = TSNE(n_components=2, random_state=0)
489                 tsne_obj= tsne.fit_transform(initial_df)
490
491                 tsne_df = pd.DataFrame({'X':tsne_obj[:,0],'Y':
                        tsne_obj[:,1],'Label':test_labels})
492
493                 import numpy as np
494                 import pandas as pd
495                 from plotnine import *
496
497                 print("")
498                 print("  Network visualization  ")
499                 print(ggplot(tsne_df, aes(x='X', y='Y')) +
                        geom_point(aes(colour = 'Label')))
500
501             print("")
502             print("Training complete!")
503             print("")
504             print("Final result is below!")
505             print(final_info)
506
507         elif self.postposition == "Eyse":
508
509             def flat_accuracy(preds, labels):
510                 pred_flat = np.argmax(preds, axis=1).flatten()
511                 labels_flat = labels.flatten()
512                 return np.sum(pred_flat == labels_flat) / len(
                        labels_flat)
513
514             def SRC_flat_accuracy(preds, labels):
515                 pred_flat = np.argmax(preds, axis=1).flatten()
```

```
516                 labels_flat = labels.flatten()
517                 match_num = 0
518                 func_num = 0
519                 for i in range(0,len(pred_flat)):
520                   if (pred_flat[i] == labels_flat[i]) and (
                        labels_flat[i] == 0):
521                     match_num += 1
522                   if labels_flat[i] == 0:
523                     func_num += 1
524                 if match_num == 0 or func_num == 0:
525                   return 0
526                 else:
527                   return match_num / func_num
528
529             def LOC_flat_accuracy(preds, labels):
530                 pred_flat = np.argmax(preds, axis=1).flatten()
531                 labels_flat = labels.flatten()
532                 match_num = 0
533                 func_num = 0
534                 for i in range(0,len(pred_flat)):
535                   if (pred_flat[i] == labels_flat[i]) and (
                        labels_flat[i] == 1):
536                     match_num += 1
537                   if labels_flat[i] == 1:
538                     func_num += 1
539                 if match_num == 0 or func_num == 0:
540                   return 0
541                 else:
542                   return match_num / func_num
543
544             model.zero_grad()
```

```
545             final_info = {}
546
547             for epoch_i in range(0, epochs):
548
549                 print("")
550                 print('======== Epoch {:} / {:} ========'.
                        format(epoch_i + 1, epochs))
551                 print('Training...')
552
553                 t0 = time.time()
554                 total_loss = 0
555                 model.train()
556
557                 for step, batch in enumerate(train_dataloader):
558                     if step % 500 == 0 and not step == 0:
559                         elapsed = format_time(time.time() - t0)
560                         print('  Batch {:>5,}  of  {:>5,}.
                                Elapsed: {:}.'.format(step, len(
                                train_dataloader), elapsed))
561
562                     batch = tuple(t.to(device) for t in batch)
563                     b_input_ids, b_input_mask, b_labels = batch
564                     outputs = model(b_input_ids,
565                                     token_type_ids=None,
566                                     attention_mask=b_input_mask
                                        ,
567                                     labels=b_labels)
568
569                     loss = outputs[0]
570                     total_loss += loss.item()
571                     loss.backward()
```

```
572                     torch.nn.utils.clip_grad_norm_(model.
                            parameters(), 1.0)
573                     optimizer.step()
574                     scheduler.step()
575                     model.zero_grad()
576
577                 avg_train_loss = total_loss / len(
                        train_dataloader)
578
579                 print("")
580                 print("  Average training loss: {0:.2f}".format
                        (avg_train_loss))
581                 print("  Training epcoh took: {:}".format(
                        format_time(time.time() - t0)))
582
583                 print("")
584                 print("Running Validation...")
585
586                 t0 = time.time()
587                 model.eval()
588
589                 eval_loss, eval_accuracy = 0, 0
590                 nb_eval_steps, nb_eval_examples = 0, 0
591                 SRC_nb_eval_steps, SRC_eval_accuracy = 0, 0
592                 LOC_nb_eval_steps, LOC_eval_accuracy = 0, 0
593
594                 epoch_info = {}
595
596                 for batch in test_dataloader:
597                     batch = tuple(t.to(device) for t in batch)
598                     b_input_ids, b_input_mask, b_labels = batch
```

```
599                 with torch.no_grad():
600                     outputs = model(b_input_ids,
601                                     token_type_ids=None,
602                                     attention_mask=
                                        b_input_mask)
603
604                 logits = outputs[0]
605                 logits = logits.detach().cpu().numpy()
606                 label_ids = b_labels.to('cpu').numpy()
607
608                 tmp_eval_accuracy = flat_accuracy(logits,
                        label_ids)
609                 eval_accuracy += tmp_eval_accuracy
610                 nb_eval_steps += 1
611
612                 SRC_tmp_eval_accuracy = SRC_flat_accuracy(
                        logits, label_ids)
613                 SRC_eval_accuracy += SRC_tmp_eval_accuracy
614                 SRC_nb_eval_steps += 1
615
616                 LOC_tmp_eval_accuracy = LOC_flat_accuracy(
                        logits, label_ids)
617                 LOC_eval_accuracy += LOC_tmp_eval_accuracy
618                 LOC_nb_eval_steps += 1
619
620             print("  Accuracy: {0:.2f}".format(
                    eval_accuracy/nb_eval_steps))
621             print("  Validation took: {:}".format(
                    format_time(time.time() - t0)))
622             print("")
623             print("  Detail accuracy  ")
```

```
624                 print("  SRC_Accuracy: {0:.2f}".format(
                        SRC_eval_accuracy/SRC_nb_eval_steps))
625                 print("  LOC_Accuracy: {0:.2f}".format(
                        LOC_eval_accuracy/LOC_nb_eval_steps))
626
627                 epoch_info["Total"] = round(eval_accuracy/
                        nb_eval_steps,3)
628                 epoch_info["Loss"] = round(avg_train_loss,3)
629                 epoch_info["SRC"] = round(SRC_eval_accuracy/
                        SRC_nb_eval_steps,3)
630                 epoch_info["LOC"] = round(LOC_eval_accuracy/
                        LOC_nb_eval_steps,3)
631
632                 final_info["epoch"+str(epoch_i)] = epoch_info
633
634                 model.eval()
635                 test_input_ids = []
636                 test_input_mask = []
637                 test_labels = []
638
639                 num = 0
640                 for step, batch in enumerate(test_data):
641                   batch = tuple(t.to(device) for t in batch)
642
643                   b_input_ids, b_input_mask, b_labels = batch
644                   input_ids_arr = []
645                   input_mask_arr = []
646
647                   for i in range(0,len(b_input_ids)):
648                     input_ids_arr.append(int(b_input_ids[i]))
649                     input_mask_arr.append(int(b_input_mask[i]))
```

```
650
651                 test_input_ids.append(input_ids_arr)
652                 test_input_mask.append(input_mask_arr)
653                 test_labels.append(int(b_labels))
654
655             test_input_ids = torch.tensor(test_input_ids)
656             test_input_mask = torch.tensor(test_input_mask)
657             test_labels = test_labels
658
659             test_input_ids = test_input_ids.to(device)
660             test_input_mask = test_input_mask.to(device)
661
662             with torch.no_grad():
663                 outputs = model(test_input_ids,
664                                 token_type_ids=None,
665                                 attention_mask=
                                     test_input_mask)
666
667             sentence_vecs_sum = outputs[0]
668
669             sentence_array = []
670             for i in range(0,len(sentence_vecs_sum)):
671               each_array = []
672               for j in range(0,len(sentence_vecs_sum[i])):
673                 each_array.append(float(sentence_vecs_sum[i
                      ][j]))
674               sentence_array.append(each_array)
675
676             initial_df = pd.DataFrame(sentence_array)
677
678             from sklearn.manifold import TSNE
```

```
679                 tsne = TSNE(n_components=2, random_state=0)
680                 tsne_obj= tsne.fit_transform(initial_df)
681
682                 tsne_df = pd.DataFrame({'X':tsne_obj[:,0],'Y':
                        tsne_obj[:,1],'Label':test_labels})
683
684                 import numpy as np
685                 import pandas as pd
686                 from plotnine import *
687
688                 print("")
689                 print("  Network visualization  ")
690                 print(ggplot(tsne_df, aes(x='X', y='Y')) +
                        geom_point(aes(colour = 'Label')))
691
692             print("")
693             print("Training complete!")
694             print("")
695             print("Final result is below!")
696             print(final_info)
697
698         elif self.postposition == "Lo":
699
700             def flat_accuracy(preds, labels):
701                 pred_flat = np.argmax(preds, axis=1).flatten()
702                 labels_flat = labels.flatten()
703                 return np.sum(pred_flat == labels_flat) / len(
                        labels_flat)
704
705             def FNS_flat_accuracy(preds, labels):
706                 pred_flat = np.argmax(preds, axis=1).flatten()
```

```
707             labels_flat = labels.flatten()
708             match_num = 0
709             func_num = 0
710             for i in range(0,len(pred_flat)):
711               if (pred_flat[i] == labels_flat[i]) and (
                      labels_flat[i] == 0):
712                 match_num += 1
713               if labels_flat[i] == 0:
714                 func_num += 1
715             if match_num == 0 or func_num == 0:
716               return 0
717             else:
718               return match_num / func_num
719
720         def INS_flat_accuracy(preds, labels):
721             pred_flat = np.argmax(preds, axis=1).flatten()
722             labels_flat = labels.flatten()
723             match_num = 0
724             func_num = 0
725             for i in range(0,len(pred_flat)):
726               if (pred_flat[i] == labels_flat[i]) and (
                      labels_flat[i] == 1):
727                 match_num += 1
728               if labels_flat[i] == 1:
729                 func_num += 1
730             if match_num == 0 or func_num == 0:
731               return 0
732             else:
733               return match_num / func_num
734
735         def DIR_flat_accuracy(preds, labels):
```

```
736             pred_flat = np.argmax(preds, axis=1).flatten()
737             labels_flat = labels.flatten()
738             match_num = 0
739             func_num = 0
740             for i in range(0,len(pred_flat)):
741               if (pred_flat[i] == labels_flat[i]) and (
                      labels_flat[i] == 2):
742                 match_num += 1
743               if labels_flat[i] == 2:
744                 func_num += 1
745             if match_num == 0 or func_num == 0:
746               return 0
747             else:
748               return match_num / func_num
749
750         def EFF_flat_accuracy(preds, labels):
751             pred_flat = np.argmax(preds, axis=1).flatten()
752             labels_flat = labels.flatten()
753             match_num = 0
754             func_num = 0
755             for i in range(0,len(pred_flat)):
756               if (pred_flat[i] == labels_flat[i]) and (
                      labels_flat[i] == 3):
757                 match_num += 1
758               if labels_flat[i] == 3:
759                 func_num += 1
760             if match_num == 0 or func_num == 0:
761               return 0
762             else:
763               return match_num / func_num
764
```

```
765             def CRT_flat_accuracy(preds, labels):
766                 pred_flat = np.argmax(preds, axis=1).flatten()
767                 labels_flat = labels.flatten()
768                 match_num = 0
769                 func_num = 0
770                 for i in range(0,len(pred_flat)):
771                   if (pred_flat[i] == labels_flat[i]) and (
                          labels_flat[i] == 4):
772                     match_num += 1
773                   if labels_flat[i] == 4:
774                     func_num += 1
775                 if match_num == 0 or func_num == 0:
776                   return 0
777                 else:
778                   return match_num / func_num
779
780             def LOC_flat_accuracy(preds, labels):
781                 pred_flat = np.argmax(preds, axis=1).flatten()
782                 labels_flat = labels.flatten()
783                 match_num = 0
784                 func_num = 0
785                 for i in range(0,len(pred_flat)):
786                   if (pred_flat[i] == labels_flat[i]) and (
                          labels_flat[i] == 5):
787                     match_num += 1
788                   if labels_flat[i] == 5:
789                     func_num += 1
790                 if match_num == 0 or func_num == 0:
791                   return 0
792                 else:
793                   return match_num / func_num
```

```
794
795             model.zero_grad()
796             final_info = {}
797
798             for epoch_i in range(0, epochs):
799
800                 print("")
801                 print('======== Epoch {:} / {:} ========'.
                        format(epoch_i + 1, epochs))
802                 print('Training...')
803
804                 t0 = time.time()
805                 total_loss = 0
806                 model.train()
807
808                 for step, batch in enumerate(train_dataloader):
809                     if step % 500 == 0 and not step == 0:
810                         elapsed = format_time(time.time() - t0)
811                         print('  Batch {:>5,}  of  {:>5,}.
                                Elapsed: {:}.'.format(step, len(
                                train_dataloader), elapsed))
812
813                     batch = tuple(t.to(device) for t in batch)
814                     b_input_ids, b_input_mask, b_labels = batch
815                     outputs = model(b_input_ids,
816                                     token_type_ids=None,
817                                     attention_mask=b_input_mask
                                        ,
818                                     labels=b_labels)
819
820                     loss = outputs[0]
```

```
821                     total_loss += loss.item()
822                     loss.backward()
823                     torch.nn.utils.clip_grad_norm_(model.
                            parameters(), 1.0)
824                     optimizer.step()
825                     scheduler.step()
826                     model.zero_grad()
827
828                 avg_train_loss = total_loss / len(
                        train_dataloader)
829
830                 print("")
831                 print("  Average training loss: {0:.2f}".format
                        (avg_train_loss))
832                 print("  Training epcoh took: {:}".format(
                        format_time(time.time() - t0)))
833
834                 print("")
835                 print("Running Validation...")
836
837                 t0 = time.time()
838                 model.eval()
839
840                 eval_loss, eval_accuracy = 0, 0
841                 nb_eval_steps, nb_eval_examples = 0, 0
842                 FNS_nb_eval_steps, FNS_eval_accuracy = 0, 0
843                 INS_nb_eval_steps, INS_eval_accuracy = 0, 0
844                 DIR_nb_eval_steps, DIR_eval_accuracy = 0, 0
845                 EFF_nb_eval_steps, EFF_eval_accuracy = 0, 0
846                 CRT_nb_eval_steps, CRT_eval_accuracy = 0, 0
847                 LOC_nb_eval_steps, LOC_eval_accuracy = 0, 0
```

```
848
849                 epoch_info = {}
850
851                 for batch in test_dataloader:
852                     batch = tuple(t.to(device) for t in batch)
853                     b_input_ids, b_input_mask, b_labels = batch
854                     with torch.no_grad():
855                         outputs = model(b_input_ids,
856                                         token_type_ids=None,
857                                         attention_mask=
                                            b_input_mask)
858
859                     logits = outputs[0]
860
861                     logits = logits.detach().cpu().numpy()
862                     label_ids = b_labels.to('cpu').numpy()
863
864                     tmp_eval_accuracy = flat_accuracy(logits,
                        label_ids)
865                     eval_accuracy += tmp_eval_accuracy
866                     nb_eval_steps += 1
867
868                     FNS_tmp_eval_accuracy = FNS_flat_accuracy(
                        logits, label_ids)
869                     FNS_eval_accuracy += FNS_tmp_eval_accuracy
870                     FNS_nb_eval_steps += 1
871
872                     INS_tmp_eval_accuracy = INS_flat_accuracy(
                        logits, label_ids)
873                     INS_eval_accuracy += INS_tmp_eval_accuracy
874                     INS_nb_eval_steps += 1
```

```
875
876                     DIR_tmp_eval_accuracy = DIR_flat_accuracy(
                            logits, label_ids)
877                     DIR_eval_accuracy += DIR_tmp_eval_accuracy
878                     DIR_nb_eval_steps += 1
879
880                     EFF_tmp_eval_accuracy = EFF_flat_accuracy(
                            logits, label_ids)
881                     EFF_eval_accuracy += EFF_tmp_eval_accuracy
882                     EFF_nb_eval_steps += 1
883
884                     CRT_tmp_eval_accuracy = CRT_flat_accuracy(
                            logits, label_ids)
885                     CRT_eval_accuracy += CRT_tmp_eval_accuracy
886                     CRT_nb_eval_steps += 1
887
888                     LOC_tmp_eval_accuracy = LOC_flat_accuracy(
                            logits, label_ids)
889                     LOC_eval_accuracy += LOC_tmp_eval_accuracy
890                     LOC_nb_eval_steps += 1
891
892                 print("  Accuracy: {0:.2f}".format(
                        eval_accuracy/nb_eval_steps))
893                 print("  Validation took: {:}".format(
                        format_time(time.time() - t0)))
894                 print("")
895                 print("  Detail accuracy  ")
896                 print("  FNS_Accuracy: {0:.2f}".format(
                        FNS_eval_accuracy/FNS_nb_eval_steps))
897                 print("  INS_Accuracy: {0:.2f}".format(
                        INS_eval_accuracy/INS_nb_eval_steps))
```

```
898                 print("  DIR_Accuracy: {0:.2f}".format(
                        DIR_eval_accuracy/DIR_nb_eval_steps))
899                 print("  EFF_Accuracy: {0:.2f}".format(
                        EFF_eval_accuracy/EFF_nb_eval_steps))
900                 print("  CRT_Accuracy: {0:.2f}".format(
                        CRT_eval_accuracy/CRT_nb_eval_steps))
901                 print("  LOC_Accuracy: {0:.2f}".format(
                        LOC_eval_accuracy/LOC_nb_eval_steps))
902
903                 epoch_info["Total"] = round(eval_accuracy/
                        nb_eval_steps,3)
904                 epoch_info["Loss"] = round(avg_train_loss,3)
905                 epoch_info["FNS"] = round(FNS_eval_accuracy/
                        FNS_nb_eval_steps,3)
906                 epoch_info["INS"] = round(INS_eval_accuracy/
                        INS_nb_eval_steps,3)
907                 epoch_info["DIR"] = round(DIR_eval_accuracy/
                        DIR_nb_eval_steps,3)
908                 epoch_info["EFF"] = round(EFF_eval_accuracy/
                        EFF_nb_eval_steps,3)
909                 epoch_info["CRT"] = round(CRT_eval_accuracy/
                        CRT_nb_eval_steps,3)
910                 epoch_info["LOC"] = round(LOC_eval_accuracy/
                        LOC_nb_eval_steps,3)
911
912                 final_info["epoch"+str(epoch_i)] = epoch_info
913
914                 model.eval()
915                 test_input_ids = []
916                 test_input_mask = []
917                 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 = []
```

