Semantic Specialization of Word Representation for Individual Word Senses

Master Thesis

presented by
Lifei Lu
Matriculation Number 1572374

submitted to the
Data and Web Science Group
Prof. Dr. Goran Glavaš
University of Mannheim

September 2019

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The distributed representation of words in the vector space, usually referred to as word embeddings, is a core research domain in Natural Language Processing (NLP) [23], since it contributes to a wide range of NLP tasks as the entry point. One of the earliest application of distributed word representation dates back to 1986 [38]. This technique has been applied to statistical language modeling with considerable success subsequently [2]. By efficiently handling considerable amount of texts, the distributed word representation benefits a great variety of NLP tasks, such as syntactic parsing [42], and machine translation [41].

The word embeddings intents to quantify and classify semantic and syntactic information between linguistic elements among a large amount of instances from language corpora. Almost all models for training word representations are under the distributional hypothesis that words appearing in similar contexts tend to have similar meanings [18]. For instance, the ***Skip-Gram*** model that introduced by Mikolov et al. [23], an efficient approach for learning high quality distributed word representations, relies on word co-occurrences gained from large amounts of unstructured text data.

Due to the distributional hypothesis, the models for creating the distributed representations of words tend to conflate all possible meanings of one word into a single representation. This is one of the main limitations of word embeddings : the meaning conflation deficiency. Hereafter, generating individual sense representations for all meaning of words are presented as a solution, usually referred to as sense embeddings [5].

1.2 Problem Statement

As mentioned, most distributed representation techniques usually model a word as a single denotation. That means all meanings of one word has been merged together, which cannot sketch the reality explicitly [33]. For example, the word **bank**, in most cases it indicates *a place where money is lent or exchanged, or put for safety and/or to acquire interest*. However, in fact, it could be a verb not only a noun, and it has several meanings respectively. The **Figure 1.1**, **1.2** shows all possible meanings of the word **bank** in the ***WordNet***, a large lexical database of English [43], and it can be seen that there are ten meanings of the word **bank** as a noun, and eight meanings of it as a verb.

Noun

- S: (n) **bank** (sloping land (especially the slope beside a body of water)) *"they pulled the canoe up on the bank"; "he sat on the bank of the river and watched the currents"*
- S: (n) depository financial institution, **bank**, banking concern, banking company (a financial institution that accepts deposits and channels the money into lending activities) *"he cashed a check at the bank"; "that bank holds the mortgage on my home"*
- S: (n) **bank** (a long ridge or pile) *"a huge bank of earth"*
- S: (n) **bank** (an arrangement of similar objects in a row or in tiers) *"he operated a bank of switches"*
- S: (n) **bank** (a supply or stock held in reserve for future use (especially in emergencies))
- S: (n) **bank** (the funds held by a gambling house or the dealer in some gambling games) *"he tried to break the bank at Monte Carlo"*
- S: (n) **bank**, cant, camber (a slope in the turn of a road or track; the outside is higher than the inside in order to reduce the effects of centrifugal force)
- S: (n) savings bank, coin bank, money box, **bank** (a container (usually with a slot in the top) for keeping money at home) *"the coin bank was empty"*
- S: (n) **bank**, bank building (a building in which the business of banking transacted) *"the bank is on the corner of Nassau and Witherspoon"*
- S: (n) **bank** (a flight maneuver; aircraft tips laterally about its longitudinal axis (especially in turning)) *"the plane went into a steep bank"*

Figure 1.1: All Possible Meanings of the word **bank** as a Noun in the ***WordNet***

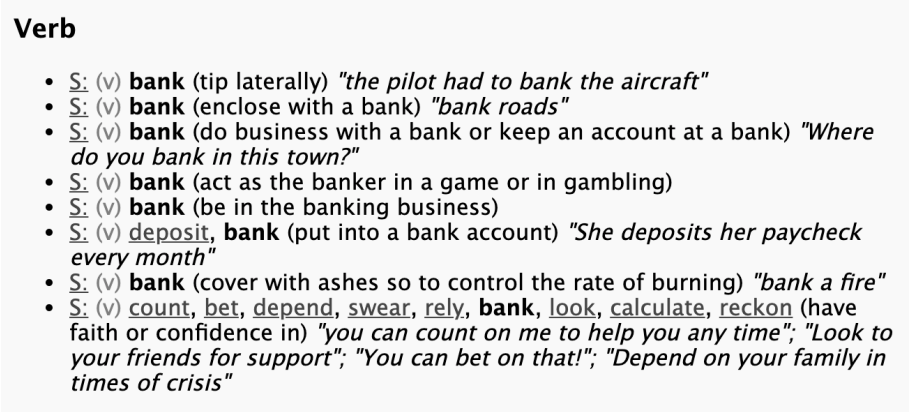

Figure 1.2: All Possible Meanings of the word **bank** as a Verb in the ***WordNet***

Using one semantic representation of **bank** means merging all its interpretations into one vector, which ignores the polysemy of the word. This would lead to low performance of other NLP tasks which use the word embeddings technique as foundation. The approaches which can deal with the meaning conflation of the word embeddings will not only improve the quality of the semantic representations of words, but also bring benefits to other NLP tasks.

Semantic specialization of the word representations uses the external knowledge from lexical resources to strengthen the semantic content of the distributional word representations [45], and this is the scope where the meaning conflation of the word representation could be addressed. The existing utilization of the external lexical knowledge on semantic specialization falls into two main categories, both of them aim to creating the sense representations of the words [27] [10]:

- ***Joint Specialization Model***

  The Joint Specialization Models merge the external linguistic knowledge as constraints when training the distributed representations of the words [30], and the original objective is modified. However, generally these models perform worse compared to the Post-Processing Models.

- ***Post-Processing Model***

  The Post-Processing Models are also termed as retrofitting models. They shape the pre-trained word representations to satisfy the external lexical constraints [7]. And the sense level representations created via these models usually achieve good performance on a large range of downstream tasks.

This paper focus on resisting the meaning conflation deficiency of the word embeddings. And as the tactics of the Post Processing Models is more powerful than it of the Joint Specialization Models generally, retrofitting model is preferred. Inspired by the models illustrated in the paper [10] [45], we would like to specialize the semantic representations at the sense level, fine-tuning the pre-trained word vectors into multiple sense-independent ones. For instance, using the well trained word representation of **bank** to derive ten sense representations for the noun meanings and eight sense representations for the verb meanings of it by satisfying the external lexical constraints, which means the word **bank** could not just have one word representation, but eighteen sense representations instead. The following **Figure 1.3** shows the basic idea of our paper.

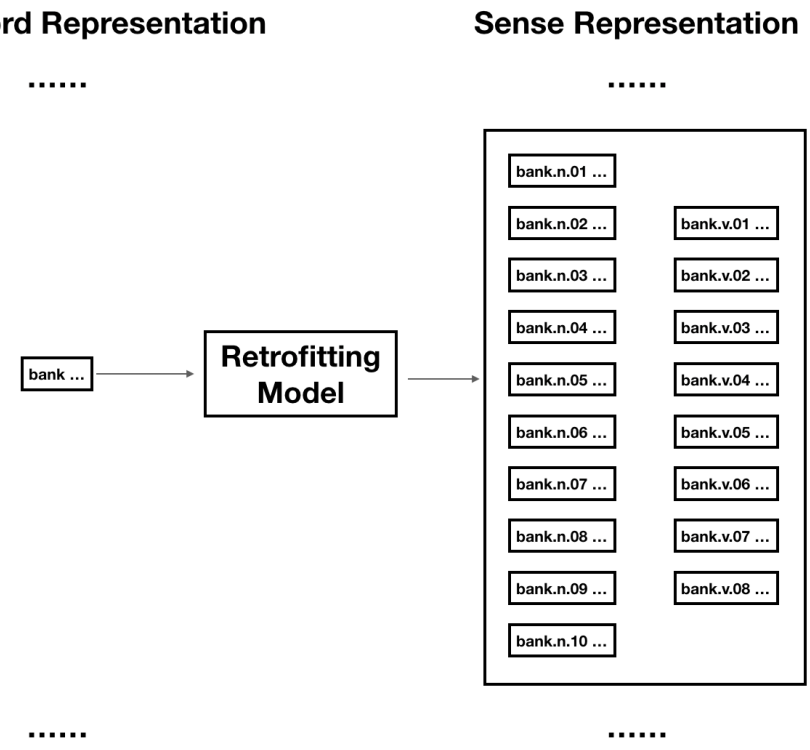

Figure 1.3: Sense Representation Objective

1.3 Contribution

In this paper, we focus on doing the research on the sense representations of the words. While there are many advanced sense specialization models which already gain super improvements compared to the previous works. We try to reuse the existing superior works to achieve higher performance on representing senses of the words, and this would be our highlight.