```
948                 for i in range(0,len(sentence_vecs_sum)):
949                   each_array = []
950                   for j in range(0,len(sentence_vecs_sum[i])):
951                     each_array.append(float(sentence_vecs_sum[i
                            ][j]))
952                   sentence_array.append(each_array)
953                 initial_df = pd.DataFrame(sentence_array)
954                 from sklearn.manifold import TSNE
955                 tsne = TSNE(n_components=2, random_state=0)
956                 tsne_obj= tsne.fit_transform(initial_df)
957                 tsne_df = pd.DataFrame({'X':tsne_obj[:,0],'Y':
                        tsne_obj[:,1],'Label':test_labels})
958
959                 import numpy as np
960                 import pandas as pd
961                 from plotnine import *
962                 print("")
963                 print("  Network visualization  ")
964                 print(ggplot(tsne_df, aes(x='X', y='Y')) +
                        geom_point(aes(colour = 'Label')))
965
966           print("")
967           print("Training complete!")
968           print("")
969           print("Final result is below!")
970           print(final_info)
971
972       model.save_pretrained('drive/My Drive/BERT/SM/KoBERT/
              Postposition/Model/')
973       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

```
1
2 <!DOCTYPE html>
3 <html>
4     <head>
5     <title>PostEmbedding</title><!--<link rel="stylesheet"
          href="./stylesheets/bubble_style.css">-->
6     <meta http-equiv="Content-Type" content="text/html;
          charset=utf-8">
7     <script src="./javascripts/d3.v3.min.js" charset="utf-8">
          </script>
8     <script src="./javascripts/d3.v4.js" charset="utf-8">
          </script>
9     <script src="./javascripts/jquery-1.12.0.min.js" charset="
          utf-8"></script>
```

```
10      <link rel="stylesheet" href="https://
            maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
            bootstrap.min.css">
11      <script src="https://maxcdn.bootstrapcdn.com/bootstrap
            /3.3.7/js/bootstrap.min.js"></script>
12      <!--<link rel="stylesheet" href="./stylesheets/PostVis.css
            ">-->
13
14      <script src="./Data/concordancedata.js" charset="utf-8">
            </script>
15      <script src="./Data/DSMsdata.js" charset="utf-8"></script>
16      <script src="./Data/Networkdata.js" charset="utf-8">
            </script>
17
18      <style>
19      body {
20          margin: 0;
21      }
22
23      #header{
24          position:relative;
25          width:100%;
26          height:50px;
27          background-color:#00cec9;
28          background-clip: content-box;
29      }
30
31      #left {
32          position:relative;
33          float:left;
34          overflow:hidden;
```

```
35          width:14%;
36          height:920px;
37          padding: 0.5%;
38          background-color:whitesmoke;
39          background-clip: content-box;
40      }
41
42      #left_top {
43          position:relative;
44          float:left;
45          overflow:hidden;
46          width:100%;
47          height:475px;
48          padding: 0.5%;
49          background-clip: content-box;
50      }
51
52      #left_bottom {
53          position:relative;
54          float:left;
55          overflow:hidden;
56          width:100%;
57          height:435px;
58          padding: 0.5%;
59          background-clip: content-box;
60      }
61
62
63      #section {
64          position:relative;
65          float:left;
```

```
66          overflow:hidden;
67          width:67%;
68          height:920px;
69          padding: 0.5%;
70          background-clip: content-box;
71      }
72
73      #section_top {
74          position:relative;
75          float:left;
76          overflow:hidden;
77          width:100%;
78          height:620px;
79          padding-right: : 0.5%;
80          padding-top: 0.5%;
81          padding-left: 0.5%;
82          background-clip: content-box;
83      }
84
85      #section_bottom {
86          position:relative;
87          float:left;
88          overflow:hidden;
89          width:100%;
90          height:295px;
91          padding-right: : 0.5%;
92          padding-bottom: 0.5%;
93          padding-left: 0.5%;
94          background-clip: content-box;
95      }
96
```

```
97      #right {
98          position:relative;
99          float:right;
100         overflow:hidden;
101         width:19%;
102         height:920px;
103         padding: 0.5%;
104         background-color:whitesmoke;
105         background-clip: content-box;
106     }
107
108     #right_top {
109         position:relative;
110         float:left;
111         overflow:hidden;
112         width:100%;
113         height:405px;
114         padding: 0.5%;
115         background-clip: content-box;
116     }
117
118
119     #right_bottom {
120         position:relative;
121         float:left;
122         overflow:hidden;
123         width:100%;
124         height:505px;
125         padding: 0.5%;
126         background-clip: content-box;
127     }
```

```
128
129     p#left_option {
130         text-align: left;
131         font-family: Open Sans;
132         font-size: 1.5em;
133         color: #666666;
134         font-weight: bold;
135         padding-top: 4%;
136         padding-bottom: 4%;
137         padding-left: 5%;
138         margin: 0;
139     }
140
141     p#right_option {
142         text-align: left;
143         font-family: Open Sans;
144         font-size: 1.6em;
145         color: #666666;
146         font-weight: bold;
147         padding-top: 4%;
148         padding-bottom: 4%;
149         padding-left: 5%;
150         margin: 0;
151     }
152
153     p#section_top_p {
154         font-family: Open Sans;
155         font-size: 1.5em;
156         color: #666666;
157         font-weight: bold;
158         padding-left: 1%;
```

```
159         margin: 0;
160     }
161 
162     p#section_bottom_p {
163         font-family: Open Sans;
164         font-size: 1.5em;
165         color: #666666;
166         font-weight: bold;
167         padding-left: 1%;
168         margin: 0;
169     }
170 
171 
172     p#header_p_left {
173         position: relative;
174         text-align: left;
175         font-family: Open Sans;
176         font-size: 2em;
177         color: white;
178         font-weight: bold;
179         padding-top: 0.5%;
180         padding-bottom: 0.5%;
181         padding-left: 1%;
182         margin: 0;
183     }
184 
185     p#header_p_right {
186         position: relative;
187         text-align: right;
188         font-family: Open Sans;
189         font-size: 2em;
```

```
190         color: white;
191         font-weight: bold;
192         padding-top: 0.5%;
193         padding-bottom: 0.5%;
194         padding-right: 2%;
195         margin: 0;
196     }
197
198     p#footer_p{
199         font-size: 1em;
200     }
201
202     a#header_a{
203         font-family: Open Sans;
204         color: white;
205         font-weight: bold;
206         cursor: pointer;
207     }
208
209     a#footer_a{
210         font-family: Open Sans;
211         color: #f3c623;
212         font-weight: bold;
213         cursor: pointer;
214     }
215
216     #header_left{
217         float:left;
218         width:49%;
219         height:50px;
220         padding-top: 0.1%;
```

```
221         background-clip: content-box;
222     }
223 
224     #header_right{
225         float:right;
226         width:49%;
227         height:50px;
228         padding-top: 0.1%;
229         background-clip: content-box;
230     }
231 
232     #footer {
233         height:20px;
234         text-align: center;
235         color: white;
236         background-color:#717171;
237         clear:both;
238     }
239 
240     select#op_postposition { /*text-align-last:center;*/
241        width: 90%;
242        height: 30px;
243        font-size: 17px;
244        border-radius: 3px;
245        position: relative;
246        left:5%;
247        background: white;
248        cursor: pointer;
249     }
250 
251     select#op_function { /*text-align-last:center;*/
```

```
252         width: 90%;
253         height: 30px;
254         font-size: 17px;
255         border-radius: 3px;
256         position: relative;
257         left:5%;
258         background: white;
259         cursor: pointer;
260     }
261
262     select#op_method { /*text-align-last:center;*/
263         width: 90%;
264         height: 30px;
265         font-size: 17px;
266         border-radius: 3px;
267         position: relative;
268         left:5%;
269         background: white;
270         cursor: pointer;
271     }
272
273     select#op_window {
274         width: 90%;
275         height: 30px;
276         font-size: 17px;
277         border-radius: 3px;
278         position: relative;
279         left:5%;
280         background: white;
281         cursor: pointer;
282     }
```

```
283
284     input#onoff {
285       cursor: pointer;
286       position: relative;
287       font-size: 14px;
288       left:8%;
289     }
290
291
292     select#op_node_size {
293        width: 90%;
294        height: 30px;
295        font-size: 17px;
296        border-radius: 3px;
297        position: relative;
298        left:5%;
299        background: white;
300        cursor: pointer;
301     }
302
303     select#op_node_color {
304        width: 90%;
305        height: 30px;
306        font-size: 17px;
307        border-radius: 3px;
308        position: relative;
309        left:5%;
310        background: white;
311        cursor: pointer;
312     }
313
```

```
314
315     #container_leftbottom {
316         border:2px solid #ccc;
317         width:88%;
318         height: 360px;
319         position: absolute;
320         left: 5%;
321         overflow-y: scroll;
322         overflow-x: auto;
323         white-space: nowrap;
324         border-radius: 10px;
325         background: white;
326     }
327
328     #container_rightbottom {
329         border:2px solid #ccc;
330         width:92.5%;
331         height: 430px;
332         position: absolute;
333         left: 5%;
334         overflow-y: scroll;
335         overflow-x: auto;
336         white-space: nowrap;
337         border-radius: 10px;
338         background: white;
339     }
340
341     #container_section_bottom {
342         border:2px solid #ccc;
343         width:98%;
344         height: 245px;
```

```
345         position: absolute;
346         left: 1%;
347         overflow-y: scroll;
348         overflow-x: auto;
349         white-space: nowrap;
350         border-radius: 8px;
351         background: white;
352     }
353
354     .CB_leftbottom{
355         cursor: pointer;
356         position: relative;
357         font-size: 14px;
358         left:5%;
359     }
360
361     .networklinks line {
362       stroke: #999;
363       stroke-opacity: 0.6;
364     }
365
366     .networknodes circle {
367       stroke: #666666;
368       stroke-width: 1.5px;
369     }
370
371     table {
372         font-size: 15px;
373         max-height: 200px;
374         overflow: auto;
375         background: #ddd;
```

```
376         box-shadow: 0 0 1px 1px #ddd;
377     }
378
379     th, td {
380         background: #fff;
381         padding: 8px 16px;
382         padding-bottom: 0px;
383     }
384
385     thead th {
386         background-color: #ddd;
387         position: sticky;
388         top: 0;
389     }
390
391     div.tooltip {
392         position: absolute;
393         text-align: left;
394         padding: 5px;
395         font-size: 17px;
396         background-color: #efefef;
397         border: solid 1px #cecece;
398         border-radius: 8px;
399         box-shadow: 0 3px 5px 0 #dfdfdf;
400         pointer-events: none;
401     }
402
403     h5 {
404         font-size: 15px;
405     }
406
```

```
407     g.networknodes text {
408         font-size: 13px;
409     }
410
411     text#nodegroup.nodetext {
412         font-size: 13px;
413     }
414
415     @media all and (min-width:951px) and (max-height: 1000px)
            { /*0.95*/
416         #header{
417             height:47.55px;
418         }
419         #header_left{
420             height:47.55px;
421         }
422
423         #header_right{
424             height:47.55px;
425         }
426         #left {
427             height:874.00px;
428         }
429         #left_top {
430             height:460.75px;
431         }
432         #left_bottom {
433             height:403.75px;
434         }
435         p#left_option {
436             font-size: 1.42em;
```

```
437         }
438         p#right_option {
439             font-size: 1.52em;
440         }
441         p#section_top_p {
442             font-size: 1.42em;
443         }
444         p#section_bottom_p {
445             font-size: 1.42em;
446         }
447         p#header_p {
448             font-size: 1.90em;
449         }
450         p#header_p_left {
451             font-size: 1.9em;
452         }
453
454         p#header_p_right {
455             font-size: 1.9em;
456         }
457
458         p#footer_p{
459             font-size: 0.95em;
460         }
461         #section {
462             height:874.92px;
463         }
464         #section_top {
465             height:589.62px;
466         }
467         #section_bottom {
```

```
468             height:280.25px;
469         }
470         #right {
471             height:874.92px;
472         }
473         #right_top {
474             height:380.75px;
475         }
476         #right_bottom {
477             height:465.75px;
478         }
479         #footer {
480             height:19px;
481         }
482         select#op_postposition {
483             height: 28.53px;
484             font-size: 16.16px;
485         }
486         select#op_function {
487             height: 28.53px;
488             font-size: 16.16px;
489         }
490         select#op_method {
491             height: 28.53px;
492             font-size: 16.16px;
493         }
494         select#op_window {
495             height: 28.53px;
496             font-size: 16.16px;
497         }
498         select#op_node_size {
```

```
499             height: 28.53px;
500             font-size: 16.16px;
501         }
502         select#op_node_color {
503             height: 28.53px;
504             font-size: 16.16px;
505         }
506         input#onoff {
507             font-size: 13.30px;
508         }
509         #container_leftbottom {
510             height: 342px;
511         }
512         #container_rightbottom {
513             height: 420px;
514         }
515         #container_section_bottom {
516             height: 232.75px;
517         }
518         div.tooltip {
519             padding: 4.75px;
520             font-size: 16.16px;
521             border-radius: 7.60px;
522         }
523         .CB_leftbottom{
524             font-size: 14.26px;
525         }
526         table {
527             font-size: 14.26px;
528         }
529         img#header_img{
```

```
530             width: 33.28px;
531             height: 30.43px;
532         }
533         h5 {
534             font-size: 14.26px;
535         }
536         g.networknodes text {
537             font-size: 12.35px;
538         }
539         text#nodegroup.nodetext {
540             font-size: 12.35px;
541         }
542     }
543     @media all and (min-width:901px) and (max-height: 950px) {
        /*0.948*/
544         #header{
545             height:45.05px;
546         }
547         #header_left{
548             height:45.05px;
549         }
550
551         #header_right{
552             height:45.05px;
553         }
554         #left {
555             height:828.55px;
556         }
557         #left_top {
558             height:432.52px;
559         }
```

```
560         #left_bottom {
561             height:387.02px;
562         }
563         p#left_option {
564             font-size: 1.35em;
565         }
566         p#right_option {
567             font-size: 1.44em;
568         }
569         p#section_top_p {
570             font-size: 1.35em;
571         }
572         p#section_bottom_p {
573             font-size: 1.35em;
574         }
575         p#header_p {
576             font-size: 1.80em;
577         }
578         p#header_p_left {
579             font-size: 1.80em;
580         }
581
582         p#header_p_right {
583             font-size: 1.80em;
584         }
585         p#footer_p{
586             font-size: 0.9em;
587         }
588         #section {
589             height:828.92px;
590         }
```

```
591         #section_top {
592             height:558.62px;
593         }
594         #section_bottom {
595             height:265.677px;
596         }
597         #right {
598             height:828.92px;
599         }
600         #right_top {
601             height:367.113px;
602         }
603         #right_bottom {
604             height:432.433px;
605         }
606         #footer {
607             height:18px;
608         }
609         select#op_postposition {
610             height: 27.03px;
611             font-size: 15.31px;
612         }
613         select#op_function {
614             height: 27.03px;
615             font-size: 15.31px;
616         }
617         select#op_method {
618             height: 27.03px;
619             font-size: 15.31px;
620         }
621         select#op_window {
```

```
622             height: 27.03px;
623             font-size: 15.31px;
624         }
625         select#op_node_size {
626             height: 27.03px;
627             font-size: 15.31px;
628         }
629         select#op_node_color {
630             height: 27.03px;
631             font-size: 15.31px;
632         }
633         input#onoff {
634             font-size: 12.6px;
635         }
636 
637         #container_leftbottom {
638             height: 328.482px;
639         }
640         #container_rightbottom {
641             height: 399.288px;
642         }
643         #container_section_bottom {
644             height: 220.647px;
645         }
646         div.tooltip {
647             padding: 4.50px;
648             font-size: 15.31px;
649             border-radius: 7.21px;
650         }
651         .CB_leftbottom{
652             font-size: 13.51px;
```

```
653         }
654         table {
655             font-size: 13.51px;
656         }
657         img#header_img{
658             width: 31.53px;
659             height: 28.83px;
660         }
661         h5 {
662             font-size: 13.51px;
663         }
664         g.networknodes text {
665             font-size: 11.71px;
666         }
667         text#nodegroup.nodetext {
668             font-size: 11.71px;
669         }
670     }
671
672     @media all and (min-width:819px) and (max-height: 900px) {
        /*0.909*/
673         #header{
674             height:40.95px;
675         }
676         #header_left{
677             height:40.95px;
678         }
679
680         #header_right{
681             height:40.95px;
682         }
```

```
683         #left {
684             height:753.15px;
685         }
686         #left_top {
687             height:388.9px;
688         }
689         #left_bottom {
690             height:356.05px;
691         }
692         p#left_option {
693             font-size: 1.22em;
694         }
695         p#right_option {
696             font-size: 1.31em;
697         }
698         p#section_top_p {
699             font-size: 1.22em;
700         }
701         p#section_bottom_p {
702             font-size: 1.22715em;
703         }
704         p#header_p {
705             font-size: 1.63em;
706         }
707         p#header_p_left {
708             font-size: 1.63em;
709         }
710
711         p#header_p_right {
712             font-size: 1.63em;
713         }
```

```
714         p#footer_p{
715             font-size: 0.8em;
716         }
717         #section {
718             height:753.56px;
719         }
720         #section_top {
721             height:507.83px;
722         }
723         #section_bottom {
724             height:241.5004px;
725         }
726         #right {
727             height:753.56px;
728         }
729         #right_top {
730             height:336.0691px;
731         }
732         #right_bottom {
733             height:390.8982px;
734         }
735         #footer {
736             height:16.4px;
737         }
738         select#op_postposition {
739             height: 24.57px;
740             font-size: 13.92px;
741         }
742         select#op_function {
743             height: 24.57px;
744             font-size: 13.92px;
```

```
745         }
746         select#op_method {
747             height: 24.57px;
748             font-size: 13.92px;
749         }
750         select#op_window {
751             height: 24.57px;
752             font-size: 13.92px;
753         }
754         select#op_node_size {
755             height: 24.57px;
756             font-size: 13.92px;
757         }
758         select#op_node_color {
759             height: 24.57px;
760             font-size: 13.92px;
761         }
762         input#onoff {
763             font-size: 11.45px;
764         }
765
766         #container_leftbottom {
767             height: 302.84px;
768         }
769         #container_rightbottom {
770             height: 357.9498px;
771         }
772         #container_section_bottom {
773             height: 200.5681px;
774         }
775         div.tooltip {
```

```
776             padding: 4.09px;
777             font-size: 13.92px;
778             border-radius: 6.55px;
779         }
780         .CB_leftmiddle{
781             font-size: 12.28px;
782         }
783         .CB_leftbottom{
784             font-size: 12.28px;
785         }
786         table {
787             font-size: 12.28px;
788         }
789         img#header_img{
790             width: 28.66px;
791             height: 26.21px;
792         }
793         h5 {
794             font-size: 12.28px;
795         }
796         g.networknodes text {
797             font-size: 10.64px;
798         }
799         text#nodegroup.nodetext {
800             font-size: 10.64px;
801         }
802     }
803
804     @media all and (min-width:701px) and (max-height: 818px) {
        /*0.856*/
805         #header{
```

```
806             height:35.05px;
807         }
808         #header_left{
809             height:35.05px;
810         }
811 
812         #header_right{
813             height:35.05px;
814         }
815         #left {
816             height:644.69px;
817         }
818         #left_top {
819             height:325.89px;
820         }
821         #left_bottom {
822             height:311.79px;
823         }
824         p#left_option {
825             font-size: 1.05em;
826         }
827         p#right_option {
828             font-size: 1.12em;
829         }
830         p#section_top_p {
831             font-size: 1.05em;
832         }
833         p#section_bottom_p {
834             font-size: 1.05em;
835         }
836         p#header_p {
```

```
837             font-size: 1.40em;
838         }
839         p#header_p_left {
840             font-size: 1.40em;
841         }
842 
843         p#header_p_right {
844             font-size: 1.40em;
845         }
846         p#footer_p{
847             font-size: 0.7em;
848         }
849         #section {
850             height:644.92px;
851         }
852         #section_top {
853             height:434.62px;
854         }
855         #section_bottom {
856             height:206.72px;
857         }
858         #right {
859             height:644.92px;
860         }
861         #right_top {
862             height:291.56px;
863         }
864         #right_bottom {
865             height:331.12px;
866         }
867         #footer {
```

```
868             height:14px;
869         }
870         select#op_postposition {
871             height: 21.03px;
872             font-size: 11.91px;
873         }
874         select#op_function {
875             height: 21.03px;
876             font-size: 11.91px;
877         }
878         select#op_method {
879             height: 21.03px;
880             font-size: 11.91px;
881         }
882         select#op_window {
883             height: 21.03px;
884             font-size: 11.91px;
885         }
886         select#op_node_size {
887             height: 21.03px;
888             font-size: 11.91px;
889         }
890         select#op_node_color {
891             height: 21.03px;
892             font-size: 11.91px;
893         }
894         input#onoff {
895             font-size: 9.8px;
896         }
897
898         #container_leftbottom {
```

```
899             height: 266.24px;
900         }
901         #container_rightbottom {
902             height: 301.37px;
903         }
904         #container_section_bottom {
905             height: 171.69px;
906         }
907         div.tooltip {
908             padding: 3.50px;
909             font-size: 11.91px;
910             border-radius: 5.60px;
911         }
912         .CB_leftmiddle{
913             font-size: 10.51px;
914         }
915         .CB_leftbottom{
916             font-size: 10.51px;
917         }
918         table {
919             font-size: 10.51px;
920         }
921         img#header_img{
922             width: 24.53px;
923             height: 22.43px;
924         }
925         h5 {
926             font-size: 10.51px;
927         }
928         g.networknodes text {
929             font-size: 9.11px;
```

```
930         }
931         text#nodegroup.nodetext {
932             font-size: 9.11px;
933         }
934     }
935 
936     @media all and (min-width:450px) and (max-height: 700px) {
            /*0.642*/
937         #header{
938             height:22.5px;
939         }
940         #header_left{
941             height:22.5px;
942         }
943 
944         #header_right{
945             height:22.5px;
946         }
947         #left {
948             height:413.89px;
949         }
950         #left_top {
951             height:200.9px;
952         }
953         #left_bottom {
954             height:208.49px;
955         }
956         p#left_option {
957             font-size: 0.675em;
958         }
959         p#right_option {
```

```
960             font-size: 0.72em;
961         }
962         p#section_top_p {
963             font-size: 0.675em;
964         }
965         p#section_bottom_p {
966             font-size: 0.6741em;
967         }
968         p#header_p {
969             font-size: 0.9em;
970         }
971         p#header_p_left {
972             font-size: 0.9em;
973         }
974 
975         p#header_p_right {
976             font-size: 0.9em;
977         }
978         p#footer_p{
979             font-size: 0.3em;
980         }
981         #section {
982             height:414px;
983         }
984         #section_top {
985             height:279px;
986         }
987         #section_bottom {
988             height:132.7142px;
989         }
990         #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;
        }
```

```
1022        select#op_node_color {
1023            height: 13.5px;
1024            font-size: 7.65px;
1025        }
1026        input#onoff {
1027            font-size: 6.29px;
1028        }
1029
1030        #container_leftbottom {
1031            height: 179.24px;
1032        }
1033        #container_rightbottom {
1034            height: 194.9495px;
1035        }
1036        #container_section_bottom {
1037            height: 110.225px;
1038        }
1039        div.tooltip {
1040            padding: 2.25px;
1041            font-size: 7.65px;
1042            border-radius: 3.6px;
1043        }
1044        .CB_leftmiddle{
1045            font-size: 6.75px;
1046        }
1047        .CB_leftbottom{
1048            font-size: 6.75px;
1049        }
1050        table {
1051            font-size: 6.75px;
1052        }
```

```
1053            img#header_img{
1054                width: 15.75px;
1055                height: 14.4px;
1056            }
1057            h5 {
1058                font-size: 6.75px;
1059            }
1060            g.networknodes text {
1061                font-size: 5.85px;
1062            }
1063            text#nodegroup.nodetext {
1064                font-size: 5.85px;
1065            }
1066        }
1067        </style>
1068
1069        </head>
1070        <body>
1071            <div id="header">
1072                <div id="header_left">
1073                    <p id="header_p_left" align="left">
                            PostEmbedding</p>
1074                </div>
1075                <div id="header_right" align="right">
1076                    <p id="header_p_right"><a id="header_a" href="
                            https://github.com/seongmin-mun/
                            VisualSystem/tree/master/Major/
                            PostEmbedding">GitHub</a></p>
1077                </div>
1078            </div>
1079            <div id="left">
```

```
1080            <div id="left_top">
1081                <p id="left_option">Postposition</p>
1082                <select id="op_postposition">
1083                    <option value="ey" selected="selected">
                        -ey</option>
1084                    <option value="eyse">-eyse</option>
1085                    <option value="(u)lo">-(u)lo</option>
1086                </select>
1087                <p id="left_option">Method</p>
1088                <select id="op_method">
1089                    <option value="ppmi_svd" selected="
                        selected">PPMI & SVD</option>
1090                    <option value="sgns">SGNS</option>
1091                </select>
1092                <p id="left_option">Context window size</p>
1093                <select id="op_window">
1094                    <option value="window1" selected="selected
                        ">window 1</option>
1095                    <option value="window2">window 2</option>
1096                    <option value="window3">window 3</option>
1097                    <option value="window4">window 4</option>
1098                    <option value="window5">window 5</option>
1099                    <option value="window6">window 6</option>
1100                    <option value="window7">window 7</option>
1101                    <option value="window8">window 8</option>
1102                    <option value="window9">window 9</option>
1103                    <option value="window10">window 10</option
                        >
1104                </select>
1105                <p id="left_option">Node size</p>
1106                <select id="op_node_size">
```

```
1107                    <option id="frequency_size" value="
                            frequency" selected="selected">
                            frequency</option>
1108                    <option id="nomal_size" value="nomal">
                            default</option>
1109                </select>
1110                <p id="left_option">Node color</p>
1111                <select id="op_node_color">
1112                    <option id="class_color" value="pos"
                            selected="selected">POS</option>
1113                    <option id="nomal_color" value="nomal">
                            default</option>
1114                </select>
1115                <p id="left_option">Text switch</p>
1116                <input class='CB_lefttop' type='checkbox'
                        value='onoff' id='onoff' checked="checked"/
                        ><label class='CB_lefttop' style="
                        padding-left: 13%;">On/Off</label>
1117            </div>
1118            <div id="left_bottom">
1119                <p id="left_option">Select POS</p>
1120                <div id="container_leftbottom">
1121
1122                </div>
1123            </div>
1124        </div>
1125        <div id="section">
1126            <div id="section_top">
1127                <p id="section_top_p">Distributional semantic
                        map with t-SNE</p>
1128            </div>
```

```
1129            <div id="section_bottom">
1130                <p id="section_bottom_p">Concordance table</p>
1131                <div id="container_section_bottom">
1132                    <table class="table" style="margin-bottom:
                            0px;">
1133                        <thead>
1134                            <tr>
1135                                <th>id</th>
1136                                <th>name</th>
1137                                <th>function</th>
1138                                <th>sentences</th>
1139                                <th>lexeme with POS</th>
1140                            </tr>
1141                        </thead>
1142                        <tbody id="concordancetable">
1143                        </tbody>
1144                    </table>
1145                </div>
1146            </div>
1147        </div>
1148        <div id="right">
1149
1150            <div id="right_top">
1151                <p id="right_option">Function</p>
1152                <select id="op_function">
1153
1154                </select>
1155                <p id="right_option">Force directed graph</p>
1156            </div>
1157            <div id="right_bottom">
1158                <p id="right_option">Nearest words</p>
```

```
1159                    <div id="container_rightbottom">
1160                        <table class="table" style="margin-bottom:
                                0px;">
1161                            <thead>
1162                                <tr>
1163                                    <th>id</th>
1164                                    <th>name</th>
1165                                    <th>similarity</th>
1166                                    <th>frequency</th>
1167                                </tr>
1168                            </thead>
1169                            <tbody id="similaritytable">
1170                            </tbody>
1171                        </table>
1172                    </div>
1173                </div>
1174            </div>
1175            <div id="footer">
1176                <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>
1177            </div>
1178    <script>
1179    $(document).ready(function () {
1180
1181    var functionslist_ey = ['LOC','GOL','EFF','CRT','THM','INS','
        AGT','FNS'];