As the famous saying goes, we stand on the shoulders of giants. Reorganizing different works which belong to different NLP areas, is also an innovation. We design a procedure to combine several elapsed techniques to reach our objective: semantic specialization of word representation for individual word senses. And the combination of different existing techniques is re-implemented in our experiment.

A huge amount of thoughts and efforts are employed to integrate different techniques to make them adapt to each other. In other word, we pay much attention to decorate the input and the output of each step to lead the procedure to run unimpededly. Moreover, not only simply measuring our experiment on several standard data set, we also manage horizontal and vertical contradistinction over the experiment results. And this is delineated in the **Chapter 4 and 5** detailedly.

The most important thing is that, the evaluation results on the standard data set ***SimLex-999*** give prominence to the fact that the specialization approach highly beneficial the sense level distributed representations. It outperforms two recent state-of-the-art sense representation techniques, ***DECONF*** model [33] and the ***Pilehvar and Navigli (2015)*** model [34], and this is the main achievement we gain.

We believe that the improvement we achieve would benefit a range of NLP tasks, and this is discussed in the **Section 6.1**.

Chapter 2

Related Work

2.1 Word Representation

The learning approaches for creating word representations, which can handle a huge amount of linguistic data to embed the semantic meanings of the words into vectors, have attracted great quantity attention since the beginning of the Natural Language Processing [22]. The main theory behind word representations learning is based on Vector Space Model [5]. And here we would like to introduce several word representation learning approaches.

2.1.1 Vector Space Model

Considering a document as a vector whose dimension is the whole vocabulary might be the earliest Vector Space Model applied in Natural Language Processing [39]. The occurrences of individual words are the elements of the vector, and the respective dimensions are weighted by the word frequencies or normalized word frequencies within the document [40]. And this document-based Vector Space Model has been successfully applied to multiple NLP tasks, such as information retrieval [19].

Later, the word-based Vector Space Model has been constructed according to the normalized frequencies of the co-occurring words in a corpus [20]. It is under the hypothesis that words appearing in similar contexts are near to each other in the vector space [18].

Although the word-based Vector Space Models also have been successfully applied to a variety of NLP tasks, such as spelling correction [15], information extraction [17] and word sense disambiguation [35], the high dimensionality of the produced vectors limits its development. In order to solve this problem, Latent

Semantic Analysis [14] is leveraged to reduce the high dimensions. More importantly, neural network is applied to train low-dimensional vectors directly.

2.1.2 Word Embeddings

The word representation technique has achieved a qualitative revolution when employing the neural network technique. The word2vec model is a typical word embeddings model utilizing the neural network, and two word2vec related model are proposed: Continuous ***Bag-of-Words*** model and ***Sikp-Gram*** model [23].

The ***Bag-of-Words*** model uses the feed-forward neural network language model [2] to predict the word using its linguistic context by minimizing a certain loss function. The **Figure 2.1** shows the basic architecture of the ***Bag-of-Words*** model.

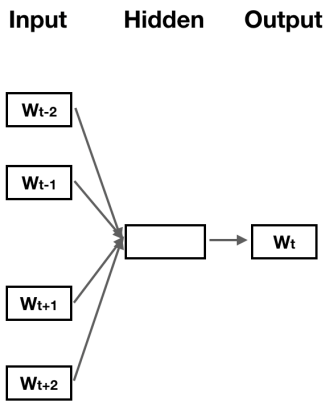

Figure 2.1: The Brief Architecture of the ***Bag-of-Words*** Model

The W_t is the target word and the $[W_{t-n}, ..., W_{t-1}, W_{t+1}, ...W_{t+n}]$ refer to the surrounding contexts. The ***Bag-of-Words*** model contains three layers, and it uses the combination of vector representations of surrounding words of a target word as the input and the vector of the target word which has the same dimensions as the input vectors is the output of the model.

Instead of learning the vector of the target word, the ***Sikp-Gram*** model predicts the words in its surrounding context using neural networks. The **Figure 2.2** shows the basic architecture of the ***Sikp-Gram*** model.

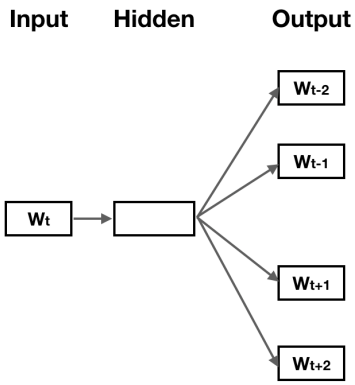

Figure 2.2: The Brief Architecture of the ***Sikp-Gram*** Model

The training objective of ***Sikp-Gram*** model is different from the ***Bag-of-Words*** model, and the input and the output are swapped.

Apart from the classic neural network word embedding approaches above, there are some other models which could also provide high quality word representations. The ***GolVe*** model injects global matrix factorization and local context window methods by training only on the nonzero elements in a word-to-word co-occurrence matrix, to produces a vector space with meaningful substructure [32]. The ***Dict2vec*** uses the natural language dictionaries, the data source for describing words, to generate new word pairs to move the similar words closer [44]. All of these models make progresses over the previous works.

2.2 Meaning Conflation Deficiency

Representing a word as a single point in the semantic space ignores the fact that a word can have more than one sense, which means the word embeddings models conflate all the meanings of a word into a single representation. Even when all meanings of a word occur in the training corpus, the word embeddings model is unable to capture different meanings. And this limitation could bring certain disadvantages negatively influencing on a variety of NLP tasks.

The meaning conflation deficiency could cause that two semantically unrelated words which are similar to two different meanings of another one word, appear close to each other in the semantic space [29]. For instance, the word **fly** can be the insect with a pair of functional wings for flight when it is considered as a noun, and it usually represents the same meaning as the word **housefly** in most contexts.

However, when the word **fly** is a verb, it can be understand as traveling over the sea in an aircraft, which is similar as the word **sail**. The traditional word embeddings technique could pull the word **housefly** and the word **sail** near to each other in the vector space, which does not make sense.

2.3 Word Sense Disambiguation

The Word Sense Disambiguation is related to the meaning conflation deficiency, but considered as an AI-complete task. It aims to associate a word with its most appropriate meaning [35] [28] depending on the knowledge resources.

The Word Sense Disambiguation also meets the challenge of the individual sense representations. The task named as word sense induction or discrimination directs this conundrum by inducing senses automatically from a corpus [1]). And it roughly divided into two types : supervised method and knowledge-based method. The supervised method utilize the sense-annotated corpus directly [21], and the knowledge-based methods learn from the structure and content of the knowledge resources [6].

Similar to the Word Sense Disambiguation, the meaning conflation deficiency of word embeddings is also possible to be improved.

2.4 Sense Representation

In order to ameliorate the meaning conflation deficiency, the sense representations for individual word meanings is developed upon the word representation models. In most instances, external linguistic knowledge is employed by the word representation technique to induce the sense representations. Depending on where to inject the external linguistic knowledge, the sense representation models can be roughly divided into two main paradigms : the Joint Specialization Model and the Post-Processing Model.

2.4.1 Joint Specialization Model

Integrating the external linguistic knowledge directly into the traditional word representation models [30] or modifying the regularization of the original objective [47], is the main idea behind the joint sense specialization models.

However, generally these models per-form not well as the Post-Processing Models.

2.4.2 Post-Processing Model

The post-processing models use the pre-trained distributional word representations to induce the sense representations of individual word meanings by restricting the word representation to satisfy the lexical constraints extracted from the external linguistic resources [7]. A large number of post-processing models use the similarity constraints, but as a matter of fact, not only the similar words constraints could lead to better performance of the sense representations, also the dissimilar word constraints could [27].

The Explicit Retrofitting model [10] follows the post-processing model strategy and the **Figure 2.3** shows the basic architecture of the ex model.

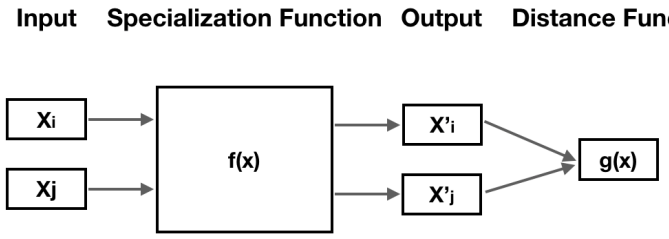

Figure 2.3: The Brief Architecture of the Explicit Retrofitting Model

Inspired by the theories behind the joint specialization model and the post-processing model, we propose an approach to generate representations for individual word senses using the pre-trained word representations by injecting the external knowledge.

Chapter 3

Methodology

The objective of this paper is to specialize the word representations to create sense level representations by injecting the external linguistic knowledge constraints. Since Post-Processing Specialization Models show strong competitiveness over the Joint Specialization Models, we design a post-processing specialization procedure to try to ameliorate the meaning conflation deficiency of the word embeddings technique.

In this chapter, the overview of the post-processing specialization procedure we desing is introduced firstly, and the particulars are delivered detailedly next.