1182    var functionsname_ey = ['Location','Goal','Effector','
        Criterion','Theme','Instrument','Agent','Final State'];]
1183
```

```
1184 var functionslist_eyse = ['SRC','LOC'];
1185 var functionsname_eyse = ['Source','Location'];
1186 //var functionslist_eyse_number_frame = [487,197];
1187
1188 var functionslist_lo = ['LOC','DIR','EFF','CRT','INS','FNS'];
1189 var functionsname_lo = ['Location','Direction','Effector','
        Criterion','Instrument','Final State'];
1190
1191 function draw_op_function_after(post,list,name,name_kr){
1192 //var idname_0 = post[0]+"_"+list[0];
1193 var idname_0 = list[0];
1194 $('#op_function').empty();
1195 $("#op_function").append("<option id='"+idname_0+"' value='"+
        idname_0.toLowerCase()+"' selected='selected'>"+list[0]+" (
        "+name_kr[0]+", "+name[0]+")"+"</option>")
1196 for (var i = 1; i < list.length ; i++) {
1197 //var idname_i = post[i]+"_"+list[i];
1198 var idname_i = list[i];
1199 $("#op_function").append("<option id='"+idname_i+"' value='"+
        idname_i.toLowerCase()+"'>"+list[i]+" ("+name_kr[i]+", "+
        name[i]+")"+"</option>");
1200 }
1201 }
1202
1203 function op_function_change(){
1204 var selected_postposition = $( "#op_postposition" ).val();
1205 if (selected_postposition === "ey"){
1206 draw_op_function_after("ey",functionslist_ey,functionsname_ey,
        functionsname_kr_ey)
1207 } else if (selected_postposition === "eyse"){
```

```
1208 draw_op_function_after("eyse",functionslist_eyse,
        functionsname_eyse,functionsname_kr_eyse)
1209 } else if (selected_postposition === "(u)lo"){
1210 draw_op_function_after("(u)lo",functionslist_lo,
        functionsname_lo,functionsname_kr_lo)
1211 }
1212 drawall();
1213 }
1214
1215 var typeslist = ['NNG','NNP','NNB','NP','NR','VV','VA','MAG','
        MAJ','JKB'];
1216 var typesname = ['Common Noun','Proper Noun','Bound Noun','
        Pronoun','Numeral','Verb','Adjective','General Adverb','
        Conjunctive Adverb','Adverbial Case Marker'];
1217 var POS_name = ['NNG','NNP','NNB','NP','NR','VV','VA','MAG','
        MAJ','JKB'];
1218 var POS_color = ['#4f4cb4','#003783','#6685c7','#7faded','#16
        a1c6','#ab1432','#6c039d','#1d5041','#4c9046','#5b2e90'];
1219
1220 for(var i = 0 ; i < typeslist.length; i++){
1221 var color = ""
1222 var currentPOS = typeslist[i]
1223 if (currentPOS == POS_name[0]) {
1224 color = POS_color[0]
1225 } else if (currentPOS == POS_name[1]) {
1226 color = POS_color[1]
1227 } else if (currentPOS == POS_name[2]) {
1228 color = POS_color[2]
1229 } else if (currentPOS == POS_name[3]) {
1230 color = POS_color[3]
1231 } 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]);
}
```

```
1257 }
1258 for(var i = 0 ; i < sentencedata.length; i++){
1259 for(var j = 0 ; j < sentencedata[i].sentences.length; j++){
1260 $("#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>");
1261 }
1262 }
1263 }
1264
1265 function checkbox() {
1266
1267 var data_checkbox = []
1268 for(var i = 0; i < typeslist.length; i++){
1269 if(checkedeachValue('CB_leftbottom_'+i)!=='null'){
1270 data_checkbox.push(checkedeachValue('CB_leftbottom_'+i));
1271 }
1272 }
1273 return data_checkbox;
1274 }
1275
1276 function checkedeachValue(checkeddata){
1277 var value;
1278 var checkedValue = document.querySelector('#'+checkeddata+':
        checked');
1279 if(checkedValue == null){
1280 value = "null";
1281 } else {
1282 value = checkedValue.value;
```

```
1283  }
1284  return value;
1285  }
1286
1287  var width = $(window).width()
1288  var height = $(window).height()
1289
1290  var section_top_height = $("#section_top").height()
1291
1292  var svgSection = d3.select('#section_top').append('svg')
1293  .attr('width', width * 0.655)
1294  .attr('height', (section_top_height*0.935))
1295  .call(d3.zoom().scaleExtent([0.5, 5]).on("zoom", function () {
1296  svgSection.attr("transform", d3.event.transform)
1297  }))
1298  .append("g");
1299
1300  var right_top_height = $("#right_top").height()
1301
1302  var svgright_top = d3.select("#right_top")
1303  .append("svg")
1304  .attr("width", width * 0.172)
1305  .attr("height", (right_top_height*0.63))
1306
1307  svgright_top.append("rect")
1308  .attr("class", "svgright_rect")
1309  .attr("x", width * 0.008)
1310  .attr("y", 0)
1311  .attr("width", (width * 0.162))
1312  .attr("height",(right_top_height*0.63))
1313  .attr("rx", 6)
```

```
1314  .attr("ry", 6)
1315  .attr("fill", "white")
1316  .attr('stroke', '#C2C1C1')
1317  .attr('stroke-width', '2')
1318
1319  var NodeGroup = svgSection.append("g");
1320
1321  var div_inner = d3.select("#section").append("div")
1322  .attr("class", "tooltip")
1323  .style("opacity", 0);
1324
1325  firstdrawdata();
1326  op_function_change();
1327  var originalsize = $("#nodegroup.nodetext").css("font-size");
1328
1329  d3.selectAll("#op_postposition").on("change",
          op_function_change);
1330  d3.selectAll("#op_function").on("change", drawall);
1331  d3.selectAll("#op_method").on("change", drawall);
1332  d3.selectAll("#op_window").on("change", drawall);
1333  d3.selectAll("#container_leftbottom").on("change", drawall);
1334  d3.selectAll("#onoff").on("change", drawall);
1335  d3.selectAll("#op_node_size").on("change", drawall);
1336  d3.selectAll("#op_node_color").on("change", drawall);
1337
1338  function drawall(){
1339  var selected_postposition = $( "#op_postposition" ).val();
1340  var selected_function = $( "#op_function" ).val();
1341  var selected_method = $( "#op_method" ).val();
1342  var selected_window = $( "#op_window" ).val();
1343  var selected_node_size = $( "#op_node_size" ).val();
```

```
1344 var selected_node_color = $( "#op_node_color" ).val();
1345 drawconcordance_table(sentence_concordance,
         selected_postposition)
1346 var partofspeeches = checkbox();
1347 changedrawdata(selected_postposition,selected_function,
         selected_method,selected_window,selected_node_size,
         selected_node_color,partofspeeches)
1348 }
1349
1350 function firstdrawdata() {
1351 var data = [];
1352 for (var i = 0; i < network_info.length ; i++) {
1353 if ((network_info[i].postposition === 'ey') && (network_info[i
         ].function === 'loc') && (network_info[i].method === '
         ppmi_svd') && (network_info[i].window === 'window1')) {
1354 data.push(network_info[i]);
1355 }
1356 }
1357
1358 drawtable(data[0]);
1359 drawnetwork(data[0]);
1360 var map_data = [];
1361 for (var i = 0; i < DSMs_info.length ; i++) {
1362 if ((DSMs_info[i].postposition === 'ey') && (DSMs_info[i].
         method === 'ppmi_svd') && (DSMs_info[i].window === 'window1
         ')) {
1363 for (var j = 0; j < DSMs_info[i].wordnet.length ; j++) {
1364 var each = {
1365 opacity_value: []
1366 };
1367 each.opacity_value.push(0.6);
```

```
1368 var settings = $.extend({}, each, DSMs_info[i].wordnet[j]);
1369 map_data.push(settings);
1370 }
1371 }
1372 }
1373
1374 var w = width*0.6;
1375 var h = section_top_height;
1376 var padding = (section_top_height*0.12);
1377
1378 var xScale = d3.scale.linear()
1379 .domain([d3.min(map_data, function(d) { return d.y; }), d3.max
        (map_data, function(d) { return d.x; })])
1380 .range([0+padding, w-padding]);
1381
1382 var yScale = d3.scale.linear()
1383 .domain([d3.min(map_data, function(d) { return d.y; }), d3.max
        (map_data, function(d) { return d.y; })])
1384 .range([h-padding, 0+padding]);
1385
1386 NodeGroup.selectAll(".nodedot")
1387 .data(map_data)
1388 .enter()
1389 .append("circle")
1390 .attr("class", "nodedot")
1391 .attr("id", "nodegroup")
1392 .attr("cx", function (d) {
1393 return xScale(d.x)
1394 })
1395 .attr("cy", function (d) {
1396 return yScale(d.y)
```

```
1397  })
1398  .attr("r", function (d) {
1399  if(d.pos=="JKB"){
1400  return 10
1401  } else {
1402  var size = (d.frequency/30 * 4)
1403  if (size <= 4) {
1404  return 4
1405  } else if (20 <= size) {
1406  return 20
1407  } else {
1408  return size
1409  }
1410  }
1411  })
1412  .attr("fill", function (d) {
1413  if (d.pos == POS_name[0]) {
1414  return POS_color[0]
1415  } else if (d.pos == POS_name[1]) {
1416  return POS_color[1]
1417  } else if (d.pos == POS_name[2]) {
1418  return POS_color[2]
1419  } else if (d.pos == POS_name[3]) {
1420  return POS_color[3]
1421  } else if (d.pos == POS_name[4]) {
1422  return POS_color[4]
1423  } else if (d.pos == POS_name[5]) {
1424  return POS_color[5]
1425  } else if (d.pos == POS_name[6]) {
1426  return POS_color[6]
1427  } else if (d.pos == POS_name[7]) {
```

```
1428 return POS_color[7]
1429 } else if (d.pos == POS_name[8]) {
1430 return POS_color[8]
1431 } else if (d.pos == POS_name[9]) {
1432 return POS_color[9]
1433 }
1434 })
1435 .attr("stroke", "black")
1436 .attr("stroke-width", "1px")
1437 .attr("opacity", function (d) {
1438 return d.opacity_value
1439 })
1440 .style("cursor", "pointer")
1441 .on("mouseover", function (d) {
1442 d3.select(this)
1443 .attr("stroke", "black")
1444 .attr("stroke-width", "1px")
1445 .attr("opacity", 1)
1446 })
1447 .on("mouseout", function (d) {
1448 d3.select(this)
1449 .attr("stroke", "black")
1450 .attr("stroke-width", "1px")
1451 .attr("opacity", function (d) {
1452 return d.opacity_value
1453 });
1454 })
1455 .on("mouseenter", function (d) {
1456 if(d.pos=="JKB"){
1457 div_inner.transition()
1458 .duration(200)
```

```
1459  .style("opacity", 0.85);
1460  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/>")
1461  .style("right", "20px")
1462  .style("top", "20px");
1463  } else {
1464  div_inner.transition()
1465  .duration(200)
1466  .style("opacity", 0.85);
1467  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/>")
1468  .style("right", "20px")
1469  .style("top", "20px");
1470  }
1471  })
1472  .on("mouseleave", function () {
1473  div_inner.transition()
1474  .duration(500)
1475  .style("opacity", 0);
1476  });
1477
1478  NodeGroup.selectAll(".nodetext")
1479  .data(map_data)
1480  .enter()
1481  .append("text")
```

```
1482  .attr("class", "nodetext")
1483  .attr("id", "nodegroup")
1484  .text(function (d) {
1485  if(d.pos=="JKB"){
1486  return d.name_kr+"/"+d.name_eng+"/"+d.pos_long;
1487  } else {
1488  return d.name_kr+"/"+d.name_eng+"/"+d.pos;
1489  }
1490  })
1491  .attr("x", function (d) {
1492  return xScale(d.x) + 10
1493  })
1494  .attr("y", function (d) {
1495  return yScale(d.y) + 4
1496  })
1497  .attr("font-family", "sans-serif")
1498  .attr("fill", "rgb(51,51,51)")
1499  .attr("opacity", function (d) {
1500  return d.opacity_value
1501  })
1502  .style("cursor", "pointer")
1503  .on("mouseover", function () {
1504  d3.select(this)
1505  .attr("opacity", 1);
1506  })
1507  .on("mouseout", function (d) {
1508  d3.select(this)
1509  .attr("opacity", function (d) {
1510   return d.opacity_value
1511  });
1512  })
```

```
1513  .on("mouseenter", function (d) {
1514  if(d.pos=="JKB"){
1515  div_inner.transition()
1516  .duration(200)
1517  .style("opacity", 0.85);
1518  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/>")
1519  .style("right", "20px")
1520  .style("top", "20px");
1521  } else {
1522  div_inner.transition()
1523  .duration(200)
1524  .style("opacity", 0.85);
1525  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/>")
1526  .style("right", "20px")
1527  .style("top", "20px");
1528  }
1529
1530  })
1531  .on("mouseleave", function () {
1532  div_inner.transition()
1533  .duration(500)
1534  .style("opacity", 0);
1535  });
```

```
1536 }
1537
1538 function changedrawdata(selected_postposition,
        selected_function,selected_method,selected_window,
        selected_node_size,selected_node_color,partofspeeches) {
1539
1540 $('#similaritytable').empty();
1541 svgright_top.selectAll(".networklinks").remove();
1542 svgright_top.selectAll(".networknodes").remove();
1543
1544 var data = [];
1545 for (var i = 0; i < network_info.length ; i++) {
1546 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)) {
1547 data.push(network_info[i]);
1548 }
1549 }
1550
1551 drawtable(data[0]);
1552 drawnetwork(data[0]);
1553
1554 var map_data = [];
1555 for (var i = 0; i < DSMs_info.length ; i++) {
1556 if ((DSMs_info[i].postposition === selected_postposition) && (
        DSMs_info[i].method === selected_method) && (DSMs_info[i].
        window === selected_window)) {
1557 for (var j = 0; j < DSMs_info[i].wordnet.length ; j++) {
1558 var each = {
1559 opacity_value: []
```

```
1560 };
1561 if(partofspeeches.length > 0 == true){
1562 var checked = false;
1563 for(var k = 0; k < partofspeeches.length ; k++){
1564 if(DSMs_info[i].wordnet[j].pos == partofspeeches[k]){
1565     checked = true;
1566 }
1567 }
1568 if(checked == true){
1569   each.opacity_value.push(0.9);
1570 } else {
1571   each.opacity_value.push(0.2);
1572 }
1573 } else if(partofspeeches.length > 0 == false){
1574 each.opacity_value.push(0.6);
1575 }
1576 var settings = $.extend({}, each, DSMs_info[i].wordnet[j]);
1577 map_data.push(settings);
1578 }
1579 }
1580 }
1581
1582 var w = width*0.6;
1583 var h = section_top_height;
1584 var padding = (section_top_height*0.12);
1585 var xScale = d3.scale.linear()
1586 .domain([d3.min(map_data, function(d) { return d.y; }), d3.max
        (map_data, function(d) { return d.x; })])
1587 .range([0+padding, w-padding]);
1588 var yScale = d3.scale.linear()
```

```
1589  .domain([d3.min(map_data, function(d) { return d.y; }), d3.max
          (map_data, function(d) { return d.y; })])
1590  .range([h-padding, 0+padding]);
1591  var circle = NodeGroup.selectAll(".nodedot")
1592  .data(map_data);
1593
1594  circle.enter()
1595  .append("circle")
1596  .attr("class", "nodedot")
1597  .attr("id", "nodegroup")
1598  .attr("cx", function (d) {
1599  return xScale(d.x)
1600  })
1601  .attr("cy", function (d) {
1602  return yScale(d.y)
1603  })
1604  .attr("r", function (d){
1605  if (selected_node_size === "nomal") {
1606  return 4;
1607  } else if (selected_node_size === "frequency") {
1608  if(d.pos=="JKB"){
1609  return 10
1610  } else {
1611  var size = (d.frequency/30 * 4)
1612  if (size <= 4) {
1613    return 4
1614  } else if (20 <= size) {
1615    return 20
1616  } else {
1617    return size
1618  }
```

```
1619  }
1620  }
1621  })
1622  .attr("fill", function (d){
1623  if (selected_node_color === "nomal") {
1624  return "rgb(51,51,51)"
1625  } else if (selected_node_color === "pos") {
1626  if (d.pos == POS_name[0]) {
1627    return POS_color[0]
1628  } else if (d.pos == POS_name[1]) {
1629    return POS_color[1]
1630  } else if (d.pos == POS_name[2]) {
1631    return POS_color[2]
1632  } else if (d.pos == POS_name[3]) {
1633    return POS_color[3]
1634  } else if (d.pos == POS_name[4]) {
1635    return POS_color[4]
1636  } else if (d.pos == POS_name[5]) {
1637    return POS_color[5]
1638  } else if (d.pos == POS_name[6]) {
1639    return POS_color[6]
1640  } else if (d.pos == POS_name[7]) {
1641    return POS_color[7]
1642  } else if (d.pos == POS_name[8]) {
1643    return POS_color[8]
1644  } else if (d.pos == POS_name[9]) {
1645    return POS_color[9]
1646  }
1647  }
1648  })
1649  .attr("stroke", "black")
```

```
1650  .attr("stroke-width", "1px")
1651  .attr("opacity", function (d) {
1652  return d.opacity_value
1653  })
1654  .style("cursor", "pointer");
1655
1656  circle.transition()
1657  .duration(2000)
1658  .attr("cx", function (d) {
1659  return xScale(d.x)
1660  })
1661  .attr("cy", function (d) {
1662  return yScale(d.y)
1663  })
1664  .attr("r", function (d){
1665  if (selected_node_size === "nomal") {
1666  return 4;
1667  } else if (selected_node_size === "frequency") {
1668  if(d.pos=="JKB"){
1669  return 10
1670  } else {
1671  var size = (d.frequency/30 * 4)
1672  if (size <= 4) {
1673    return 4
1674  } else if (20 <= size) {
1675    return 20
1676  } else {
1677    return size
1678  }
1679  }
1680  }
```

```
1681  })
1682  .attr("fill", function (d){
1683  if (selected_node_color === "nomal") {
1684  return "rgb(51,51,51)"
1685  } else if (selected_node_color === "pos") {
1686  if (d.pos == POS_name[0]) {
1687    return POS_color[0]
1688  } else if (d.pos == POS_name[1]) {
1689    return POS_color[1]
1690  } else if (d.pos == POS_name[2]) {
1691    return POS_color[2]
1692  } else if (d.pos == POS_name[3]) {
1693    return POS_color[3]
1694  } else if (d.pos == POS_name[4]) {
1695    return POS_color[4]
1696  } else if (d.pos == POS_name[5]) {
1697    return POS_color[5]
1698  } else if (d.pos == POS_name[6]) {
1699    return POS_color[6]
1700  } else if (d.pos == POS_name[7]) {
1701    return POS_color[7]
1702  } else if (d.pos == POS_name[8]) {
1703    return POS_color[8]
1704  } else if (d.pos == POS_name[9]) {
1705    return POS_color[9]
1706  }
1707  }
1708  })
1709  .attr("stroke", "black")
1710  .attr("stroke-width", "1px")
1711  .attr("opacity", function (d) {
```

```
1712 return d.opacity_value
1713 })
1714 .style("cursor", "pointer");
1715
1716 circle.exit().remove();
1717
1718 var text = NodeGroup.selectAll(".nodetext")
1719 .data(map_data);
1720
1721 text.enter()
1722 .append("text")
1723 .attr("class", "nodetext")
1724 .attr("id", "nodegroup")
1725 .text(function (d) {
1726 if(d.pos=="JKB"){
1727 return d.name_kr+"/"+d.name_eng+"/"+d.pos_long;
1728 } else {
1729 return d.name_kr+"/"+d.name_eng+"/"+d.pos;
1730 }
1731 })
1732 .attr("x", function (d) {
1733 return xScale(d.x) + 10
1734 })
1735 .attr("y", function (d) {
1736 return yScale(d.y) + 4
1737 })
1738 .attr("font-family", "sans-serif")
1739 .attr("fill", "rgb(51,51,51)")
1740 .attr("opacity", function (d) {
1741 return d.opacity_value
1742 })
```

```
1743  .style("cursor", "pointer");
1744
1745  text.transition()
1746  .duration(2000)
1747  .text(function (d) {
1748  if(d.pos=="JKB"){
1749  return d.name_kr+"/"+d.name_eng+"/"+d.pos_long;
1750  } else {
1751  return d.name_kr+"/"+d.name_eng+"/"+d.pos;
1752  }
1753  })
1754  .attr("x", function (d) {
1755  return xScale(d.x) + 10
1756  })
1757  .attr("y", function (d) {
1758  return yScale(d.y) + 4
1759  })
1760  .attr("font-family", "sans-serif")
1761  .attr("fill", "rgb(51,51,51)")
1762  .attr("opacity", function (d) {
1763  return d.opacity_value
1764  })
1765  .style("cursor", "pointer");
1766
1767  text.exit().remove();
1768
1769  if(checkedeachValue("onoff")=='null'){
1770  svgSection.selectAll(".nodetext").remove();
1771  }
1772  }
1773
```

```
1774 function drawtable_first(data){
1775 $('#similaritytable').empty();
1776 for(var i = 0 ; i < data.links.length; i++){
1777 $("#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>");
1778 }
1779 }
1780
1781 function drawtable(data){
1782 $('#similaritytable').empty();
1783 console.log(typeof(data.links[0].target));
1784
1785 if(typeof(data.links[0].target)=="object"){
1786 for(var i = 0 ; i < data.links.length; i++){
1787 $("#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>");
1788 }
1789 } else {
1790 for(var i = 0 ; i < data.links.length; i++){
1791 $("#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>");
1792 }
1793 }
1794 }
1795
1796 function drawnetwork(data){
1797
```

```
1798 svgright_top.selectAll(".networklinks").remove();
1799 svgright_top.selectAll(".networknodes").remove();
1800
1801 var simulation = d3.forceSimulation()
1802 .force("link", d3.forceLink().id(function(d) { return d.id; })
        .distance(function (d) { return (right_top_height*0.828)
        *0.28}))
1803 .force("charge", d3.forceManyBody())
1804 .force("center", d3.forceCenter(width * 0.14 / 2, (
        right_top_height*0.63) / 2));
1805
1806 var link = svgright_top.append("g")
1807 .attr("class", "networklinks")
1808 .selectAll("line")
1809 .data(data.links)
1810 .enter().append("line")
1811 .attr("stroke-width", function(d) { return Math.sqrt(d.value
        *6); });
1812
1813 var node = svgright_top.append("g")
1814 .attr("class", "networknodes")
1815 .selectAll("g")
1816 .data(data.nodes)
1817 .enter().append("g")
1818
1819 var circles = node.append("circle")
1820 .attr("r", function(d) {
1821 var size = d.frequency/2
1822 if(15 < size){
1823 return 15;
1824 } else if (size < 4){
```

```
1825  return 4;
1826  } else {
1827  return size;
1828  }
1829  })
1830  .attr("fill", function (d) {
1831  if (d.pos == POS_name[0]) {
1832  return POS_color[0]
1833  } else if (d.pos == POS_name[1]) {
1834  return POS_color[1]
1835  } else if (d.pos == POS_name[2]) {
1836  return POS_color[2]
1837  } else if (d.pos == POS_name[3]) {
1838  return POS_color[3]
1839  } else if (d.pos == POS_name[4]) {
1840  return POS_color[4]
1841  } else if (d.pos == POS_name[5]) {
1842  return POS_color[5]
1843  } else if (d.pos == POS_name[6]) {
1844  return POS_color[6]
1845  } else if (d.pos == POS_name[7]) {
1846  return POS_color[7]
1847  } else if (d.pos == POS_name[8]) {
1848  return POS_color[8]
1849  } else if (d.pos == POS_name[9]) {
1850  return POS_color[9]
1851  } else {
1852  return '#C2C1C1'
1853  }
1854  })
1855  .call(d3.drag()
```

```
1856  .on("start", dragstarted)
1857  .on("drag", dragged)
1858  .on("end", dragended));
1859
1860  var lables = node.append("text")
1861  .text(function(d) {
1862  return d.id;//.split("/")[0]+"/"+d.id_eng;
1863  })
1864  .attr('x', 6)
1865  .attr('y', 3)
1866  .attr('opacity',0.4)
1867  .on("mouseover", function () {
1868  d3.select(this).attr("opacity", 1);
1869  })
1870  .on("mouseout", function (d) {
1871  d3.select(this).attr("opacity", 0.4);
1872  });
1873
1874  node.append("title")
1875  .text(function(d) { return d.id; });
1876
1877  simulation.nodes(data.nodes)
1878  .on("tick", ticked);
1879
1880  simulation.force("link")
1881  .links(data.links);
1882
1883  function ticked() {
1884  link.attr("x1", function(d) { return d.source.x; })
1885  .attr("y1", function(d) { return d.source.y; })
1886  .attr("x2", function(d) { return d.target.x; })
```

```
1887  .attr("y2", function(d) { return d.target.y; });
1888
1889  node.attr("transform", function(d) {
1890  return "translate(" + d.x + "," + d.y + ")";
1891  })
1892  }
1893
1894  function dragstarted(d) {
1895  if (!d3.event.active) simulation.alphaTarget(0.3).restart();
1896  d.fx = d.x;
1897  d.fy = d.y;
1898  }
1899
1900  function dragged(d) {
1901  d.fx = d3.event.x;
1902  d.fy = d3.event.y;
1903  }
1904
1905  function dragended(d) {
1906  if (!d3.event.active) simulation.alphaTarget(0);
1907  d.fx = null;
1908  d.fy = null;
1909  }
1910  }
1911  })
1912  </script>
1913  </body>
1914  </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

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4     <title>PostBERT</title><!--<link rel="stylesheet" href="./
          stylesheets/bubble_style.css">-->
5     <meta http-equiv="Content-Type" content="text/html;
          charset=utf-8">
6     <script src="./javascripts/d3.v3.min.js" charset="utf-8">
          </script>
7     <script src="./javascripts/d3.v4.js" charset="utf-8">
          </script>
8     <script src="./javascripts/jquery-1.12.0.min.js" charset="
          utf-8"></script>
```

```
9       <link rel="stylesheet" href="https://
            maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
            bootstrap.min.css">
10      <script src="https://maxcdn.bootstrapcdn.com/bootstrap
            /3.3.7/js/bootstrap.min.js"></script>
11      <!--<link rel="stylesheet" href="./stylesheets/PostVis.css
            ">-->
12
13      <script src="./Data/Madpdata.js" charset="utf-8"></script>
14      <script src="./Data/Sentencedata.js" charset="utf-8">
            </script>
15      <script src="./Data/Accuracydata.js" charset="utf-8">
            </script>
16      <script src="./Data/Clusterdata.js" charset="utf-8">
            </script>
17
18      <style>
19      body {
20          margin: 0;
21      }
22
23      #header{
24          position:relative;
25          width:100%;
26          height:50px;
27          background-color:#f3c623;
28          background-clip: content-box;
29      }
30
31      #left {
32          position:relative;
```

```
33          float:left;
34          overflow:hidden;
35          width:14%;
36          height:920px;
37          padding: 0.5%;
38          background-color:whitesmoke;
39          background-clip: content-box;
40      }
41
42      #left_top {
43          position:relative;
44          float:left;
45          overflow:hidden;
46          width:100%;
47          height:85px;
48          padding: 0.5%;
49          background-clip: content-box;
50      }
51
52      #left_middle {
53          position:relative;
54          float:left;
55          overflow:hidden;
56          width:100%;
57          height:250px;
58          padding: 0.5%;
59          background-clip: content-box;
60      }
61
62      #left_bottom {
63          position:relative;
```

```
64          float:left;
65          overflow:hidden;
66          width:100%;
67          height:290px;
68          padding: 0.5%;
69          background-clip: content-box;
70      }
71
72      #left_bottom_bottom {
73          position:relative;
74          float:left;
75          overflow:hidden;
76          width:100%;
77          height:355px;
78          padding: 0.5%;
79          background-clip: content-box;
80      }
81
82
83      #section {
84          position:relative;
85          float:left;
86          overflow:hidden;
87          width:57%;
88          height:920px;
89          padding: 0.5%;
90          background-clip: content-box;
91      }
92
93      #section_top {
94          position:relative;
```

```
95          float:left;
96          overflow:hidden;
97          width:100%;
98          height:795px;
99          padding-right: : 0.5%;
100         padding-top: 0.5%;
101         padding-left: 0.5%;
102         background-clip: content-box;
103     }
104
105     #section_bottom {
106         position:relative;
107         float:left;
108         overflow:hidden;
109         width:100%;
110         height:120px;
111         padding-right: : 0.5%;
112         padding-bottom: 0.5%;
113         padding-left: 0.5%;
114         background-clip: content-box;
115     }
116
117     #section_bottom_left {
118         position:relative;
119         float:left;
120         overflow:hidden;
121         width:12%;
122         height:80px;
123         padding-right: : 0.5%;
124         padding-bottom: 0.5%;
125         padding-left: 0.5%;
```

```
126         background-clip: content-box;
127     }
128 
129     #section_bottom_right {
130         position:relative;
131         float:left;
132         overflow:hidden;
133         width:88%;
134         height:80px;
135         padding-right: : 0.5%;
136         padding-bottom: 0.5%;
137         padding-left: 0.5%;
138         background-clip: content-box;
139     }
140 
141     #right {
142         position:relative;
143         float:right;
144         overflow:hidden;
145         width:29%;
146         height:920px;
147         padding: 0.5%;
148         background-color:whitesmoke;
149         background-clip: content-box;
150     }
151 
152     #right_top {
153         position:relative;
154         float:left;
155         overflow:hidden;
156         width:100%;
```

```
157         height:300px;
158         padding: 0.5%;
159         background-clip: content-box;
160     }
161 
162     #right_middle {
163         position:relative;
164         float:left;
165         overflow:hidden;
166         width:100%;
167         height:300px;
168         padding: 0.5%;
169         background-clip: content-box;
170     }
171 
172 
173     #right_bottom {
174         position:relative;
175         float:left;
176         overflow:hidden;
177         width:100%;
178         height:305px;
179         padding: 0.5%;
180         background-clip: content-box;
181     }
182 
183     #header_left{
184         float:left;
185         width:49%;
186         height:50px;
187         padding-top: 0.1%;
```

```
188         background-clip: content-box;
189     }
190 
191     #header_right{
192         float:right;
193         width:49%;
194         height:50px;
195         padding-top: 0.1%;
196         background-clip: content-box;
197     }
198 
199     #footer {
200         height:20px;
201         text-align: center;
202         color: white;
203         background-color:#717171;
204         clear:both;
205     }
206 
207 
208     p#left_option {
209         text-align: left;
210         font-family: Open Sans;
211         font-size: 1.5em;
212         color: #666666;
213         font-weight: bold;
214         padding-top: 4%;
215         padding-bottom: 4%;
216         padding-left: 5%;
217         margin: 0;
218     }
```

```
219
220     p#right_option {
221         text-align: left;
222         font-family: Open Sans;
223         font-size: 1.6em;
224         color: #666666;
225         font-weight: bold;
226         padding-top: 4%;
227         padding-bottom: 4%;
228         padding-left: 5%;
229         margin: 0;
230     }
231
232     p#section_top_p {
233         font-family: Open Sans;
234         font-size: 1.5em;
235         color: #666666;
236         font-weight: bold;
237         padding-left: 1%;
238         margin: 0;
239     }
240
241     p#section_bottom_p {
242         font-family: Open Sans;
243         font-size: 1.5em;
244         color: #666666;
245         font-weight: bold;
246         padding-left: 1%;
247         margin: 0;
248     }
249
```

```
250     p#header_p_left {
251         position: relative;
252         text-align: left;
253         font-family: Open Sans;
254         font-size: 2em;
255         color: white;
256         font-weight: bold;
257         padding-top: 0.5%;
258         padding-bottom: 0.5%;
259         padding-left: 1%;
260         margin: 0;
261     }
262 
263     p#header_p_right {
264         position: relative;
265         text-align: right;
266         font-family: Open Sans;
267         font-size: 2em;
268         color: white;
269         font-weight: bold;
270         padding-top: 0.5%;
271         padding-bottom: 0.5%;
272         padding-right: 2%;
273         margin: 0;
274     }
275 
276     p#footer_p{
277         font-size: 1em;
278     }
279 
280     a#header_a{
```

```
281         font-family: Open Sans;
282         color: white;
283         font-weight: bold;
284         cursor: pointer;
285     }
286
287     a#footer_a{
288         font-family: Open Sans;
289         color: #f3c623;
290         font-weight: bold;
291         cursor: pointer;
292     }
293
294
295     select#op_postposition { /*text-align-last:center;*/
296        width: 90%;
297        height: 30px;
298        font-size: 17px;
299        border-radius: 3px;
300        position: relative;
301        left:5%;
302        background: white;
303        cursor: pointer;
304     }
305
306     /*select#op_node_color {
307        width: 90%;
308        height: 30px;
309        font-size: 17px;
310        border-radius: 3px;
311        position: relative;
```

```
312         left:5%;
313         background: white;
314         cursor: pointer;
315     }*/
316
317     #container_leftmiddle {
318         border:2px solid #ccc;
319         width:88%;
320         height: 200px;
321         position: absolute;
322         left: 5%;
323         overflow-y: scroll;
324         overflow-x: auto;
325         white-space: nowrap;
326         border-radius: 10px;
327         background: white;
328     }
329
330     .CB_leftmiddle {
331         cursor: pointer;
332         position: relative;
333         font-size: 14px;
334         left:5%;
335     }
336
337
338     #container_leftbottom {
339         border:2px solid #ccc;
340         width:88%;
341         height: 240px;
342         position: absolute;
```

```
343         left: 5%;
344         overflow-y: scroll;
345         overflow-x: auto;
346         white-space: nowrap;
347         border-radius: 10px;
348         background: white;
349     }
350 
351     .CB_leftbottom{
352         cursor: pointer;
353         position: relative;
354         font-size: 14px;
355         left:5%;
356     }
357 
358 
359 
360     #play-button {
361       position: absolute;
362       top: 25%;
363       background: #f08080;
364       padding-right: 10px;
365       border-radius: 3px;
366       border: none;
367       color: white;
368       margin: 0;
369       width: 80%;
370       cursor: pointer;
371       height: 40%;
372       font: 13px sans-serif;
373     }
```

```
374
375     #play-button:hover {
376       background-color: #696969;
377     }
378
379     #play-button:active {
380       background-color: #002657;
381     }
382
383
384     .ticks {
385       font: 10px sans-serif;
386     }
387
388     .track,
389     .track-inset,
390     .track-overlay {
391       stroke-linecap: round;
392     }
393
394     .track {
395       stroke: #000;
396       stroke-opacity: 0.3;
397       stroke-width: 10px;
398     }
399
400     .track-inset {
401       stroke: #dcdcdc;
402       stroke-width: 8px;
403     }
404
```

```
405     .track-overlay {
406       pointer-events: stroke;
407       stroke-width: 50px;
408       cursor: pointer;
409     }
410
411     .handle {
412       fill: #fff;
413       stroke: #000;
414       stroke-opacity: 0.5;
415       stroke-width: 1.25px;
416     }
417
418
419     div.tooltip {
420         position: absolute;
421         text-align: left;
422         padding: 5px;
423         font-size: 17px;
424         background-color: #efefef;
425         border: solid 1px #cecece;
426         border-radius: 8px;
427         box-shadow: 0 3px 5px 0 #dfdfdf;
428         pointer-events: none;
429     }
430
431     div.epoch {
432         position: absolute;
433         text-align: right;
434         padding: 5px;
435         font-size: 40px;
```

```
436         background-color: white;
437         border: solid 1px white;
438         border-radius: 8px;
439         pointer-events: none;
440     }
441 
442     #corBar:hover {
443             fill: orange;
444         }
445 
446     #tooltip_top {
447         position: absolute;
448         width: 120px;
449         height: auto;
450         padding: 10px;
451         background-color: white;
452         -webkit-border-radius: 10px;
453         -moz-border-radius: 10px;
454         border-radius: 10px;
455         -webkit-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
456         -moz-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
457         box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
458         pointer-events: none;
459         font-size: 16px;
460     }
461 
462     #tooltip_top.hidden {
463         display: none;
464     }
465 
466     #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;
    }
```

```
498
499     .rightbottom_bar:hover {
500         fill: #f08080;
501     }
502
503     #tooltip_bottom {
504         position: absolute;
505         width: 160px;
506         height: auto;
507         padding: 10px;
508         background-color: white;
509         -webkit-border-radius: 10px;
510         -moz-border-radius: 10px;
511         border-radius: 10px;
512         -webkit-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
513         -moz-box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
514         box-shadow: 4px 4px 10px rgba(0, 0, 0, 0.4);
515         pointer-events: none;
516         opacity: 0.7;
517     }
518
519     #tooltip_bottom.hidden {
520         display: none;
521     }
522
523     #tooltip_bottom p {
524         margin: 0;
525         font-family: sans-serif;
526         font-size: 16px;
527         line-height: 20px;
528     }
```

```
529
530     p.tooltip_topdiv{
531         font-size: 16px;
532     }
533     p.tooltip_middlediv{
534         font-size: 16px;
535     }
536
537     h5 {
538         font-size: 15px;
539     }
540
541
542     @media all and (min-width:951px) and (max-height: 1000px)
            { /*0.95*/
543         #header{
544             height:47.55px;
545         }
546         #header_left{
547             height:47.55px;
548         }
549
550         #header_right{
551             height:47.55px;
552         }
553
554         #left {
555             height:874px;
556         }
557
558         #left_top {
```

```
559             height:80.75px;
560         }
561
562         #left_middle {
563             height:237.5px;
564         }
565
566         #left_bottom {
567             height:275.5px;
568         }
569
570         #left_bottom_bottom {
571             height:337.25px;
572         }
573         #section {
574             height:874px;
575         }
576
577         #section_top {
578             height:755.25px;
579         }
580
581         #section_bottom {
582             height:114px;
583         }
584
585         #section_bottom_left {
586             height:76px;
587         }
588
589         #section_bottom_right {
```

```
590             height:76px;
591         }
592
593         #right {
594             height:874px;
595         }
596
597         #right_top {
598             height:285px;
599         }
600
601         #right_middle {
602             height:285px;
603         }
604         #right_bottom {
605             height:289.75px;
606         }
607
608         #footer {
609             height:19px;
610         }
611
612         p#left_option {
613             font-size: 1.425em;
614         }
615         p#section_top_p {
616             font-size: 1.425em;
617         }
618         p#section_bottom_p {
619             font-size: 1.425em;
620         }
```

```
621         p#right_option {
622             font-size: 1.52em;
623         }
624         p#header_p_left {
625             font-size: 1.9em;
626         }
627         p#header_p_right {
628             font-size: 1.9em;
629         }
630         p#footer_p{
631             font-size: 0.95em;
632         }
633         select#op_postposition {
634            height: 28.5px;
635            font-size: 16.15px;
636         }
637         #container_leftmiddle {
638             height: 190px;
639             border-radius: 9.5px;
640         }
641         #container_leftbottom {
642             height: 228px;
643             border-radius: 9.5px;
644         }
645         .CB_leftmiddle {
646             font-size: 13.3px;
647         }
648
649         .CB_leftbottom{
650             font-size: 13.3px;]
651         }
```

```
652         #play-button {
653           font: 12.35px sans-serif;
654         }
655         .ticks {
656           font: 9.5px sans-serif;
657         }
658         .track {
659           stroke-width: 9.5px;
660         }
661         .track-inset {
662           stroke-width: 7.6px;
663         }
664         .track-overlay {
665           stroke-width: 47.5px;
666         }
667         div.tooltip {
668             font-size: 16.15px;
669         }
670         div.epoch {
671             font-size: 38px;
672         }
673         #tooltip_top {
674             width: 114px;
675             font-size: 15.2px;
676         }
677         #tooltip_middle {
678             width: 114px;
679             font-size: 15.2px;
680         }
681         #tooltip_bottom {
682             width: 152px;
```

```
683         }
684         #tooltip_bottom p {
685             font-size: 15.2px;
686         }
687         h5 {
688             font-size: 14.25px;
689         }
690 
691         text.rightbottom_text {
692             font-size: 19px;
693         }
694         text.rangetext{
695             font-size: 19.95px;
696         }
697 
698     }
699 
700 
701     @media all and (min-width:901px) and (max-height: 950px) {
            /*0.90*/
702         #header{
703             height:45px;
704         }
705         #header_left{
706             height:45px;
707         }
708 
709         #header_right{
710             height:45px;
711         }
712 
```

```
713         #left {
714             height:828px;
715         }
716
717         #left_top {
718             height:76.5px;
719         }
720
721         #left_middle {
722             height:225px;
723         }
724
725         #left_bottom {
726             height:261px;
727         }
728
729         #left_bottom_bottom {
730             height:319.5px;
731         }
732         #section {
733             height:828px;
734         }
735
736         #section_top {
737             height:715.5px;
738         }
739
740         #section_bottom {
741             height:108px;
742         }
743
```

```
744         #section_bottom_left {
745             height:72px;
746         }
747 
748         #section_bottom_right {
749             height:72px;
750         }
751 
752         #right {
753             height:828px;
754         }
755 
756         #right_top {
757             height:270px;
758         }
759 
760         #right_middle {
761             height:270px;
762         }
763 
764 
765         #right_bottom {
766             height:274.5px;
767         }
768 
769         #footer {
770             height:18px;
771         }
772 
773         p#left_option {
774             font-size: 1.35em;
```

```
775         }
776         p#section_top_p {
777             font-size: 1.35em;
778         }
779         p#section_bottom_p {
780             font-size: 1.35em;
781         }
782         p#right_option {
783             font-size: 1.44em;
784         }
785         p#header_p_left {
786             font-size: 1.8em;
787         }
788         p#header_p_right {
789             font-size: 1.8em;
790         }
791         p#footer_p{
792             font-size: 0.9em;
793         }
794         select#op_postposition {
795            height: 27px;
796            font-size: 15.3px;
797         }
798         #container_leftmiddle {
799             height: 180px;
800             border-radius: 9px;
801         }
802         #container_leftbottom {
803             height: 216px;
804             border-radius: 9px;
805         }
```

```
806         .CB_leftmiddle {
807             font-size: 12.6px;
808         }
809
810         .CB_leftbottom{
811             font-size: 12.6px;]
812         }
813         #play-button {
814           font: 11.7px sans-serif;
815         }
816         .ticks {
817           font: 9px sans-serif;
818         }
819         .track {
820           stroke-width: 9px;
821         }
822         .track-inset {
823           stroke-width: 7.2px;
824         }
825         .track-overlay {
826           stroke-width: 45px;
827         }
828         div.tooltip {
829             font-size: 15.3px;
830         }
831         div.epoch {
832             font-size: 36px;
833         }
834         #tooltip_top {
835             width: 108px;
836             font-size: 14.4px;
```

```
837         }
838         #tooltip_middle {
839             width: 108px;
840             font-size: 14.4px;
841         }
842         #tooltip_bottom {
843             width: 144px;
844         }
845         #tooltip_bottom p {
846             font-size: 14.4px;
847         }
848         h5 {
849             font-size: 13.5px;
850         }
851 
852         text.rightbottom_text {
853             font-size: 18px;
854         }
855         text.rangetext{
856             font-size: 18.9px;
857         }
858     }
859 
860 
861     @media all and (min-width:819px) and (max-height: 900px) {
        /*0.82*/
862         #header{
863             height:40.95px;
864         }
865         #header_left{
866             height:40.95px;
```

```
867         }
868
869         #header_right{
870             height:40.95px;
871         }
872         #left {
873             height:754.4px;
874         }
875
876         #left_top {
877             height:69.7px;
878         }
879
880         #left_middle {
881             height:205px;
882         }
883
884         #left_bottom {
885             height:237.8px;
886         }
887
888         #left_bottom_bottom {
889             height:291.1px;
890         }
891         #section {
892             height:754.4px;
893         }
894
895         #section_top {
896             height:651.9px;
897         }
```

```
898
899         #section_bottom {
900             height:98.4px;
901         }
902
903         #section_bottom_left {
904             height:65.6px;
905         }
906
907         #section_bottom_right {
908             height:65.6px;
909         }
910
911         #right {
912             height:754.4px;
913         }
914
915         #right_top {
916             height:246px;
917         }
918
919         #right_middle {
920             height:246px;
921         }
922
923
924         #right_bottom {
925             height:250.1px;
926         }
927
928         #footer {
```

```
929             height:16.4px;
930         }
931
932         p#left_option {
933             font-size: 1.23em;
934         }
935         p#section_top_p {
936             font-size: 1.23em;
937         }
938         p#section_bottom_p {
939             font-size: 1.23em;
940         }
941         p#right_option {
942             font-size: 1.312em;
943         }
944         p#header_p_left {
945             font-size: 1.64em;
946         }
947         p#header_p_right {
948             font-size: 1.64em;
949         }
950         p#footer_p{
951             font-size: 0.82em;
952         }
953         select#op_postposition {
954            height: 24.6px;
955            font-size: 13.94px;
956         }
957         #container_leftmiddle {
958             height: 164px;
959             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 {
```

```
991             font-size: 32.8px;
992         }
993         #tooltip_top {
994             width: 98.4px;
995             font-size: 13.12px;
996         }
997         #tooltip_middle {
998             width: 98.4px;
999             font-size: 13.12px;
1000        }
1001        #tooltip_bottom {
1002            width: 131.2px;
1003        }
1004        #tooltip_bottom p {
1005            font-size: 13.12px;
1006        }
1007        h5 {
1008            font-size: 12.3px;
1009        }
1010
1011        text.rightbottom_text {
1012            font-size: 16.4px;
1013        }
1014        text.rangetext{
1015            font-size: 17.22px;
1016        }
1017    }
1018
1019    @media all and (min-width:701px) and (max-height: 818px) {
        /*0.72*/
1020        #header{
```

```
1021            height:35.05px;
1022        }
1023        #header_left{
1024            height:35.05px;
1025        }
1026
1027        #header_right{
1028            height:35.05px;
1029        }
1030        #left {
1031            height:662.4px;
1032        }
1033
1034        #left_top {
1035            height:61.2px;
1036        }
1037
1038        #left_middle {
1039            height:180px;
1040        }
1041
1042        #left_bottom {
1043            height:208.8px;
1044        }
1045
1046        #left_bottom_bottom {
1047            height:255.6px;
1048        }
1049        #section {
1050            height:662.4px;
1051        }
```

```
1052
1053            #section_top {
1054                height:572.4px;
1055            }
1056
1057            #section_bottom {
1058                height:86.4px;
1059            }
1060
1061            #section_bottom_left {
1062                height:57.6px;
1063            }
1064
1065            #section_bottom_right {
1066                height:57.6px;
1067            }
1068
1069            #right {
1070                height:662.4px;
1071            }
1072
1073            #right_top {
1074                height:216px;
1075            }
1076
1077            #right_middle {
1078                height:216px;
1079            }
1080
1081
1082            #right_bottom {
```

```
1083            height:219.6px;
1084        }
1085
1086        #footer {
1087            height:14.4px;
1088        }
1089
1090        p#left_option {
1091            font-size: 1.08em;
1092        }
1093        p#section_top_p {
1094            font-size: 1.08em;
1095        }
1096        p#section_bottom_p {
1097            font-size: 1.08em;
1098        }
1099        p#right_option {
1100            font-size: 1.152em;
1101        }
1102        p#header_p_left {
1103            font-size: 1.44em;
1104        }
1105        p#header_p_right {
1106            font-size: 1.44em;
1107        }
1108        p#footer_p{
1109            font-size: 0.72em;
1110        }
1111        select#op_postposition {
1112           height: 21.6px;
1113           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;
        }
    }
```

```
1176
1177    @media all and (min-width:450px) and (max-height: 700px) {
            /*0.45*/
1178        #header{
1179            height:22.5px;
1180        }
1181        #header_left{
1182            height:22.5px;
1183        }
1184
1185        #header_right{
1186            height:22.5px;
1187        }
1188        #left {
1189            height:414px;
1190        }
1191
1192        #left_top {
1193            height:38.25px;
1194        }
1195
1196        #left_middle {
1197            height:112.5px;
1198        }
1199
1200        #left_bottom {
1201            height:130.5px;
1202        }
1203
1204        #left_bottom_bottom {
1205            height:159.75px;
```

```
1206        }
1207        #section {
1208            height:414px;
1209        }
1210
1211        #section_top {
1212            height:357.75px;
1213        }
1214
1215        #section_bottom {
1216            height:54px;
1217        }
1218
1219        #section_bottom_left {
1220            height:36px;
1221        }
1222
1223        #section_bottom_right {
1224            height:36px;
1225        }
1226
1227        #right {
1228            height:414px;
1229        }
1230
1231        #right_top {
1232            height:135px;
1233        }
1234
1235        #right_middle {
1236            height:135px;
```

```
1237        }
1238
1239
1240        #right_bottom {
1241            height:137.25px;
1242        }
1243
1244        #footer {
1245            height:9px;
1246        }
1247
1248        p#left_option {
1249            font-size: 0.675em;
1250        }
1251        p#section_top_p {
1252            font-size: 0.675em;
1253        }
1254        p#section_bottom_p {
1255            font-size: 0.675em;
1256        }
1257        p#right_option {
1258            font-size: 0.72em;
1259        }
1260        p#header_p_left {
1261            font-size: 0.9em;
1262        }
1263        p#header_p_right {
1264            font-size: 0.9em;
1265        }
1266        p#footer_p{
1267            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;
```

```
1299        }
1300        .track-overlay {
1301          stroke-width: 22.5px;
1302        }
1303        div.tooltip {
1304            font-size: 7.65px;
1305        }
1306        div.epoch {
1307            font-size: 18px;
1308        }
1309        #tooltip_top {
1310            width: 54px;
1311            font-size: 7.2px;
1312        }
1313        #tooltip_middle {
1314            width: 54px;
1315            font-size: 7.2px;
1316        }
1317        #tooltip_bottom {
1318            width: 72px;
1319        }
1320        #tooltip_bottom p {
1321            font-size: 7.2px;
1322        }
1323        h5 {
1324            font-size: 6.75px;
1325        }
1326
1327        text.rightbottom_text {
1328            font-size: 9px;
1329        }
```

```
        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>
```

```
1356                <div id="container_leftmiddle">
1357                </div>
1358            </div>
1359            <div id="left_bottom">
1360                <p id="left_option">Select sentence</p>
1361                <div id="container_leftbottom">
1362                </div>
1363            </div>
1364            <div id="left_bottom_bottom">
1365                <p id="left_option">Density cluster</p>
1366
1367            </div>
1368        </div>
1369        <div id="section">
1370            <div id="section_top">
1371                <p id="section_top_p">t-SNE visualization of
                        BERT sentence classification</p>
1372            </div>
1373            <div id="section_bottom">
1374                <p id="section_bottom_p">Current epoch</p>
1375                <div id="section_bottom_left">
1376                    <button id="play-button">Play</button>
1377                </div>
1378                <div id="section_bottom_right">
1379                </div>
1380            </div>
1381        </div>
1382        <div id="right">
1383            <div id="right_top">
1384                <p id="right_option">Overall accuracy & Loss
                        </p>
```

```
1385                <div id="tooltip_top" style='display:none'>
1386                </div>
1387            </div>
1388            <div id="right_middle">
1389                <p id="right_option">Individual accuracy</p>
1390                <div id="tooltip_middle" style='display:none'>
1391                </div>
1392            </div>
1393            <div id="right_bottom">
1394                <p id="right_option">Bar chart for density
                        cluster</p>
1395                <div id="tooltip_bottom" class="hidden">
1396                </div>
1397                </div>
1398            </div>
1399        </div>
1400        <div id="footer">
1401            <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>
1402        </div>
1403    <script>
1404    $(document).ready(function () {
1405
1406    var function_name = ['agt','crt','eff','fns','gol','ins','loc'
        ,'thm','src','dir'];
1407    var function_color = ['#993366','#FF99CC','#FF00FF','#FF6600',
        '#000080','#003300',' #0066CC','#666699','#FFCC00','#5b2e90
        '];
1408
```

```
1409 var LeftsectionWidth = $("#left_bottom_bottom").width()
1410 var LeftsectionHeight = $("#left_bottom_bottom").height()
1411
1412 var LeftsvgSection = d3.select('#left_bottom_bottom').append('
        svg')
1413 .attr('width', LeftsectionWidth)
1414 .attr('height', (LeftsectionHeight*0.9))
1415
1416 var imgs = LeftsvgSection.append("image")
1417 .attr("class", "PNG")
1418 .attr("xlink:href", "https://seongmin-mun.github.io/
        VisualSystem/Major/PostBERT.ko/images/densityClusterPNG_r/
        Ey_tSNE_epoch_0.png")
1419 .attr("x", LeftsectionWidth*0.05)
1420 .attr("y", 0)
1421 .attr('width', LeftsectionWidth*0.9)
1422 .attr('height', LeftsectionWidth*0.9);
1423
1424 var sectionWidth = $("#section_top").width()
1425 var sectionHeight = $("#section_top").height()
1426
1427 var svgSection = d3.select('#section_top').append('svg')
1428 .attr('width', sectionWidth)
1429 .attr('height', (sectionHeight*0.935))
1430 .call(d3.zoom().scaleExtent([0.5, 5]).on("zoom", function () {
1431 svgSection.attr("transform", d3.event.transform)
1432 }))
1433 .append("g");
1434
1435 var div_epoch = d3.select("#section").append("div")
1436 .attr("class", "epoch")
```

```
1437  .style("opacity", 0.8)
1438  .style("right", sectionWidth*0.03+"px")
1439  .style("top", sectionHeight*0.01+"px");
1440
1441  var textlabel = div_epoch.append("text")
1442  .attr("class", "textlabel")
1443  .attr("text-anchor", "middle")
1444  .text("1 Epoch")
1445  .attr("text-anchor", "end")
1446  .attr("font-family", "Open Sans")
1447  .attr("font-size", "25px")
1448  .attr("fill", "#C2C1C1")
1449
1450  var NodeGroup = svgSection.append("g");
1451
1452  var div_inner = d3.select("#section").append("div")
1453  .attr("class", "tooltip")
1454  .style("opacity", 0);
1455
1456  var right_top_width = $("#right_top").width()
1457  var right_top_height = $("#right_top").height()
1458
1459  var svgright_top = d3.select("#right_top")
1460  .append("svg")
1461  .attr("width", right_top_width*0.95)
1462  .attr("height", (right_top_height*0.95))
1463  .append('g')
1464  .attr('transform', 'translate(' + 0 + ',' + 0 + ')');
1465
1466  svgright_top.append("rect")
1467  .attr("class", "svgright_rect")
```

```
1468  .attr("x", right_top_width * 0.05)
1469  .attr("y", 0)
1470  .attr("width", right_top_width*0.9)
1471  .attr("height",(right_top_height*0.75))
1472  .attr("rx", 6)
1473  .attr("ry", 6)
1474  .attr("fill", "white")
1475  .attr('stroke', '#C2C1C1')
1476  .attr('stroke-width', '2')
1477
1478  var  epoch_right_top = ["0", "10", "20", "30", "40", "50"]
1479
1480  for (var k = 0; k < 6; k++) {
1481  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")
1482  }
1483
1484  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"))
1485
1486  for (var k = 0; k < 6; k++) {
1487  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"))
1488  }
1489
1490  var right_middle_width = $("#right_middle").width()
1491  var right_middle_height = $("#right_middle").height()
1492
1493  var svgright_middle = d3.select("#right_middle")
1494  .append("svg")
1495  .attr("width", right_middle_width*0.95)
1496  .attr("height", (right_middle_height*0.95))
1497
1498  svgright_middle.append("rect")
1499  .attr("class", "svgright_rect")
1500  .attr("x", right_middle_width * 0.05)
1501  .attr("y", 0)
1502  .attr("width", right_middle_width*0.9)
1503  .attr("height",(right_middle_height*0.75))
1504  .attr("rx", 6)
1505  .attr("ry", 6)
1506  .attr("fill", "white")
1507  .attr('stroke', '#C2C1C1')
1508  .attr('stroke-width', '2')
1509
1510  var  epoch_right_middle = ["0", "10", "20", "30", "40", "50"]
1511
1512  for (var k = 0; k < 6; k++) {
1513  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
        ")
1514 }
1515
1516 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"))
1517
1518 for (var k = 0; k < 6; k++) {
1519 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"))
1520 }
1521
1522 var right_bottom_width = $("#right_bottom").width()
1523 var right_bottom_height = $("#right_bottom").height()
1524
1525 var svgright_bottom = d3.select("#right_bottom")
1526 .append("svg")
1527 .attr("width", right_bottom_width*0.95)
1528 .attr("height", (right_bottom_height*0.95))
1529
1530 svgright_bottom.append("rect")
1531 .attr("class", "svgright_rect")
1532 .attr("x", right_bottom_width * 0.05)
1533 .attr("y", 0)
```

```
1534  .attr("width", right_bottom_width*0.9)
1535  .attr("height",(right_bottom_height*0.75))
1536  .attr("rx", 6)
1537  .attr("ry", 6)
1538  .attr("fill", "white")
1539  .attr('stroke', '#C2C1C1')
1540  .attr('stroke-width', '2')
1541
1542  var  epoch_right_bottom = ["0", "10", "20", "30", "40", "50"]
1543
1544  for (var k = 0; k < 6; k++) {
1545  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
        ")
1546  }
1547
1548  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"))
1549
1550  for (var k = 0; k < 6; k++) {
1551  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")
```

```
1581  .attr("transform", "translate(" + SB_width*0.05 + "," +
          SB_height/2 + ")");
1582
1583  slider.append("line")
1584  .attr("class", "track")
1585  .attr("x1", x.range()[0])
1586  .attr("x2", x.range()[1])
1587  .select(function() { return this.parentNode.appendChild(
          this.cloneNode(true)); })
1588  .attr("class", "track-inset")
1589  .select(function() { return this.parentNode.appendChild(
          this.cloneNode(true)); })
1590  .attr("class", "track-overlay")
1591  .call(d3.drag()
1592  .on("start.interrupt", function() { slider.interrupt(); })
1593  .on("start drag", function() {
1594  currentValue = d3.event.x;
1595  update(x.invert(currentValue));
1596  })
1597  );
1598
1599  slider.insert("g", ".track-overlay")
1600  .attr("class", "ticks")
1601  .attr("transform", "translate(0," + 18 + ")")
1602  .selectAll("text")
1603  .data(x.ticks(10))
1604  .enter().append("text")
1605  .attr("x", x)
1606  .attr("text-anchor", "middle")
1607  .text(d => d+" epoch" );
1608
```

```
1609 const handle = slider.insert("circle", ".track-overlay")
1610 .attr("class", "handle")
1611 .attr("r", 9);
1612
1613 d3.select(window)
1614 .on("keydown", keydowned);
1615
1616 playButton
1617 .on("click", paused)
1618 .each(paused);
1619
1620 function update(d) {
1621 target = d;
1622 moving = true;
1623 timer.restart(updateTween);
1624 drawall();
1625 }
1626
1627 function updateTween() {
1628 let diff = target - actual;
1629 if (Math.abs(diff) < 1e-3) actual = target, timer.stop();
1630 else actual += diff * alpha;
1631 handle.attr("cx", x(actual));
1632 currentEpoch = Math.round(actual)
1633
1634 return currentEpoch
1635
1636 }
1637
1638 function keydowned() {
1639 let currentValue = actual;
```

```
1640  if (d3.event.metaKey || d3.event.altKey) return;
1641  switch (d3.event.keyCode) {
1642  case 37: currentValue = Math.max(x.domain()[0], actual -
          trailLength); break;
1643  case 39: currentValue = Math.min(x.domain()[1], actual +
          trailLength); break;
1644  default: return;
1645  }
1646  update(currentValue);
1647  paused();
1648  }
1649
1650  function paused() {
1651  if (moving) {
1652  slider.interrupt();
1653  clearInterval(stepTimer);
1654  moving = false;
1655  playButton.text("Play");
1656  } else {
1657  if (actual > maxValue) actual = 0;
1658  stepTimer = setInterval(step, 1500);
1659  moving = true;
1660  playButton.text("Pause");
1661  }
1662  }
1663
1664  function step() {
1665  if (actual > maxValue) paused();
1666  else update(actual + trailLength / 10);
1667  }
1668
```

```
1669 paused();
1670 var functionslist_ey = ['LOC','GOL','EFF','CRT','THM','INS','
        AGT','FNS'];
1671 var functionsname_ey = ['Location','Goal','Effector','
        Criterion','Theme','Instrument','Agent','Final State'];
1672
1673 var functionslist_eyse = ['SRC','LOC'];
1674 var functionsname_eyse = ['Source','Location'];
1675
1676 var functionslist_lo = ['LOC','DIR','EFF','CRT','INS','FNS'];
1677 var functionsname_lo = ['Location','Direction','Effector','
        Criterion','Instrument','Final State'];
1678
1679 function draw_op_function_after(post,list,name,name_kr){
1680 $('#container_leftmiddle').empty();
1681 if (post === "ey"){
1682 for(var i = 0 ; i < list.length; i++){
1683 var color = ""
1684 var currentFunc = list[i].toLowerCase()
1685 if (currentFunc == function_name[0]) {
1686 color = function_color[0]
1687 } else if (currentFunc == function_name[1]) {
1688 color = function_color[1]
1689 } else if (currentFunc == function_name[2]) {
1690 color = function_color[2]
1691 } else if (currentFunc == function_name[3]) {
1692 color = function_color[3]
1693 } else if (currentFunc == function_name[4]) {
1694 color = function_color[4]
1695 } else if (currentFunc == function_name[5]) {
1696 color = function_color[5]
```

```
1697  } else if (currentFunc == function_name[6]) {
1698  color = function_color[6]
1699  } else if (currentFunc == function_name[7]) {
1700  color = function_color[7]
1701  } else if (currentFunc == function_name[8]) {
1702  color = function_color[8]
1703  } else if (currentFunc == function_name[9]) {
1704  color = function_color[9]
1705  }
1706  $("#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>");
1707  }
1708  } else if (post === "eyse"){
1709  for(var i = 0 ; i < list.length; i++){
1710  var color = ""
1711  var currentFunc = list[i].toLowerCase()
1712  if (currentFunc == function_name[0]) {
1713  color = function_color[0]
1714  } else if (currentFunc == function_name[1]) {
1715  color = function_color[1]
1716  } else if (currentFunc == function_name[2]) {
1717  color = function_color[2]
1718  } else if (currentFunc == function_name[3]) {
1719  color = function_color[3]
1720  } else if (currentFunc == function_name[4]) {
1721  color = function_color[4]
```

```
1722  } else if (currentFunc == function_name[5]) {
1723  color = function_color[5]
1724  } else if (currentFunc == function_name[6]) {
1725  color = function_color[6]
1726  } else if (currentFunc == function_name[7]) {
1727  color = function_color[7]
1728  } else if (currentFunc == function_name[8]) {
1729  color = function_color[8]
1730  } else if (currentFunc == function_name[9]) {
1731  color = function_color[9]
1732  }
1733  $("#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>");
1734  }
1735  } else if (post === "(u)lo"){
1736  for(var i = 0 ; i < list.length; i++){
1737  var color = ""
1738  var currentFunc = list[i].toLowerCase()
1739  if (currentFunc == function_name[0]) {
1740  color = function_color[0]
1741  } else if (currentFunc == function_name[1]) {
1742  color = function_color[1]
1743  } else if (currentFunc == function_name[2]) {
1744  color = function_color[2]
1745  } else if (currentFunc == function_name[3]) {
1746  color = function_color[3]
```

```
1747  } else if (currentFunc == function_name[4]) {
1748  color = function_color[4]
1749  } else if (currentFunc == function_name[5]) {
1750  color = function_color[5]
1751  } else if (currentFunc == function_name[6]) {
1752  color = function_color[6]
1753  } else if (currentFunc == function_name[7]) {
1754  color = function_color[7]
1755  } else if (currentFunc == function_name[8]) {
1756  color = function_color[8]
1757  } else if (currentFunc == function_name[9]) {
1758  color = function_color[9]
1759  }
1760  $("#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>");
1761  }
1762  }
1763  }
1764
1765  function op_function_change(){
1766  var selected_postposition = $( "#op_postposition" ).val();
1767  if (selected_postposition === "ey"){
1768  draw_op_function_after("ey",functionslist_ey,functionsname_ey,
        functionsname_kr_ey)
1769  draw_op_index_after("ey")
1770
```

```
1771 } else if (selected_postposition === "eyse"){
1772 draw_op_function_after("eyse",functionslist_eyse,
        functionsname_eyse,functionsname_kr_eyse)
1773 draw_op_index_after("eyse")
1774 } else if (selected_postposition === "(u)lo"){
1775 draw_op_function_after("(u)lo",functionslist_lo,
        functionsname_lo,functionsname_kr_lo)
1776 draw_op_index_after("(u)lo")
1777 }
1778
1779 drawall();
1780 }
1781
1782 function draw_op_index_after(post){
1783 $('#container_leftbottom').empty();
1784 if (post === "ey"){
1785 for(var i = 0 ; i < 467; i++){
1786 $("#container_leftbottom").append("<input class='CB_leftbottom
        ' type='checkbox' value='index"+i+"' id='CB_leftbottom_"+i+
        "' /> <label class='CB_leftbottom'>index_"+i+"</label></br>
        ");
1787 }
1788 } else if (post === "eyse"){
1789 for(var i = 0 ; i < 484; i++){
1790 $("#container_leftbottom").append("<input class='CB_leftbottom
        ' type='checkbox' value='index"+i+"' id='CB_leftbottom_"+i+
        "' /> <label class='CB_leftbottom'>index_"+i+"</label></br>
        ");
1791 }
1792 } else if (post === "(u)lo"){
1793 for(var i = 0 ; i < 467; i++){
```

```
1794  $("#container_leftbottom").append("<input class='CB_leftbottom
          ' type='checkbox' value='index"+i+"' id='CB_leftbottom_"+i+
          "' /> <label class='CB_leftbottom'>index_"+i+"</label></br>
          ");
1795  }
1796  }
1797  }
1798
1799  function functioncheckbox() {
1800
1801  var data_checkbox = []
1802
1803  var selected_postposition = $( "#op_postposition" ).val();
1804  if (selected_postposition === "ey"){
1805  for(var i = 0; i < functionslist_ey.length; i++){
1806  if(checkedeachValue('CB_leftmiddle_'+functionslist_ey[i].
          toLowerCase())!=='null'){
1807  data_checkbox.push(checkedeachValue('CB_leftmiddle_'+
          functionslist_ey[i].toLowerCase()));
1808  }
1809  }
1810
1811  } else if (selected_postposition === "eyse"){
1812  for(var i = 0; i < functionslist_eyse.length; i++){
1813  if(checkedeachValue('CB_leftmiddle_'+functionslist_eyse[i].
          toLowerCase())!=='null'){
1814  data_checkbox.push(checkedeachValue('CB_leftmiddle_'+
          functionslist_eyse[i].toLowerCase()));
1815  }
1816  }
1817  } else if (selected_postposition === "(u)lo"){
```

```
1818 for(var i = 0; i < functionslist_lo.length; i++){
1819 if(checkedeachValue('CB_leftmiddle_'+functionslist_lo[i].
        toLowerCase())!=='null'){
1820 data_checkbox.push(checkedeachValue('CB_leftmiddle_'+
        functionslist_lo[i].toLowerCase()));
1821 }
1822 }
1823 }
1824
1825 return data_checkbox;
1826 }
1827
1828 function indexcheckbox() {
1829
1830 var data_checkbox = []
1831
1832 var selected_postposition = $( "#op_postposition" ).val();
1833 if (selected_postposition === "ey"){
1834 for(var i = 0 ; i < 467; i++){
1835 if(checkedeachValue('CB_leftbottom_'+i)!=='null'){
1836 data_checkbox.push(checkedeachValue('CB_leftbottom_'+i));
1837 }
1838 }
1839
1840 } else if (selected_postposition === "eyse"){
1841 for(var i = 0 ; i < 484; i++){
1842 if(checkedeachValue('CB_leftbottom_'+i)!=='null'){
1843 data_checkbox.push(checkedeachValue('CB_leftbottom_'+i));
1844 }
1845 }
1846 } else if (selected_postposition === "(u)lo"){
```

```
1847 for(var i = 0 ; i < 467; i++){
1848 if(checkedeachValue('CB_leftbottom_'+i)!=='null'){
1849 data_checkbox.push(checkedeachValue('CB_leftbottom_'+i));
1850 }
1851 }
1852 }
1853
1854 return data_checkbox;
1855 }
1856
1857 function checkedeachValue(checkeddata){
1858 var value;
1859 var checkedValue = document.querySelector('#'+checkeddata+':
        checked');
1860 if(checkedValue == null){
1861 value = "null";
1862 } else {
1863 value = checkedValue.value;
1864 }
1865 return value;
1866 }
1867
1868 op_function_change()
1869
1870 function right_top_draw(){
1871 var selected_postposition = $( "#op_postposition" ).val();
1872
1873 EpochNow = updateTween()
1874
1875 EpochNoww = EpochNow
1876
```

```
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)
}

```

```
1938 var right_top_x = d3.scale.linear().domain([0,50]).range([
        right_top_width*0.07,right_top_width*0.9])
1939 var right_top_y = d3.scale.linear().domain([-0.25,1.5]).range
        ([right_top_height*0.7,0])
1940
1941 var line = d3.svg.line()
1942 .x(function(d){ return right_top_x(d.x)})
1943 .y(function(d){return right_top_y(d.y)})
1944 .interpolate("linear");
1945
1946 const tooltip_top = d3.select('#tooltip_top');
1947 const tooltipLine_top = svgright_top.append('line');
1948
1949 svgright_top.selectAll()
1950 .data(final_data).enter()
1951 .append('path')
1952 .attr('class','righttoppath')
1953 .attr('fill', 'none')
1954 .attr('stroke', d => d.color)
1955 .attr('stroke-width', 2)
1956 .datum(d => d.history)
1957 .attr('d', line)
1958 .attr("opacity",0.7);
1959
1960 svgright_top.selectAll()
1961 .data(final_data).enter()
1962 .append('text')
1963 .html(d => d.name)
1964 .attr('fill', d => d.color)
1965 .attr('alignment-baseline', 'middle')
1966 .attr('x', right_top_width)
```

```
1967  .attr('dx', '.5em')
1968  .attr('y', d => right_top_y(d.currentLoss));
1969
1970  tipBox = svgright_top.append('rect')
1971  .attr('width', right_top_width)
1972  .attr('height', right_top_height)
1973  .attr('opacity', 0)
1974  .on('mousemove', drawTooltip)
1975  .on('mouseout', removeTooltip);
1976
1977
1978  function removeTooltip() {
1979  if (tooltip_top) tooltip_top.style('display', 'none');
1980  if (tooltipLine_top) tooltipLine_top.attr('stroke', 'none');
1981  }
1982
1983  function drawTooltip() {
1984  const x = Math.floor((right_top_x.invert(d3.mouse(tipBox.node
        ())[0])+0.5));
1985
1986  final_data.sort((a, b) => {
1987  return b.history.find(h => h.x == x).y - a.history.find(h =>
        h.x == x).y;
1988  })
1989
1990  tooltipLine_top.attr('stroke', 'black')
1991  .attr('x1', right_top_x(x)-1)
1992  .attr('x2', right_top_x(x)-1)
1993  .attr('y1', 0)
1994  .attr('y2', right_top_height)
1995  .attr('opacity',0.7);
```

```
1996
1997  tooltip_top.html(x+1)
1998  .style('display', 'block')
1999  .style('right', right_top_width*0.07+"px")
2000  .style('top', right_top_height*0.28+"px")
2001  .style('opacity',0.7)
2002  .selectAll()
2003  .data(final_data).enter()
2004  .append('div')
2005  .style('color', d => d.color)
2006  .html(d => d.name + ': ' + d.history.find(h => h.x == x).y);
2007  }
2008  }
2009
2010  function right_middle_draw(){
2011  var selected_postposition = $( "#op_postposition" ).val();
2012
2013  EpochNow = updateTween()
2014
2015  EpochNoww = EpochNow
2016
2017  svgright_middle.selectAll(".rightmiddlepath").remove();
2018
2019  var dataTotal = []
2020
2021  for (var i = 0; i < Accuracy_info.length ; i++) {
2022  if ((Accuracy_info[i].postposition === selected_postposition))
          {
2023  for(var j = 0; j < Accuracy_info[i].accuracy.length ; j++){
2024  if(j<=EpochNoww){
2025  dataTotal.push(Accuracy_info[i].accuracy[j])
```

```
2026 }
2027 }
2028 }
2029 }
2030
2031 funcList = []
2032
2033 for (var key in dataTotal[0]){
2034 if((key === 'epoch')||(key === 'total')||(key === 'loss')||(
        key === 'correlation')){
2035 continue;
2036 } else {
2037 funcList.push(key)
2038 }
2039
2040 }
2041
2042 var final_data = []
2043
2044 for (var i = 0; i < funcList.length ; i++) {
2045 currentFunc = funcList[i]
2046 var dataT = {}
2047 var dataY = [];
2048 var currentaccuracy = 0
2049 for (var j = 0; j < dataTotal.length ; j++) {
2050 dataY.push(dataTotal[j][funcList[i]])
2051 currentaccuracy = dataTotal[j][funcList[i]]
2052 }
2053 dataT['y'] = dataY
2054 var dataX = [];
2055 for (var j = 0; j < 50 ; j++) {
```

```
2056  dataX.push(j)
2057  }
2058  dataT['x'] = dataX
2059
2060  var rearrangedData = dataT.x.map(function(d,i) {
2061  return {x:d,y:dataT.y[i]};
2062  })
2063
2064  var re_data = {}
2065  re_data['name'] = currentFunc
2066  re_data['show'] = true
2067  re_data['currentLoss'] = currentaccuracy
2068  re_data['history'] = rearrangedData
2069
2070  if (currentFunc == function_name[0]) {
2071  re_data['color'] = function_color[0]
2072  } else if (currentFunc == function_name[1]) {
2073  re_data['color'] = function_color[1]
2074  } else if (currentFunc == function_name[2]) {
2075  re_data['color'] = function_color[2]
2076  } else if (currentFunc == function_name[3]) {
2077  re_data['color'] = function_color[3]
2078  } else if (currentFunc == function_name[4]) {
2079  re_data['color'] = function_color[4]
2080  } else if (currentFunc == function_name[5]) {
2081  re_data['color'] = function_color[5]
2082  } else if (currentFunc == function_name[6]) {
2083  re_data['color'] = function_color[6]
2084  } else if (currentFunc == function_name[7]) {
2085  re_data['color'] = function_color[7]
2086  } else if (currentFunc == function_name[8]) {
```

```
2087 re_data['color'] = function_color[8]
2088 } else if (currentFunc == function_name[9]) {
2089 re_data['color'] = function_color[9]
2090 }
2091
2092 final_data.push(re_data)
2093
2094 }
2095
2096 var right_middle_x = d3.scale.linear().domain([0,50]).range([
        right_middle_width*0.07,right_middle_width*0.9])
2097 var right_middle_y = d3.scale.linear().domain([-0.25,1.1]).
        range([right_middle_height*0.72,0])
2098
2099 var line = d3.svg.line()
2100 .x(function(d){ return right_middle_x(d.x)})
2101 .y(function(d){return right_middle_y(d.y)})
2102 .interpolate("linear");
2103
2104 const tooltip_middle = d3.select('#tooltip_middle');
2105 const tooltipLine_middle = svgright_middle.append('line');
2106
2107 svgright_middle.selectAll()
2108 .data(final_data).enter()
2109 .append('path')
2110 .attr('class','rightmiddlepath')
2111 .attr('fill', 'none')
2112 .attr('stroke', d => d.color)
2113 .attr('stroke-width', 2)
2114 .datum(d => d.history)
2115 .attr('d', line)
```

```
2116  .attr("opacity",0.7);
2117
2118  svgright_middle.selectAll()
2119  .data(final_data).enter()
2120  .append('text')
2121  .html(d => d.name)
2122  .attr('fill', d => d.color)
2123  .attr('alignment-baseline', 'middle')
2124  .attr('x', right_middle_width)
2125  .attr('dx', '.5em')
2126  .attr('y', d => right_middle_y(d.currentLoss));
2127
2128  tipBox = svgright_middle.append('rect')
2129  .attr('width', right_middle_width)
2130  .attr('height', right_middle_height)
2131  .attr('opacity', 0)
2132  .on('mousemove', drawTooltip)
2133  .on('mouseout', removeTooltip);
2134
2135  function removeTooltip() {
2136  if (tooltip_middle) tooltip_middle.style('display', 'none');
2137  if (tooltipLine_middle) tooltipLine_middle.attr('stroke', '
        none');
2138  }
2139
2140  function drawTooltip() {
2141  const x = Math.floor((right_middle_x.invert(d3.mouse(
        tipBox.node())[0])+0.5));
2142
2143  final_data.sort((a, b) => {
```

```
2144  return b.history.find(h => h.x == x).y - a.history.find(h =>
          h.x == x).y;
2145  })
2146
2147  tooltipLine_middle.attr('stroke', 'black')
2148  .attr('x1', right_middle_x(x)-1)
2149  .attr('x2', right_middle_x(x)-1)
2150  .attr('y1', 0)
2151  .attr('y2', right_middle_height)
2152  .attr('opacity',0.7);
2153
2154  tooltip_middle.html(x+1)
2155  .style('display', 'block')
2156  .style('right', right_top_width*0.07+"px")
2157  .style('top', right_middle_height*0.28+"px")
2158  .style('opacity',0.7)
2159  .selectAll()
2160  .data(final_data).enter()
2161  .append('div')
2162  .style('color', d => d.color)
2163  .html(d => d.name + ': ' + d.history.find(h => h.x == x).y);
2164  }
2165  }
2166
2167  function right_bottom_draw(){
2168  var selected_postposition = $( "#op_postposition" ).val();
2169
2170  EpochNow = updateTween()
2171
2172  EpochNoww = EpochNow
2173
```

```
2174 svgright_bottom.selectAll(".rightbottom_bar").remove();
2175 svgright_bottom.selectAll(".rightbottom_text").remove();
2176
2177 var dataTotal = [0]
2178
2179 for (var i = 0; i < Density_info.length ; i++) {
2180 if ((Density_info[i].postposition === selected_postposition))
        {
2181 for(var j = 0; j < Density_info[i].cluster.length ; j++){
2182 if(j<=EpochNoww){
2183 dataTotal.push(Density_info[i].cluster[j].clusterNumber)
2184 }
2185 }
2186 }
2187 }
2188
2189 yScaleMax = 0;
2190 if(selected_postposition==="ey"){
2191 yScaleMax = 7;
2192 } else if (selected_postposition==="eyse"){
2193 yScaleMax = 3;
2194 } else {
2195 yScaleMax = 7;
2196 }
2197
2198 var right_bottom_x = d3.scale.linear()
2199 .domain([0,50])
2200 .range([right_bottom_width*0.05,right_bottom_width*0.88], 0.1)
        ;
2201
2202 var right_bottom_y = d3.scale.linear()
```

```
2203  .domain([0, yScaleMax])
2204  .range([0,right_bottom_height*0.62]);
2205
2206  svgright_bottom.selectAll("rect")
2207  .data(dataTotal)
2208  .enter()
2209  .append("rect")
2210  .attr("class","rightbottom_bar")
2211  .attr("x", function(d, i) {
2212  return right_bottom_x(i);
2213  })
2214  .attr("y", function(d) {
2215  return (right_bottom_height*0.6) - right_bottom_y(d);
2216  })
2217  .attr("width", right_bottom_width*0.015)
2218  .attr("height", function(d) {
2219  return right_bottom_y(d);
2220  })
2221  .attr("fill", function(d) {
2222  return "#666666";
2223  })
2224  .on("mouseover", function(d, i) {
2225
2226  d3.select("#tooltip_bottom")
2227  .style("right", right_bottom_width*0.07 + "px")
2228  .style("top",  right_bottom_height*0.28 + "px")
2229  .html("<p><strong>Selected epoch: </strong>"+i+"<br><strong>
         Cluster number: </strong>"+d+"</p>")
2230
2231  d3.select("#tooltip_bottom").classed("hidden", false);
2232
```

```
2233 })
2234 .on("mouseout", function() {
2235
2236 d3.select("#tooltip_bottom").classed("hidden", true);
2237
2238 });
2239
2240 svgright_bottom.append("text")
2241 .attr("class","rightbottom_text")
2242 .text("Current cluster number: "+dataTotal[EpochNow+1])
2243 .attr("x", right_bottom_width*0.08)
2244 .attr("y", right_bottom_height*0.1)
2245 .attr("text-anchor", "start")
2246 .attr("font-family", "Open Sans")
2247 .attr("font-size", "20px")
2248 .attr("fill", "#666666")
2249 }
2250
2251 firstdrawdata();
2252
2253 d3.selectAll("#op_postposition").on("change",
        op_function_change);
2254 d3.selectAll("#container_leftmiddle").on("change", drawall);
2255 d3.selectAll("#container_leftbottom").on("change", drawall);
2256
2257 function drawall(){
2258
2259 var selected_postposition = $( "#op_postposition" ).val();
2260
2261 var functions = functioncheckbox();
2262 var indexs = indexcheckbox();
```

```
2263  changedrawdata(selected_postposition,functions,indexs)
2264
2265  }
2266
2267  function firstdrawdata() {
2268  var data = {};
2269  for (var i = 0; i < Map_info.length ; i++) {
2270  if ((Map_info[i].postposition === 'ey') && (Map_info[i].epoch
        === 'epoch0')) {
2271  var innerArray = [];
2272  for(var j = 0; j < Sentence_info.length ; j++){
2273  if (Sentence_info[j].postposition === 'ey') {
2274  for(var k = 0; k < Map_info[i].sentences.length ; k++){
2275  var sentenceDic = {}
2276  sentenceDic["index"] = Sentence_info[j].sentences[k].index
2277  sentenceDic["function"] = Sentence_info[j].sentences[k].
        function
2278  sentenceDic["sentence"] = Sentence_info[j].sentences[k].
        sentence
2279  sentenceDic["sentence_pos"] = Sentence_info[j].sentences[k].
        sentence_pos
2280  sentenceDic["X"] = Map_info[i].sentences[k].X
2281  sentenceDic["Y"] = Map_info[i].sentences[k].Y
2282  sentenceDic["opacity_value"] = 0.7
2283  innerArray.push(sentenceDic)
2284  }
2285  }
2286  }
2287  data["postposition"] = Map_info[i].postposition
2288  data["epoch"] = Map_info[i].epoch
2289  data["sentences"] = innerArray
```

```
2290 }
2291 }
2292
2293 var w = sectionWidth;
2294 var h = sectionHeight;
2295 var padding = (sectionHeight*0.12);
2296
2297 var xScale = d3.scale.linear()
2298 .domain([d3.min(data.sentences, function(d) { return d.Y; }),
        d3.max(data.sentences, function(d) { return d.X; })])
2299 .range([0+padding, w-padding]);
2300
2301 var yScale = d3.scale.linear()
2302 .domain([d3.min(data.sentences, function(d) { return d.Y; }),
        d3.max(data.sentences, function(d) { return d.Y; })])
2303 .range([h-padding, 0+padding]);
2304
2305 NodeGroup.selectAll(".nodedot")
2306 .data(data.sentences)
2307 .enter()
2308 .append("circle")
2309 .attr("class", "nodedot")
2310 .attr("id", function (d) {
2311 return d.index
2312 })
2313 .attr("cx", function (d) {
2314 return xScale(d.X)
2315 })
2316 .attr("cy", function (d) {
2317 return yScale(d.Y)
2318 })
```

```
2319  .attr("r", 3)
2320  .attr("fill", function (d) {
2321  if (d.function == function_name[0]) {
2322  return function_color[0]
2323  } else if (d.function == function_name[1]) {
2324  return function_color[1]
2325  } else if (d.function == function_name[2]) {
2326  return function_color[2]
2327  } else if (d.function == function_name[3]) {
2328  return function_color[3]
2329  } else if (d.function == function_name[4]) {
2330  return function_color[4]
2331  } else if (d.function == function_name[5]) {
2332  return function_color[5]
2333  } else if (d.function == function_name[6]) {
2334  return function_color[6]
2335  } else if (d.function == function_name[7]) {
2336  return function_color[7]
2337  } else if (d.function == function_name[8]) {
2338  return function_color[8]
2339  } else if (d.function == function_name[9]) {
2340  return function_color[9]
2341  }
2342  })
2343  .attr("stroke", "black")
2344  .attr("stroke-width", "1px")
2345  .attr("opacity", function (d) {
2346  return d.opacity_value
2347  })
2348  .style("cursor", "help")
2349  .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]) {
```

```
2381      return function_color[8]
2382  } else if (d.function == function_name[9]) {
2383      return function_color[9]
2384  }
2385  });
2386  })
2387  .on("mouseenter", function (d) {
2388  div_inner.transition()
2389  .duration(200)
2390  .style("opacity", 0.85);
2391  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/>")
2392  .style("left", "20px")
2393  .style("top", sectionHeight*0.07+"px");
2394  })
2395  .on("mouseleave", function () {
2396  div_inner.transition()
2397  .duration(500)
2398  .style("opacity", 0);
2399  });
2400  }
2401
2402  function changedrawdata(selected_postposition,functionarray,
        indexarray) {
2403  if ((selected_postposition === 'ey') || (selected_postposition
         === '(u)lo')){
2404  for(var i = 467; i < 484 ; i++){
2405  svgSection.selectAll("#index"+i).remove();
2406  }
```

```
2407 }
2408
2409 EpochNow = updateTween()
2410
2411 right_top_draw();
2412 right_middle_draw();
2413 right_bottom_draw();
2414
2415 var textlabel = div_epoch.selectAll(".textlabel")
2416 textlabel.enter()
2417 .append("text")
2418 .attr("class", "label")
2419 .text((EpochNow+1)+" Epoch")
2420 textlabel.transition()
2421 .duration(10)
2422 .text((EpochNow+1)+" Epoch")
2423 textlabel.exit().remove();
2424
2425 var data = {};
2426 for (var i = 0; i < Map_info.length ; i++) {
2427 if ((Map_info[i].postposition === selected_postposition) && (
        Map_info[i].epoch === 'epoch'+EpochNow)) {
2428 var innerArray = [];
2429 for(var j = 0; j < Sentence_info.length ; j++){
2430 if (Sentence_info[j].postposition === selected_postposition &&
         (Map_info[i].epoch === 'epoch'+EpochNow)) {
2431 for(var k = 0; k < Map_info[i].sentences.length ; k++){
2432 var sentenceDic = {}
2433 sentenceDic["index"] = Sentence_info[j].sentences[k].index
2434 sentenceDic["function"] = Sentence_info[j].sentences[k].
        function
```

```
2435 sentenceDic["sentence"] = Sentence_info[j].sentences[k].
        sentence
2436 sentenceDic["sentence_pos"] = Sentence_info[j].sentences[k].
        sentence_pos
2437 sentenceDic["X"] = Map_info[i].sentences[k].X
2438 sentenceDic["Y"] = Map_info[i].sentences[k].Y
2439 if((functionarray.length > 0 == true)&&(indexarray.length > 0
        == true)){
2440
2441 var checked = false;
2442 for(var q = 0; q < functionarray.length ; q++){
2443 if(Sentence_info[j].sentences[k].function == functionarray[q])
        {
2444 checked = true;
2445 }
2446 }
2447
2448 if(checked == true){
2449 var ichecked = false;
2450 for(var t = 0; t < indexarray.length ; t++){
2451 if(Sentence_info[j].sentences[k].index == indexarray[t]){
2452 ichecked = true;
2453 }
2454 }
2455 if(ichecked == true){
2456 sentenceDic["opacity_value"] = 0.9
2457 } else {
2458 sentenceDic["opacity_value"] = 0.2
2459 }
2460 } else {
2461 sentenceDic["opacity_value"] = 0.2
```

```
2462 }
2463 } else if((functionarray.length > 0 == false)&&(
         indexarray.length > 0 == true)){
2464 var checked = false;
2465 for(var q = 0; q < indexarray.length ; q++){
2466 if(Sentence_info[j].sentences[k].index == indexarray[q]){
2467 checked = true;
2468 }
2469 }
2470 if(checked == true){
2471 sentenceDic["opacity_value"] = 0.9
2472 } else {
2473 sentenceDic["opacity_value"] = 0.2
2474 }
2475 } else if((functionarray.length > 0 == true)&&(
         indexarray.length > 0 == false)){
2476 var checked = false;
2477 for(var q = 0; q < functionarray.length ; q++){
2478 if(Sentence_info[j].sentences[k].function == functionarray[q])
         {
2479 checked = true;
2480 }
2481 }
2482 if(checked == true){
2483 sentenceDic["opacity_value"] = 0.9
2484 } else {
2485 sentenceDic["opacity_value"] = 0.2
2486 }
2487 } else {
2488 sentenceDic["opacity_value"] = 0.7
2489 }
```

```
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);
```

```
2519
2520 var w = sectionWidth;
2521 var h = sectionHeight;
2522 var padding = (sectionHeight*0.12);
2523
2524 var xScale = d3.scale.linear()
2525 .domain([d3.min(data.sentences, function(d) { return d.Y; }),
        d3.max(data.sentences, function(d) { return d.X; })])
2526 .range([0+padding, w-padding]);
2527
2528 var yScale = d3.scale.linear()
2529 .domain([d3.min(data.sentences, function(d) { return d.Y; }),
        d3.max(data.sentences, function(d) { return d.Y; })])
2530 .range([h-padding, 0+padding]);
2531
2532 var circle = NodeGroup.selectAll(".nodedot")
2533 .data(data.sentences);
2534
2535 circle.enter()
2536 .append("circle")
2537 .attr("class", "nodedot")
2538 .attr("id", function (d) {
2539 return d.index
2540 })
2541 .attr("cx", function (d) {
2542 return xScale(d.X)
2543 })
2544 .attr("cy", function (d) {
2545 return yScale(d.Y)
2546 })
2547 .attr("r", 3)
```

```
2548  .attr("fill", function (d) {
2549  if (d.function == function_name[0]) {
2550  return function_color[0]
2551  } else if (d.function == function_name[1]) {
2552  return function_color[1]
2553  } else if (d.function == function_name[2]) {
2554  return function_color[2]
2555  } else if (d.function == function_name[3]) {
2556  return function_color[3]
2557  } else if (d.function == function_name[4]) {
2558  return function_color[4]
2559  } else if (d.function == function_name[5]) {
2560  return function_color[5]
2561  } else if (d.function == function_name[6]) {
2562  return function_color[6]
2563  } else if (d.function == function_name[7]) {
2564  return function_color[7]
2565  } else if (d.function == function_name[8]) {
2566  return function_color[8]
2567  } else if (d.function == function_name[9]) {
2568  return function_color[9]
2569  }
2570  })
2571  .attr("stroke", "black")
2572  .attr("stroke-width", "1px")
2573  .attr("opacity", function (d) {
2574  return d.opacity_value
2575  })
2576  .style("cursor", "help")
2577  .on("mouseover", function (d) {
2578  d3.select(this)
```

```
2579  .attr("stroke", "black")
2580  .attr("stroke-width", "1px")
2581  .attr("opacity", 1)
2582  .attr("fill","#FF0000")
2583  })
2584  .on("mouseout", function (d) {
2585  d3.select(this)
2586  .attr("stroke", "black")
2587  .attr("stroke-width", "1px")
2588  .attr("opacity", function (d) {
2589  return d.opacity_value
2590  })
2591  .attr("fill", function (d) {
2592  if (d.function == function_name[0]) {
2593  return function_color[0]
2594  } else if (d.function == function_name[1]) {
2595  return function_color[1]
2596  } else if (d.function == function_name[2]) {
2597  return function_color[2]
2598  } else if (d.function == function_name[3]) {
2599  return function_color[3]
2600  } else if (d.function == function_name[4]) {
2601  return function_color[4]
2602  } else if (d.function == function_name[5]) {
2603  return function_color[5]
2604  } else if (d.function == function_name[6]) {
2605  return function_color[6]
2606  } else if (d.function == function_name[7]) {
2607  return function_color[7]
2608  } else if (d.function == function_name[8]) {
2609  return function_color[8]
```

```
2610  } else if (d.function == function_name[9]) {
2611  return function_color[9]
2612  }
2613  });
2614  })
2615  .on("mouseenter", function (d) {
2616  div_inner.transition()
2617  .duration(200)
2618  .style("opacity", 0.85);
2619  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/>")
2620  .style("left", "20px")
2621  .style("top", sectionHeight*0.07+"px");
2622  })
2623  .on("mouseleave", function () {
2624  div_inner.transition()
2625  .duration(500)
2626  .style("opacity", 0);
2627  })
2628
2629  circle.transition()
2630  .duration(2000)
2631  .attr("cx", function (d) {
2632  return xScale(d.X)
2633  })
2634  .attr("cy", function (d) {
2635  return yScale(d.Y)
2636  })
2637  .attr("r", 3)
```

```
2638  .attr("fill", function (d) {
2639  if (d.function == function_name[0]) {
2640  return function_color[0]
2641  } else if (d.function == function_name[1]) {
2642  return function_color[1]
2643  } else if (d.function == function_name[2]) {
2644  return function_color[2]
2645  } else if (d.function == function_name[3]) {
2646  return function_color[3]
2647  } else if (d.function == function_name[4]) {
2648  return function_color[4]
2649  } else if (d.function == function_name[5]) {
2650  return function_color[5]
2651  } else if (d.function == function_name[6]) {
2652  return function_color[6]
2653  } else if (d.function == function_name[7]) {
2654  return function_color[7]
2655  } else if (d.function == function_name[8]) {
2656  return function_color[8]
2657  } else if (d.function == function_name[9]) {
2658  return function_color[9]
2659  }
2660  })
2661  .attr("stroke", "black")
2662  .attr("stroke-width", "1px")
2663  .attr("opacity", function (d) {
2664  return d.opacity_value
2665  })
2666  .style("cursor", "help")
2667  .on("mouseover", function (d) {
2668  d3.select(this)
```

```
2669  .attr("stroke", "black")
2670  .attr("stroke-width", "1px")
2671  .attr("opacity", 1)
2672  .attr("fill","#FF0000")
2673  })
2674  .on("mouseout", function (d) {
2675  d3.select(this)
2676  .attr("stroke", "black")
2677  .attr("stroke-width", "1px")
2678  .attr("opacity", function (d) {
2679  return d.opacity_value
2680  })
2681  .attr("fill", function (d) {
2682  if (d.function == function_name[0]) {
2683  return function_color[0]
2684  } else if (d.function == function_name[1]) {
2685  return function_color[1]
2686  } else if (d.function == function_name[2]) {
2687  return function_color[2]
2688  } else if (d.function == function_name[3]) {
2689  return function_color[3]
2690  } else if (d.function == function_name[4]) {
2691  return function_color[4]
2692  } else if (d.function == function_name[5]) {
2693  return function_color[5]
2694  } else if (d.function == function_name[6]) {
2695  return function_color[6]
2696  } else if (d.function == function_name[7]) {
2697  return function_color[7]
2698  } else if (d.function == function_name[8]) {
2699  return function_color[8]
```

```
2700 } else if (d.function == function_name[9]) {
2701 return function_color[9]
2702 }
2703 });
2704 })
2705 .on("mouseenter", function (d) {
2706 div_inner.transition()
2707 .duration(200)
2708 .style("opacity", 0.85);
2709 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/>")
2710 .style("left", "20px")
2711 .style("top", sectionHeight*0.07+"px");
2712 })
2713 .on("mouseleave", function () {
2714 div_inner.transition()
2715 .duration(500)
2716 .style("opacity", 0);
2717 });
2718
2719 circle.exit().remove();
2720 }
2721 })
2722 </script>
2723 </body>
2724 </html>
```

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 rep-

resentations. 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 se-jong 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.

Université Paris Nanterre
École doctorale 139 – Connaissance, Langage, Modélisation

La résolution de la polysémie à l'aide de modèles de vecteur de mots et la visualisation de données : le cas des postpositions adverbiales *-ey*, *-eyse*, et *-(u)lo* en coréen

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

Résumé

Ce projet de thèse présente des comptes rendus informatiques de la résolution de la polysémie au niveau des mots dans une langue peu étudiée—le Coréen. Les postpositions, qui se caractérisent par une correspondance forme-fonction multiple et qui sont donc polysémiques par nature, posent un défi à l'analyse automatique et à la performance des modèles pour identifier leurs fonctions. Dans ce projet, je consolide les modèles existants de classification de vecteur au niveau du mot (*Positive Pointwise Mutual Information* et *Singular Value Decomposition* ; *Skip-Gram and Negative Sampling*) en tenant compte du Window du contexte, et j'introduis un modèle de classification de vecteur au niveau de la phrase (*Bidirectional Encoder Representations from Transformers* (BERT)) dans le cadre de la modélisation sémantique distributionnelle. Par ailleurs, je développe deux systèmes de visualisation qui montrent (i) les relations entre les postpositions et leurs mots co-occurrents pour les modèles de vecteur au niveau du mot, et (ii) les clusters entre les phrases pour le modèle de vecteur au niveau de la phrase. Ces systèmes de visualisation ont l'avantage de mieux comprendre comment ces modèles de classification classent les fonctions prévues de ces postpositions. Les résultats montrent que, alors que la performance des modèles de vecteur au niveau du mot est modulée par la taille des corpus d'entraînement contenant les fonctions spécifiques des postpositions, le modèle de vecteur au niveau des phrases est stable (i.e., moins affecté par la taille du corpus) et simule la façon dont les humains reconnaissent la polysémie des postpositions adverbiales coréennes de façon plus appropriée que les modèles de vecteur au niveau du mot.

Mots-clés : polysémie, traitement automatique des langues, classification, modèles de vecteur de mots, visualisation de données, Coréen

1 Introduction

La polysémie, qui est un type d'ambiguïté, se produit lorsqu'une forme exprime des significations/fonctions multiples mais pourtant liées (Glynn and Robinson, 2014). Ce type de relation se retrouve également en coréen, une langue de la forme Sujet-Objet-Verbe avec une marquage explicite des cas par le biais d'une postposition dédiée (i.e., un morphème lié qui ajoute une signification grammaticale à un mot contenu où il est attaché ; Sohn, 1999). Une postposition coréenne implique normalement de nombreuses correspondances entre la forme et la fonction, et est de ce fait polysémique (Choo and Kwak, 2008). Par exemple, une postposition adverbiale *-ey*, l'une des postpositions étudiées dans cette thèse, est interprétée comme ayant huit fonctions majeures : localisation (LOC), but (GOL), effecteur (EFF), critère (CRT), thème (THM), instrument (INS), agent (AGT), et état final (FNS) (Shin, 2008). Supposons que la phrase suivante implique la postposition *-ey* comme fonction de LOC (localisation) tel que dans (1). Les locuteurs natifs du coréen (ou une personne ayant une bonne connaissance du coréen) peuvent facilement comprendre la fonction prévue de *-ey*.

(1) 지붕 위에 구멍이 났다.
cipung wi-ey kwumeng-i na-ss-ta.
Toit le dessus-LOC trou-NOM Il y a-PST-DECL
« Il y a un trou sur le dessus du toit. »

À cet égard, une question se pose quant à la manière dont un locuteur comprend la fonction de *-ey* en tant que LOC, compte tenu de ces diverses fonctions. Considérant que la signification d'un mot est étroitement liée à un contexte qui est créé par un groupe de mots voisins (DSMs ; Harris, 1954), cette question peut être résolue à travers le réseau de mots qui représente la relation entre les mots. Sur

la base de ce concept, plusieurs études ont cherché à saisir et à distinguer les différentes significations/fonctions des postpositions coréennes en appliquant des approches informatiques (e.g., Bae et al., 2014, 2015, Kim and Ock, 2016, Lee et al., 2015, Shin et al., 2005). Toutefois, les études antérieures ne se sont concentrées que sur l'amélioration des performances des modèles et n'ont pas essayé de comprendre comment ces modèles de classification classent les fonctions prévues de ces postpositions. Face à ce constat, je cherche, dans le cadre de cette thèse de doctorat, à appliquer des approches informatiques pour résoudre les problèmes que pose à la polysémie de ces postpositions au niveau du mot. De plus, afin de mieux comprendre comment ces modèles de classification classent les fonctions des postpositions, j'implémente des systèmes de visualisation qui montrent les vecteurs des mots et des phrases pour chaque modèle.

2 Contexte

2.1 Les modèles sémantiques distributionnels (DSMs)

L'idée fondamentale des DSMs repose sur le fait que la signification d'un mot est étroitement liée au contexte qui est créé par un groupe de mots voisins (Bullinaria and Levy, 2007, Turney and Pantel, 2010). Cette idée découle des premiers travaux en linguistique théorique de Harris (1954) et Firth (1957). Harris (1954) affirme que *les mots qui apparaissent dans des contextes similaires ont tendance à avoir des significations similaires*, tandis que Firth (1957) affirme que l'*on connaît un mot par son context*. Par exemple, *maison* et *appartement* apparaissent fréquemment avec des mots de contexte comme *loyer*, *chambre*, *vente*, etc., ce qui prouve aux modèles informatiques que *maison* et *appartement* peuvent être similaires l'un à l'autre. Il est important de noter que de nombreuses études ont signalé la force des modèles sémantiques distributionnels pour résoudre la polysémie du niveau des mots (e.g., Bae et al., 2015, Lee et al., 2015, Mun and Shin, 2020, Shin et al., 2005).

Dans son application réelle, les DSMs convertissent en vecteurs les informations contextuelles obtenues grâce aux mots situés autour d'un mot cible. Ils appliquent ensuite des algorithmes d'apprentissage automatique à ces vecteurs afin de mesurer la similarité sémantique de mot (e.g., Clark, 2015, Erk, 2012, Turney and Pantel, 2010). Les DSMs sont composés de deux types de modèles de vecteur de mot. Le premier est un modèle basé sur le comptage (e.g., Singular Value Decomposition (SVD) : Eckart and Young, 1936) qui est sensible à la fréquence des tokens (Jurafsky and Martin, 2019). Le second est un modèle basé sur la prédiction (e.g., Skip-Gram and Negative Sampling (SGNS) : Mikolov et al., 2013) qui s'appuie sur la fréquence des types (Mikolov et al., 2013). Les études antérieures ont montré que ces modèles de vecteur de mot ont l'avantage de représenter la relation entre les mots (e.g., Bae et al., 2015, Lee et al., 2015, Shin et al., 2005).

En plus de ces modèles traditionnels de vecteur de mot, des études récentes ont proposé un modèle de vecteur de mot contextualisé qui prend en compte les informations de voisinage d'un mot polysémique sur la base des séquences de mots autour du mot cible. Divers modèles ont été proposés pour cette tâche, tels que les *Embeddings de Language Models* (e.g., Peters et al., 2018), le *Generative Pre-Training* (e.g., Radford et al., 2018), et les *Bidirectional Encoder Representations of Transformer* (BERT ; Devlin et al., 2018). Parmi ces modèles, BERT montre les meilleures performances dans de nombreuses tâches telles que la traduction, la classification et la réponse à des questions (e.g., Devlin et al., 2018, Tang et al., 2019).

Sur la base de ces antécédents, j'utilise dans cette thèse une une combinaison de *Positive Pointwise Mutual Information* (PPMI ; Church and Hanks, 1989) et de la *Singular Value Decomposition* (SVD ; Eckart and Young, 1936) comme modèle basé sur le nombre, et du *Skip-Gram and Negative Sampling* (SGNS ; Mikolov et al., 2013) comme modèle basé sur la prédiction pour les modèles traditionnels de vecteur des mots. En outre, j'ai choisi BERT comme modèle de vecteur de motint *contextualisés* pour la tâche de

classification visant à identifier la fonction prévue d'une postposition dans une phrase.

2.2 La Polysémie dans les postpositions adverbiales coréennes

Le coréen, qui est la langue qui nous intéresse dans cette thèse, est une langue Sujet-Objet-Verbe avec le marquage explicite des cas par le biais d'une postposition—un morphème lié qui ajoute des fonctions grammaticales à un mot de contenu où il est attaché (Sohn, 1999). Les postpositions coréennes sont divisées en deux catégories : (i) grammaticale, indiquant les relations syntaxiques entre les mots du contenu et (ii) sémantique, indiquant les fonctions spécifiques selon le contexte de la phrase particulière (Sohn, 1999). Plus précisément, les postpositions adverbiales (classées comme postposition sémantique) sont polysémiques en raison de leur correspondance multiple entre forme et fonction, qui s'accompagne d'une ambiguïté fonctionnelle (Choo and Kwak, 2008). Parmi les diverses postpositions adverbiales, dans cette thèse, je me restreins à trois postpositions adverbiales : *-ey*, *-eyse* et *-(u)lo*, qui sont fréquemment utilisées en coréen et donc souvent documentées dans les études précédentes (e.g., Cho and Kim, 1996, Jeong, 2010, Nam, 1993, Park, 1999, Song, 2014).

2.2.1 *-ey*

Les études précédentes ont examiné les fonctions de *-ey* et ont proposé leurs propres affirmations concernant les types de fonctions impliquant *-ey*. Par exemple, Cho and Kim (1996) ont classé 10 types. Nam (1993) a affirmé que la relation entre un (pro-)nom et un prédicat combiné avec une postposition est importante pour déterminer sa fonction, ce qui a donné 14 types de postposition.

Afin de déterminer le nombre de fonctions de cette postposition, cette thèse met particulièrement l'accent sur huit fonctions majeures de *-ey* : localisation, but, effecteur, critère, thème, instrument, agent, et état final, qui sont fréquemment attestées dans le corpus Sejong, le corpus représentatif du Coréen.

2.2.2 *-eyse*

-eyse a moins de fonctions que les deux autres postpositions *-ey* et *-(u)lo* (Choo and Kwak, 2008). Cependant, la fréquence de son utilisation est également élevée par rapport à celle des autres (e.g., Cho and Kim, 1996, Song, 2014). Les chercheurs s'accordent généralement sur la fonction primaire comme étant le lieu qui s'engage dans le départ de l'action (e.g., Cho and Kim, 1996, Park et al., 2000, Song, 2014) et le corpus Sejong démontre également la même tendance. Les deux fonctions (source et lieu) sont majoritairement plus fréquentes que les autres.

2.2.3 *-(u)lo*

Les études antérieures ont examiné les fonctions de *-(u)lo* et ont proposé différents points de vue sur le nombre de fonctions pour *-(u)lo*. À titre d'exemple, Park (1999) affirme que la fonction centrale est instrumentale et qu'il existe neuf autres fonctions, telles que le chemin, la direction, le point de direction, le temps, le changement d'état, la qualification, le matériel, la cause et la manière. En revanche, Jeong (2010) place la fonction directionnelle au centre des différentes fonctions et explique la relation entre la fonction centrale et les fonctions étendues.

La classification du projet Sejong est quelque peu différente de ces deux études, puisqu'elle indique qu'il existe six fonctions majeures de *-(u)lo* : état final, instrument, direction, effecteur, critère, et localisation, avec les trois principales (état final ; instrument ; direction) occupant plus de 80% de l'utilisation totale. Car le corpus Sejong est largement utilisé dans les études sur le Coréen (e.g., Kang and Park, 2003, Kim et al., 2007, Park and Cha, 2017, Shin et al., 2005).

2.3 Les précédentes recherches en NLP sur les postpositions adverbiales

Les études sur la polysémie du niveau des mots en Coréen se sont principalement concentrées sur la catégorisation des différentes significations/fonctions des mots polysémiques pour l'interprétation essentielle des phénomènes linguistiques (e.g., Ahn, 1983, Hong, 1978, Lee, 1983, Maeng, 2016). Les chercheurs travaillant sur la linguistique informatique en Coréen suivent cette tendance et développent des systèmes qui classifient et reconnaissent automatiquement ces multiples significations/fonctions impliquant les mots afin de traiter les outils linguistiques d'une manière plus facile et plus efficace (e.g., Bae and Lee, 2015, Kang and Park, 2003, Kim and Ock, 2015, Lee et al., 2015, Shin et al., 2005). Par exemple, Lee et al. (2015) ont employé un SVM pour proposer un système d'étiquetage des rôles sémantiques. Dans cette étude, 4 096 phrases ont été utilisées pour l'apprentissage et 786 phrases ont été utilisées pour le test, ce qui a permis d'obtenir une précision de 0.77 pour la classification.

Malgré de nombreuses recherches sur la postposition adverbiale en Coréen, elles se sont surtout concentrées sur l'amélioration de la précision de la classification des fonctions et n'ont pas prêté attention à l'environnement des postpositions, comme les mots co-occurrents, qui génèrent un cluster centré autour de la postposition. D'un point de vue linguistique, une relation de groupes de mots liés entre eux est sans aucun doute une ressource linguistique précieuse car elle montre comment la polysémie est interprétée à travers eux. À cet égard, les modèles sémantiques distributionnels (DSMs ; Baroni et al., 2014), qui soutiennent que la signification d'un mot est étroitement liée à un contexte créé par un groupe de mots voisins, attirent l'attention sur la compréhension informatique dans le langage humain (Bullinaria and Levy, 2007, Turney and Pantel, 2010). Dans cette thèse, j'adopte l'idée que les DSMs fournissent des clusters entre le mot cible et les mots co-occurrents. Sur la base de ces DSM, j'améliore les approches précédentes de cette tâche en créant des modèles de classification basés sur des vecteur du niveau du mot—*Positive Pointwise Mutual Information* et *Singular Value Decomposition* (PPMI-SVD ; Turney and Pantel, 2010) et *Skip-Gram and Negative Sampling* (SGNS ; Mikolov et al., 2013)—ainsi que sur des vecteurs du niveau de la phrase—le *Bidirectional Encoder Representations of Transformers* (BERT ; Devlin et al., 2018). En outre, pour mieux comprendre comment ces modèles de classification reconnaissent les fonctions prévues des postpositions, ce projet met en œuvre des systèmes de visualisation qui montrent les relations des mots et des phrases pour chaque modèle.

3 Mise en place méthodologique : PPMI-SVD et SGNS

3.1 Corpus

3.1.1 Création du corpus annoté

Dans cette thèse, j'utilise le corpus de données représentatif du Coréen connu sous le nom de corpus Sejong (Kim et al., 2006). Cependant, le corpus Sejong ne code pas directement l'information sur les fonctions des postpositions dans chaque phrase (ce qui est nécessaire pour l'entraînement du modèle). Par conséquent, j'annote le corpus manuellement avec l'aide de trois locuteurs natifs du Coréen. Parmi les trois, l'un était un professeur qui enseigne le coréen aux enfants et les deux autres étaient des candidats au doctorat en linguistique. Ils ont géré tous les détails de l'annotation du corpus, depuis le développement du manuel d'annotation jusqu'à l'annotation manuelle de la fonction prévue de la postposition dans chaque phrase.

En ce qui concerne le processus de création d'un corpus codé à la main, j'extrais les phrases n'ayant qu'une seule postposition et un seul prédicat. Bien que cette manipulation ait permis d'omettre de nombreuses phrases déjà extraites du corpus original, elle a été bénéfique pour contrôler tout facteur de

confusion supplémentaire qui aurait pu interférer avec les performances de mon modèle. Si une phrase contient plus d'une postposition, y compris les trois postpositions sur lesquelles je me suis attardé, elles deviennent moins indépendantes les unes des autres. Cela signifie que les performances du modèle de chaque postposition seront affectées les unes par les autres. Ce processus de réduction a donné lieu à un total de 27,720 phrases, dont 14,096 phrases pour *-ey*, 5,078 phrases pour *-eyse* et 8,546 phrases pour *-(u)lo*. J'extrais ensuite 5,000 phrases au hasard pour chaque postposition afin de conserver un nombre égal de phrases pour chacune d'entre elles.

Tableau 1 – Liste de fréquence des sous-fonctions de *-ey*, *-eyse*, et *-(u)lo* dans le corpus validé par croisement

-ey		*-eyse*		*-(u)lo*	
Fonction	Fréquence	Fonction	Fréquence	Fonction	Fréquence
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

Les données du corpus final sont ensuite codées à la main par les trois locuteurs natifs du coréen, en suivant les fonctions des différentes postpositions. La fiabilité inter-juges des données a été mesurée avec le Fleiss's Kappa (Landis and Koch, 1977). Les résultats sont un score de 0.948 pour *-ey*, 0.928 pour *-eyse*, et 0.947 pour *-(u)lo*, qui sont considérés comme '*presque parfaits*' selon l'échelle de Kappa. Je décide ensuite d'exclure les phrases qui ont provoqué un désaccord entre les annotateurs humains (c'est-à-dire 285 phrases pour *-ey*, 147 phrases pour *-eyse*, et 292 phrases pour *-(u)lo*). Après lequel, j'obtiens les données du corpus final pour chaque postposition. Cela a donné 4,715 phrases pour *-ey*, 4,853 phrases pour *-eyse*, et 4,708 phrases pour *-(u)lo*. Le Tableau 1 présente la liste détaillée des fréquences par fonction des trois types de postpositions [1].

3.1.2 Création des ensembles de formation et de test

Chaque instance du corpus annoté a été lemmatisée et marquée par l'étiquetage morpho-syntaxiquement (aussi appelé étiquetage grammatical, POS tagging (part-of-speech tagging) en anglais) avant l'étape de traitement des données proprement dite. L'utilisation du corpus pour cette tâche exige que les fonctions de chaque postposition soient marquées ouvertement avec la forme de chaque postposition (e.g., *ᄋᆡ/JKB_CRT*). Par conséquent, je marque les fonctions des postpositions manuellement.

Les données pour la formation et le test doivent être indépendantes les unes des autres. Ainsi, je divise le corpus en deux sous-ensembles, l'un avec 90% du corpus pour la formation et les 10% restants pour les tests. Afin d'obtenir un résultat normalisé de chaque modèle, j'utilise la technique de *validation croisée à k blocs* (Salton, 1971), qui évalue le modèle en partitionnant le corpus original en *k* sous-échantillons de taille égale. Je fixe la valeur de *k* à 10 et je répète la validation croisée 10 fois, avec chacun des 10 sous-échantillons utilisés exactement une fois comme le jeu de test.

1. Le corpus codé à la main est disponible à l'adresse suivante : https://github.com/seongmin-mun/Corpora/tree/master/APIK

3.2 Formation du modèle

Pour le modèle de classification, j'ai employé PPMI-SVD (Turney and Pantel, 2010) et SGNS (Mikolov et al., 2013) sur la base de l'estimation basée sur la similarité (Dagan et al., 1995) pour le dressage du modèle, en suivant un modèle sémantique distributionnel (DSM; Baroni et al., 2014). L'entraînement du modèle se compose de deux parties : (i) vecteur au niveau des mots pour vérifier la relation entre les mots, et (ii) l'estimation basée sur la similarité (Dagan et al., 1995) pour déterminer les fonctions prévues de la postposition utilisée dans l'ensemble de test.

3.2.1 Des vecteurs au niveau des mots : PPMI-SVD et SGNS

Le flux général pour des vecteurs au niveau du mot est le suivant. Tout d'abord, le modèle crée une liste de mots qui existent dans les ensembles de formation obtenus par la technique de validation croisée à 10 blocs. Deuxièmement, sur la base de la liste de mots, une matrice de co-occurrence mot-mot (pour le modèle basé sur le comptage) et des vecteurs à un coup (pour le modèle basé sur la prédiction) sont générés. Troisièmement, le modèle produit des vecteurs au niveau du mot en utilisant PPMI-SVD et SGNS.

Le premier algorithme des vecteurs au niveau du mot a été développé dans un environnement Python. *Linalg* du package *scipy* a été utilisé pour la formation du modèle PPMI-SVD. *Word2Vec*, du package *gensim*, a été utilisé pour la formation du modèle SGNS. Les vecteurs au niveau des mots générés par chaque modèle avaient 500 dimensions, chacune d'entre elles étant stockée dans une base de données. Un total de 600 vecteurs a été réalisé par cet algorithme (2 modèles * 3 postpositions * 10 plis * 10 tailles de Window).

3.2.2 Estimation basée sur la similarité

Sur la base des vecteurs au niveau du mot généré par le premier algorithme, le second algorithme a été développé pour classifier la fonction prévue des postpositions utilisées dans l'ensemble de test. Ceci a été fait en calculant l'estimation basée sur la similarité (Dagan et al., 1995) : classer le sens du mot cible qui n'a jamais été utilisé dans les ensembles d'entraînement en utilisant les scores de similarité calculés entre les mots. Dans cette thèse, j'ai utilisé la formule de similarité en cosinus pour calculer le score de similarité entre une postposition et ses mots co-occurrents.

L'algorithme pour l'estimation basée sur la similarité se déroule comme suit. Tout d'abord, l'algorithme charge un total de 600 vecteurs au niveau du mot (2 modèles * 3 postpositions * 10 plis * 10 tailles de Window) générés par le premier algorithme et calcule la similarité entre les postpositions et les mots environnants. Deuxièmement, l'algorithme charge un jeu de test et établit la liste des mots qu'il contient. Troisièmement, l'algorithme compare la liste de mots utilisée dans l'ensemble de test à celle utilisée dans l'ensemble d'apprentissage et génère une liste de mots qui sont partagés entre eux. Quatrièmement, l'algorithme calcule le score moyen entre chaque fonction des postpositions et une liste de mots qui sont partagés entre eux. Enfin, l'algorithme détermine la fonction des postpositions utilisée dans le jeu de test avec la moyenne la plus élevée [2].

3.3 Visualisation : PostEmbedding

Afin d'interpréter intuitivement les clusters entre les postpositions et les mots qui les entourent, j'ai développé un système de visualisation (disponible sur le site : PostEmbedding). Dans le but d'exprimer

2. Le code complet des modèles des vecteurs au niveau du mot que j'ai développés sont disponible sur le site : PPMI-SVD et SGNS

les vecteurs au niveau du mot des DSM dans la visualisation bidimensionnelle, j'ai utilisé le *t-distributed Stochastic Neighbor Embedding* (t-SNE ; Maaten and Hinton, 2008) pour la réduction de la dimension des vecteurs au niveau du mot. Ces résultats ont été introduits dans le système de visualisation. Le système a été développé à l'aide des environnements JavaScript, HTML et CSS [3].

4 Résultats : les vecteurs au niveau du mot

4.1 Performance du modèle : Classification

4.1.1 PPMI-SVD

Les Tableaux suivants (Tableaux 2-4) montrent la précision de classification du modèle PPMI-SVD pour chaque postposition. Les résultats montrent que le modèle est plus performant pour *-eyse* que pour les deux autres postpositions (*-ey* et *-(u)lo*). La précision moyenne de classification pour *-ey*, *-eyse* et *-(u)lo* est d'environ 0.534, 0.773 et 0.567 respectivement.

Tableau 2 – Précision par fonction pour le modèle PPMI-SVD : *-ey*

taille du Window	Précision de classification								
	Overall	*AGT*	*CRT*	*EFF*	*FNS*	*GOL*	*INS*	*LOC*	*THM*
1	0.470	0.675	0.551	0.453	0.438	0.555	0.317	0.396	0.430
3	0.432	0.625	0.438	0.558	0.448	0.609	0.317	0.380	0.377
5	0.554	0.575	0.585	0.584	0.329	0.561	0.250	0.607	0.359
7	0.597	0.575	0.588	0.537	0.267	0.489	0.183	0.765	0.298
9	0.600	0.475	0.576	0.532	0.257	0.491	0.167	0.780	0.323
10	0.600	0.475	0.580	0.532	0.267	0.493	0.183	0.776	0.318
Moyenne	0.534	0.578	0.544	0.541	0.337	0.538	0.238	0.602	0.345

Note. Abréviation : AGT= agent ; CRT= critère ; EFF= effecteur ; FNS= état final ; GOL= but ; INS= instrument ; LOC= localisation ; THM= thème

Tableau 3 – Précision par fonction pour le modèle PPMI-SVD : *-eyse*

taille du Window	Précision de classification		
	Overall	*LOC*	*SRC*
1	0.634	0.627	0.678
3	0.648	0.643	0.680
5	0.809	0.876	0.367
7	0.859	0.970	0.130
9	0.859	0.980	0.062
10	0.856	0.977	0.062
Moyenne	0.773	0.838	0.353

Note. Abréviation : LOC= localisation ; SRC= source

3. Plus de détails sur le PostEmbedding sont disponibles sur le site : PostEmbedding

Tableau 4 – Précision par fonction pour le modèle PPMI-SVD : *-(u)lo*

taille du Window	Précision de classification						
	Overall	*CRT*	*DIR*	*EFF*	*FNS*	*INS*	*LOC*
1	0.480	0.497	0.547	0.462	0.425	0.491	0.353
3	0.532	0.552	0.730	0.562	0.394	0.482	0.347
5	0.572	0.426	0.840	0.438	0.530	0.348	0.220
7	0.608	0.313	0.855	0.312	0.714	0.248	0.140
9	0.608	0.262	0.833	0.238	0.767	0.221	0.140
10	0.607	0.257	0.830	0.238	0.770	0.220	0.127
Moyenne	0.567	0.405	0.777	0.391	0.583	0.344	0.233

Note. Abréviation : CRT= critère ; DIR= direction ; EFF= effecteur ; FNS= état final ; INS= instrument ; LOC= localisation

L'analyse statistique des comparaisons en couple (Tableau 5) a également montré que la performance en *-eyse* était significativement meilleure que celle des deux autres postpositions. En revanche, l'exactitude de *-ey* et *-(u)lo* étaient statistiquement les mêmes.

Tableau 5 – Comparaison statistique de chaque postposition (PPMI-SVD) : t-test à deux échantillons

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

Note. *** $<$ .001

Comme le montrent les Tableaux (Tableaux 2-4), pour la relation entre la taille du Window de contexte et les performances du modèle PPMI-SVD, j'ai constaté que la précision du modèle PPMI-SVD augmentait avec la taille du Window de contexte.

En outre, j'ai constaté que la performance des modèles de fonctions pour chaque postposition variait. La précision moyenne de classification de chaque fonction pour *-ey* est la plus élevée en LOC (0.602) et la plus faible en INS (0.238) ; pour *-eyse*, elle est la plus élevée en LOC (0.838) et la plus faible en SRC (0.353) ; pour *-(u)lo*, elle est la plus élevée en DIR (0.777) et la plus faible en LOC (0.233) (Tableaux 2-4). Si l'on considère que LOC pour *-ey*, LOC pour *-eyse* et DIR pour *-(u)lo* représentent la plus grande partie du corpus entier que les autres fonctions (voir Tableau 1), on peut indiquer que la performance du modèle PPMI-SVD a été affectée par la taille du corpus de chaque fonction.

4.1.2 SGNS

Comme pour le modèle PPMI-SVD, *-eyse* a surclassé les deux autres postpositions dans le modèle SGNS, comme le montrent les Tableaux (Tableaux 6-8). Cela s'est produit pour la même raison que pour le modèle PPMI-SVD (i.e., *-eyse* n'a que deux fonctions avec LOC qui occupent la majorité de la taille totale du corpus). La précision moyenne de classification pour *-ey*, *-eyse* et *-(u)lo* est d'environ 0.204, 0.693 et 0.368 respectivement.

Tableau 6 – Précision par fonction pour le modèle SGNS : *-ey*

taille du Window	Précision de classification								
	Overall	*AGT*	*CRT*	*EFF*	*FNS*	*GOL*	*INS*	*LOC*	*THM*
1	0.170	0.675	0.052	0.458	0.167	0.625	0.417	0.132	0.073
3	0.224	0.925	0.145	0.558	0.162	0.693	0.117	0.166	0.102
5	0.220	0.925	0.095	0.626	0.233	0.639	0.150	0.196	0.093
7	0.201	0.875	0.079	0.637	0.238	0.577	0.150	0.175	0.089
9	0.188	0.825	0.071	0.632	0.290	0.564	0.150	0.148	0.086
10	0.187	0.900	0.059	0.584	0.281	0.564	0.183	0.154	0.095
Moyenne	0.204	0.878	0.089	0.593	0.224	0.616	0.183	0.169	0.092

Note. Abréviation : AGT= agent ; CRT= critère ; EFF= effecteur ; FNS= état final ; GOL= but ; INS= instrument ; LOC= localisation ; THM= thème

Tableau 7 – Précision par fonction pour le modèle SGNS : *-eyse*

taille du Window	Précision de classification		
	Overall	*LOC*	*SRC*
1	0.244	0.131	0.988
3	0.612	0.578	0.834
5	0.735	0.736	0.727
7	0.829	0.881	0.491
9	0.849	0.918	0.394
10	0.851	0.919	0.406
Moyenne	0.693	0.699	0.655

Note. Abréviation : LOC= localisation ; SRC= source

Tableau 8 – Précision par fonction pour le modèle SGNS : *-(u)lo*

taille du Window	Précision de classification						
	Overall	*CRT*	*DIR*	*EFF*	*FNS*	*INS*	*LOC*
1	0.345	0.507	0.769	0.550	0.011	0.147	0.247
3	0.396	0.554	0.874	0.538	0.039	0.154	0.313
5	0.374	0.468	0.797	0.662	0.072	0.124	0.420
7	0.362	0.455	0.739	0.700	0.087	0.121	0.480
9	0.348	0.461	0.688	0.725	0.076	0.132	0.553
10	0.349	0.492	0.695	0.725	0.060	0.137	0.540
Moyenne	0.368	0.499	0.774	0.634	0.058	0.141	0.427

Note. Abréviation : CRT= critère ; DIR= direction ; EFF= effecteur ; FNS= état final ; INS= instrument ; LOC= localisation

Pourtant, contrairement aux résultats du modèle PPMI-SVD, l'analyse statistique des comparaisons en couple (Tableau 9) montre que les niveaux d'exactitude de tous les postpositions étaient différents.

En ce qui concerne le rôle de la taille du Window contextuelle dans le modèle SGNS, contrairement aux résultats du modèle PPMI-SVD, le taux de changement de la précision par postpositions semble stable, sauf pour *-eyse*, sans changement significatif lorsque la taille du Window augmente. Toutefois, comme le montre le Tableau 7, pour *-eyse*, le modèle SGNS présente une tendance similaire à celle du modèle

Tableau 9 – Comparaison statistique de chaque postposition (SGNS) : t-test à deux échantillons

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

Note. *** $<$ .001

PPMI-SVD, à savoir que la précision de chaque modèle augmente avec la taille du Window contextuelle.

De plus, comme pour le modèle PPMI-SVD, la performance du modèle SGNS varie également en fonction des fonctions de chaque postposition. La précision de classification moyenne de chaque fonction pour *-ey* est la plus élevée pour AGT (0.878) et la plus faible pour CRT (0.089) ; pour *-eyse*, elle est la plus élevée pour LOC (0.699) et la plus faible pour SRC (0.655) ; pour *-(u)lo*, elle est la plus élevée pour DIR (0.774) et la plus faible pour FNS (0.058) (Tableaux 6-8). Comme dans le cas du modèle PPMI-SVD, on peut également constater que la performance du modèle SGNS est affectée par la taille du corpus de chaque fonction.

4.2 Système de visualisation : clusters et mots co-occurrents

Le système de visualisation visait à identifier des vecteurs au niveau du mot de manière interactive afin de voir les changements des clusters entre chaque fonction des postpositions et les mots co-significatifs. Pour explorer statistiquement les changements des clusters par modèle et par taille de Window, j'ai effectué une série d'analyses de cluster en utilisant le clustering basé sur la densité (Sander et al., 1998).

Les résultats de la visualisation ont montré que les mots les plus fréquemment utilisés dans le corpus ont été placés au centre du cluster pour le modèle PPMI-SVD. C'est parce qu'il fonctionnait sur la base de la fréquence des jetons. En revanche, le modèle SGNS était basé sur la fréquence de type, et les mots ont été distribués sur toutes les tailles de Window, indépendamment de la fréquence des jetons. Toutefois, l'analyse des grappes a montré que les cartes sémantiques de distribution pour chaque modèle n'étaient pas tellement différentes en ce qui concerne le produit final du regroupement (produisant un ou deux groupes pour chaque modèle, ce qui indique que les grappes créées ne différaient pas significativement les unes des autres par environnement.

En outre, grâce à cette visualisation, j'ai découvert qu'il existait deux types de relations différentes entre la postposition particulière et ses mots co-occurrents : (i) les mots présentant une forte similarité mais une faible fréquence de co-occurrence, et (ii) les mots présentant une forte similarité et également une fréquence de co-occurrence élevée.

4.3 Question des modèles du vecteur au niveau du mot

Malgré ces résultats, les deux modèles que j'ai testés dans cette section présentent de sérieuses limitations. La performance des modèles est insatisfaisante au regard des études précédentes sur la classification des postpositions sur lesquelles je me suis concentré (e.g., Bae et al., 2015, Kim and Ock, 2016, Shin et al., 2005). Ils ont rapporté un niveau de précision allant de 0.882 (Kang and Park, 2003) à 0.623 (Bae et al., 2014). En revanche, le niveau moyen pour mes modèles était de 0.550. En outre, le modèle semble bien fonctionner uniquement lorsque les fonctions cibles apparaissent très fréquemment dans les données, ce qui n'est pas la façon dont je visais à traiter la résolution de la polysémie. Cela est dû à la nature technique de le vecteur du niveau du mot, qui distingue les mots présents dans l'ensemble du corpus en utilisant uniquement les informations morphologiques et la taille du Window, et qui utilise

les mots sans tenir compte de leurs éventuels effets différents sur la détermination de la signification d'une postposition particulière. En effet, les modèles du vecteur au niveau du mot traditionnels sont *statiques*—un vecteur unique est attribué à chaque mot (e.g., Ethayarajh, 2019, Liu et al., 2019a).

Pour surmonter ces problèmes, j'ai employé BERT (Devlin et al., 2018) pour la classification des fonctions des postpositions. BERT produit des vecteurs contextuels, et cette caractéristique peut nous aider à créer un meilleur système de classification des postpositions. Une tendance récente pour traiter cette tâche est appelée vecteur contextuel du mot, qui convertit tous les mots dans chaque vecteur en considérant le contexte (e.g., la position, une forme du mot) dans lequel ils apparaissent. Divers modèles ont été proposés, tels que *Embeddings from Language Models* (ELMo ; Peters et al., 2018) et *Generative Pre-Training* (GPT ; Radford et al., 2018), mais BERT montre la meilleure performance parmi tous les modèles présentés jusqu'à présent. Par conséquent, j'ai appliqué BERT à mon modèle de classification afin d'améliorer les performances du modèle.

5 Mise en place méthodologique : BERT

5.1 Corpus

J'ai prétraité les données en tenant compte du fonctionnement du BERT (j'ai utilisé le modèle original du BERT pour cette tâche). Tout d'abord, j'ai ajouté [CLS] ('classification' ; indiquant le début d'une phrase) avant une phrase et [SEP] ('séparation' ; indiquant la fin d'une phrase) après une phrase pour indiquer où la phrase commence et se termine. Ces indicateurs ont permis au modèle BERT de reconnaître une limite de phrase dans un texte, permettant au modèle d'apprendre le sens des mots en tenant compte des variations inter-sententielles. Ensuite, j'ai créé une colonne séparée ('Label') pour indiquer la fonction prévue de chaque postposition dans chaque phrase. J'ai par la suite divisé le corpus en deux sous-ensembles, l'un avec 90% du corpus pour l'entraînement et l'autre avec les 10% restants pour les tests.

5.2 Formation du modèle

J'ai défini les paramètres liés à l'entraînement de BERT tels que *batch size* (32), *epoch* (50), *seed* (42), *epsilon* (0.00000008), et *learning rate* (0.00002), comme conseillé par McCormick (2019). J'ai ensuite employé un modèle linguistique pré-entraîné afin d'obtenir une grande précision des résultats ; à cette fin, j'ai utilisé un modèle BERT coréen (KoBERT ; Jeon et al., 2019).

Ensuite, l'entraînement du modèle s'est déroulé comme suit. Tout d'abord, j'ai chargé KoBERT par le biais de la fonction *BertForSequenceClassification* de *Transformers* (Wolf et al., 2019). Deuxièmement, j'ai affiné le modèle pré-entraîné en utilisant l'ensemble d'entraînement, en vue de réduire les valeurs de perte et de mettre à jour le taux d'apprentissage pour une meilleure précision de classification du modèle. Troisièmement, j'ai chargé l'ensemble de test pour évaluer si le modèle affiné a reconnu avec succès les fonctions prévues de chaque postposition dans chaque phrase. Dans cette partie, les taux de précision pour chaque fonction et le taux de précision total ont été calculés en comparant la fonction prévue de chaque postposition dans chaque phrase test avec la fonction classée de chaque postposition via le modèle BERT. Enfin, j'ai employé *t-SNE* pour la réduction de la dimension des vecteurs de classification de la postposition par chaque *epoch*. En outre, pour confirmer statistiquement les changements des résultats de vecteur du niveau des phrases pour chaque *epoch*, j'ai effectué un *density-based clustering*. Ces résultats ont été introduits dans le système de visualisation [4].

4. Le code complet de la formation du BERT que j'ai développé est disponible sur le site : BERT

5.3 Visualisation : PostBERT

Afin de voir comment le BERT comprend la polysémie du niveau des mots de chaque postposition, j'ai conçu un système de visualisation avec des environnements JavaScript, HTML et CSS, en utilisant l'ensemble de test sous la distribution bidimensionnelle (disponible sur le site : PostBERT) [5]. Pour l'interface du système, j'ai créé trois zones pour la démonstration des performances du modèle : une carte de distribution pour les vecteurs au niveau de la phrase, des graphiques de précision/perte relatifs au modèle, et des graphiques pour le *density-based clustering*.

6 Résultats : Vecteur au niveau de la phrase

6.1 Performance du modèle : Classification

Les tableaux suivants (Tableaux 10-12) montrent la précision de classification du modèle BERT pour chaque postposition. Les résultats montrent que le modèle BERT a mieux fonctionné pour *-eyse*, qui n'a que deux fonctions (SRC et LOC), que pour les deux autres postpositions (*-ey* et *-(u)lo*). La précision moyenne de classification pour *-ey*, *-eyse* et *-(u)lo* est d'environ 0.815, 0.898 et 0.813 respectivement. Il s'agit d'un niveau de précision satisfaisant si l'on considère que les études précédentes sur la classification des postpositions ont rapporté un niveau de précision allant de 0.621 (Bae et al., 2014) à 0.837 (Kim and Ock, 2016).

Tableau 10 – Précision par fonction pour le modèle BERT : *-ey*

Epoch	Précision de classification								
	Overall	*AGT*	*CRT*	*EFF*	*FNS*	*GOL*	*INS*	*LOC*	*THM*
1	0.682	0	0.876	0	0	0.044	0	0.911	0.198
10	0.819	0	0.930	0.433	0.578	0.313	0.133	0.954	0.688
20	0.817	0.067	0.897	0.533	0.533	0.186	0.067	0.960	0.916
30	0.824	0.067	0.915	0.378	0.444	0.328	0.067	0.948	0.718
40	0.826	0.067	0.892	0.489	0.467	0.326	0.133	0.942	0.768
50	0.824	0.067	0.912	0.411	0.389	0.409	0.1	0.940	0.683
Moyenne	0.815	0.041	0.911	0.439	0.497	0.328	0.076	0.947	0.713

Note. Abréviation : AGT= agent ; CRT= critère ; EFF= effecteur ; FNS= état final ; GOL= but ; INS= instrument ; LOC= localisation ; THM= thème

5. Plus de détails sur le PostEmbedding sont disponibles sur le site : PostBERT

Tableau 11 – Précision par fonction pour le modèle BERT : *-eyse*

Epoch	Précision de classification		
	Overall	*LOC*	*SRC*
1	0.863	0.980	0.174
10	0.9	0.939	0.559
20	0.898	0.937	0.651
30	0.896	0.949	0.464
40	0.912	0.963	0.523
50	0.916	0.960	0.598
Moyenne	0.898	0.948	0.535

Note. Abréviation : LOC= localisation ; SRC= source

Tableau 12 – Précision par fonction pour le modèle BERT : *-(u)lo*

Epoch	Précision de classification						
	Overall	*CRT*	*DIR*	*EFF*	*FNS*	*INS*	*LOC*
1	0.704	0.476	0.943	0	0.764	0.477	0
10	0.814	0.83	0.918	0.367	0.771	0.835	0.1
20	0.812	0.694	0.951	0.3	0.838	0.709	0.044
30	0.816	0.708	0.941	0.333	0.811	0.752	0.05
40	0.819	0.694	0.927	0.267	0.855	0.777	0.05
50	0.821	0.692	0.957	0.4	0.836	0.723	0.1
Moyenne	0.813	0.721	0.938	0.278	0.815	0.763	0.106

Note. Abréviation : CRT= critère ; DIR= direction ; EFF= effecteur ; FNS= état final ; INS= instrument ; LOC= localisation

Pour explorer statistiquement la classification par postpositions/epochs, j'ai effectué un *t*-test à deux échantillons. Comme le montre le Tableau 13, la performance du modèle pour *-eyse* est significativement meilleure que pour les deux autres postpositions. Compte tenu du nombre différent de fonctions (e.g., deux pour *-eyse*, six pour *-(u)lo*, et huit pour *-ey*), ce résultat indique une relation inverse entre la précision de la classification et le nombre de fonctions que chaque postposition manifeste.

Tableau 13 – Comparaison statistique de chaque postposition (BERT) : t-test à deux échantillons

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

Note. *** $<$ .001

De plus, la précision moyenne de classification de chaque fonction pour *-ey* est la plus élevée pour LOC (0.947) et la plus faible pour AGT (0.041) ; pour *-eyse*, elle est la plus élevée pour LOC (0.948) et la plus faible pour SRC (0.535) ; pour *-(u)lo*, elle est la plus élevée pour DIR (0.938) et la plus faible pour LOC (0.106) (Tableaux 10-12). Quant aux occurrences des fonctions individuelles par postposition, LOC pour *-ey*, LOC pour *-eyse*, et DIR pour *-(u)lo* représentent la plus grande partie du corpus entier que les autres fonctions (voir Tableau 1). Ce résultat indique donc que la performance du modèle a été affectée par les

proportions asymétriques des fonctions composant l'utilisation de chaque postposition.

6.2 Système de visualisation : clusters du vecteur au niveau de la phrase

Le système de visualisation a montré que le modèle était capable de reconnaître les fonctions de chaque postposition au fur et à mesure de la progression de l'epoch. Pour *-ey*, toutes les phrases étaient divisées en deux groupes lorsque l'epoch était la première, mais au fur et à mesure que l'epoch progressait, les phrases étaient divisées en trois à l'epoch 7, quatre à l'epoch 12 et cinq à l'epoch 15. Pour *-eyse*, le nombre de groupes était de un lorsque l'epoch était la première, et il y avait deux groupes lorsque l'epoch était la neuvième. Pour *-(u)lo*, le nombre de clusters a augmenté, passant de un (Epoch 1) à trois (Epoch 4), cinq (Epoch 12), et six (Epoch 46).

En particulier, pour *-(u)lo*, j'ai fait deux découvertes intéressantes. Tout d'abord, à l'epoch 12, un groupe de fonctions EFF (fonctions dont les occurrences sont peu fréquentes dans les données) est apparu. Ce résultat indique que l'ORET peut identifier des fonctions à un niveau satisfaisant, même si elles sont relativement peu fréquentes, à condition que le nombre d'epochs fournies soit suffisant.

Deuxièmement, il est intéressant de noter que LOC n'a pas pu former un groupe désigné au final. En mettant en évidence et en zoomant sur les instances individuelles de LOC, j'ai constaté que de nombreuses instances de LOC (11 sur 15) appartenaient au groupe DIR. Cela est dû à (i) la faible fréquence des LOC dans les données et (ii) la proximité sémantique entre DIR et LOC—ils se rapportent à un lieu et sont souvent difficiles à distinguer les uns des autres. Ce résultat indique que l'identification des fonctions est encore limitée par les complications mentionnées ci-dessus.

7 Conclusion

7.1 Résumé des principaux résultats

Cette étude s'est déroulée en trois étapes : tout d'abord, j'ai identifié les fonctions spécifiques de chaque postposition en me basant sur le système de classification développé par le projet Sejong et sur les études déjà effectuées sur les postpositions adverbiales Coréennes. La postposition *-ey* a huit fonctions majeures, avec 'localisation' et 'but' qui occupent la majorité des occurrences. *-eyse* a deux fonctions, 'source' et 'localisation', et est utilisé beaucoup plus fréquemment que les autres. *-(u)lo* a six fonctions, dont les trois principales, 'état final', 'instrument' et 'direction', occupent plus de 80% de l'utilisation totale.

Ensuite, j'ai créé les modèles de classification/visualisation, l'un en utilisant une combinaison de PPMI et SVD comme modèle basé sur le nombre et l'autre en utilisant SGNS comme modèle basé sur la prédiction avec la base de l'estimation basée sur la similarité. En général, j'ai constaté que, si une postposition avait moins de fonctions, le modèle de classification obtenait une précision de classification élevée. Le modèle PPMI-SVD a atteint une grande précision de classification lorsque la taille du Window était grande, ce qui indique que pour la meilleure performance de classification, il a utilisé les caractéristiques sémantiques des grandes tailles de Window plus que les caractéristiques syntaxiques. En revanche, le modèle SGNS a montré une faible précision de classification, quelle que soit la taille des Windows. Par ailleurs, j'ai constaté que le modèle PPMI-SVD était plus affecté par la taille du corpus que le modèle SGNS. Cela s'explique par le fait que le modèle PPMI-SVD est sensible à la fréquence des tokens des mots, alors que le modèle SGNS est sensible à la fréquence des types de mots. À travers la visualisation, j'ai trouvé que (i) les clusters n'ont pas changé considérablement par les environnements des vecteurs du niveau du mot, et (ii) il y avait les deux types de mots co-occurrents : les mots qui apparaissaient fréquemment dans le corpus total et les mots qui apparaissaient seulement quand la postposition était utilisée comme une fonction spécifique.

Finalement, j'ai appliqué BERT pour *transformer* tous les mots en différents vecteurs, tout en considérant leurs informations contextuelles pour la même tâche de classification. Pour la tâche de classification, le modèle BERT a obtenu une grande précision de classification : 0.815 pour *-ey*, 0.898 pour *-eyse*, 0.813 pour *-(u)lo*. Ce résultat était supérieur aux performances des modèles des études précédentes et des modèles de vecteur du niveau du mot que j'ai utilisés. En outre, j'ai constaté que le modèle BERT n'était pas particulièrement influencé par la taille du corpus de chaque fonction, contrairement au résultat montré par les modèles de vecteur au niveau du mot. Les raisons en sont que le modèle BERT a attribué à chaque mot un vecteur basé sur les informations contextuelles et a fonctionné sur la base du modèle pré-entraîné avec une grande quantité de données de corpus. A travers la visualisation, j'ai constaté que le modèle BERT pouvait reconnaître les fonctions de chaque postposition au fur et à mesure que l'epoch (i.e., l'apprentissage) progressait, même si les fonctions occupaient une plus petite partie de la taille totale du corpus. Ceci était également contradictoire avec les résultats des modèles traditionnels de vecteur au niveau du mot, qui sont connus pour être considérablement affectés par la taille du corpus. Cela indique que le modèle BERT peut identifier des fonctions relativement peu fréquentes à un niveau satisfaisant grâce à un nombre suffisant d'epochs. D'ailleurs, cela suggère qu'il est capable de simuler la façon dont les humains interprètent la polysémie impliquant les postpositions adverbiales Coréennes de façon plus appropriée que les modèles de vecteur au niveau du mot.

7.2 Limites et travaux futurs

Malgré ces résultats, cette thèse reste limitée. Je reconnais certaines limites de ce projet comme suit.

Avant tout, je me suis concentré uniquement sur trois différentes postpositions adverbiales Coréennes qui ont une polysémie du niveau des mots. Cependant, selon la description statistique du dictionnaire *Standard-Korean* (1999), il existe 361 postpositions en langue Coréenne. Dans cette optique, les résultats obtenus à partir des trois postpositions étudiées dans cette thèse ne sont pas suffisants pour généraliser toutes les postpositions de la langue Coréenne. Par conséquent, à l'avenir, j'améliorerais cette étude pour couvrir davantage de postpositions qui ont des degrés de polysémie similaires à ceux de *-ey*, *-eyse*, et *-(u)lo*.

Deuxièmement, j'ai utilisé trois modèles de vecteur (PPMI-SVD, SGNS et BERT) pour la tâche de classification dans cette thèse. Cependant, compte tenu du fait que d'autres modèles de vecteur du mot contextualisés ont été publiés après BERT, tels que le *Generation Pre-trained Transformer 3* (GPT-3 ; Brown et al., 2020) ou le *Robustly Optimized BERT Pretraining Approach* (RoBERTa ; Liu et al., 2019b), il est nécessaire de les utiliser afin d'assurer la généralisabilité méthodologique et d'attester des méthodes de calcul récentes en Coréen, une langue typologiquement différente des principales langues indo-européennes.

7.3 Implications des résultats

Malgré ces limites, cette thèse a deux implications majeures.

Premièrement, il fournit les moyens possibles et les limites de l'application de trois modèles de vecteur différents pour la tâche d'identification de la fonction prévue des postpositions adverbiales Coréennes. De nombreuses études ont été réalisées sur l'interprétation de la polysémie au niveau du mot dans les principales langues indo-européennes en utilisant des modèles de vecteur au niveau du mot ou des modèles de vecteur du niveau des phrases dans le cadre de la modélisation sémantique distributive. Bien que de nombreuses recherches aient été menées sur l'anglais à ce sujet, très peu d'études ont porté sur l'interprétation de la polysémie dans une langue typologiquement différente de l'anglais. J'ai donc porté mon attention sur le Coréen, une langue peu explorée à cet égard, en me concentrant sur la relation entre les trois différents modèles de vecteur et la polysémie au niveau du mot des postpositions

adverbiales. Considérant que les recherches antérieures sont orientées vers les principales langues indo-européennes telles que l'anglais, la tentative de cette thèse contribue à la généralisation méthodologique en appliquant le calcul à une langue moins étudiée comme le Coréen.

En deuxième lieu, cette thèse propose deux systèmes de visualisation interactifs qui aident à identifier les relations entre les mots ou les phrases et à montrer les changements des groupes en fonction des environnements (i.e., les modèles, les postpositions, la taille des Windows et les epochs). Bien que les modèles de vecteur au niveau du mot et des phrases aient été fréquemment utilisés dans les études récentes, il est très difficile de comprendre comment ces modèles de vecteur interprètent la polysémie au niveau du mot. Le premier système de visualisation visait à explorer les résultats du vecteur au niveau du mot. Cela nous permet de voir les groupes de postpositions et leurs mots co-occurrents afin de comprendre comment les relations entre les mots ont changé en fonction des fonctions de chaque postposition. Le deuxième système de visualisation a été développé pour montrer comment le modèle de vecteur au niveau de la phrase (i.e., BERT) reconnaît la polysémie impliquant les postpositions. Considérant que le système de visualisation pourrait aider à comprendre les résultats de calcul plus facilement et plus clairement grâce à un affichage intuitif (et aussi informatif) des données linguistiques, l'essai de cette thèse a une contribution particulière pour les études futures.

Bibliographie

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

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

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.

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.

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.

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

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.

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

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.

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.

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.

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.

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

Dylan Glynn and Justyna Robinson. *Corpus Methods for Semantics*. Corpus Methods for Semantics. Quantitative studies in polysemy and synonymy. John Benjamins, January 2014. doi : 10.1075/hcp.43. URL https://halshs.archives-ouvertes.fr/halshs-01284061.

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.

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

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.

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.

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.

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.

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

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.

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.

Gyeongheum Maeng. Cognitive semantics of korean postposition '*-ey*'. *The Journal of Korean Studies*, 41 :325–366, 2016.

Chris McCormick. Bert fine-tuning tutorial with pytorch, 2019. URL http://mccormickml.com/2019/07/22/BERT-fine-tuning/.

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*, 2013. URL http://arxiv.org/abs/1301.3781.

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.

Ki-sim Nam. The use of the korean postposition : focus on '*-ey*' and '*-(u)lo*'. *sekwang hakswul calyosa*, 1993.

Jeong-woon Park. A polysemy network of the korean instrumental case. *Korean Journal of Linguistics*, 24(3) :405–425, 1999.

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.

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.

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.

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.

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.

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.

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.

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.

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.