3.1 Overview of the Procedure

According to the basic ideology of the Post-Processing Specialization Model, the explicit specialization of the word representations for individual word senses mainly consists of the following three contents: the pre-trained distributed word representations, the external lexical knowledge and the retrofitting model. The retrofitting model use the pre-trained distributed word representations to create the specialized representations at the sense level by merging the external lexical knowledge. The following **Figure 3.1** shows the overview of the specialization procedure we design, and the whole procedure is divided into three steps.

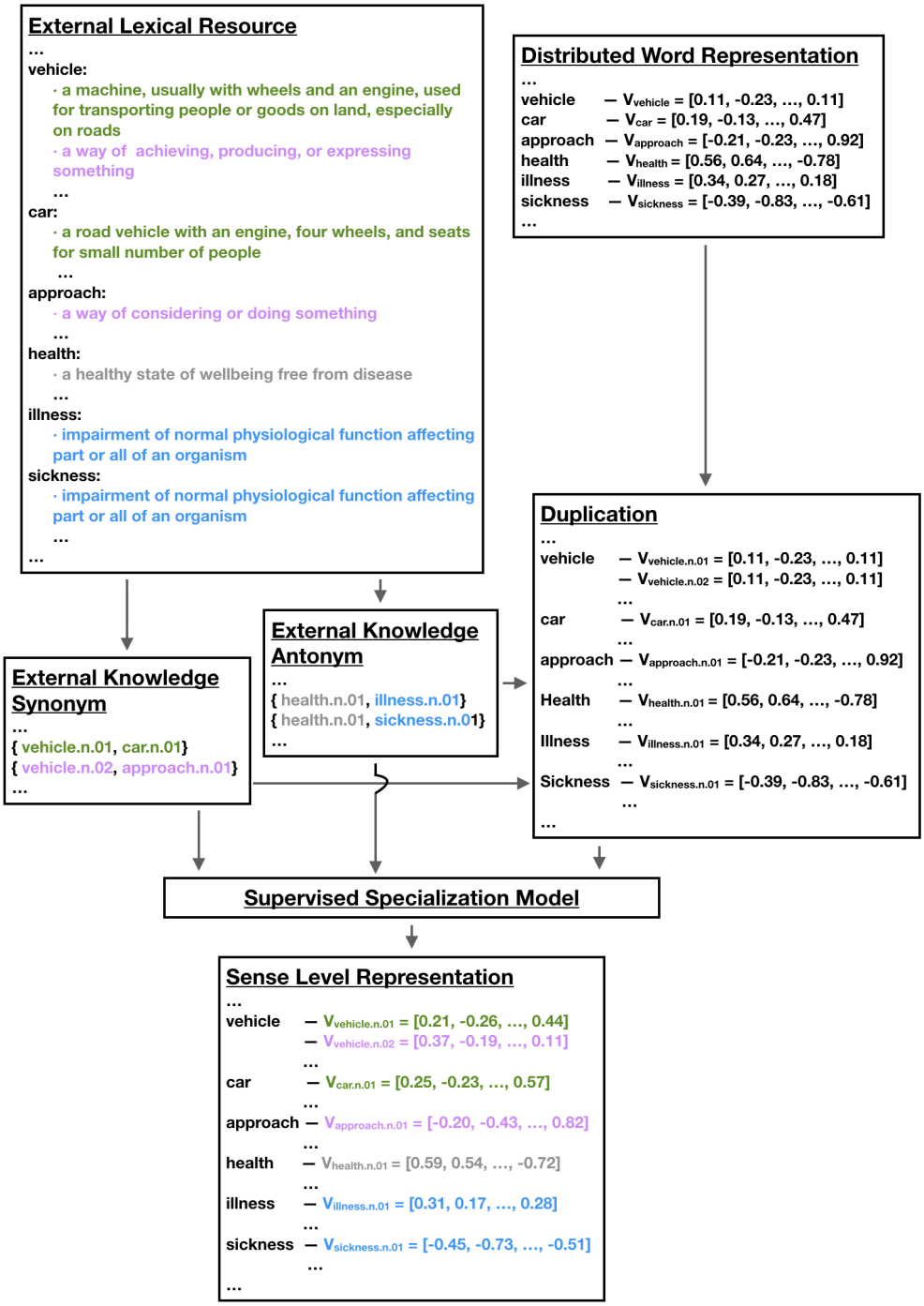

Figure 3.1: Sense Representation Specialization Procedure

3.2 Distributed Word Representation

In general, the distributed representation of each word use dimensional vector to present the word in a digital way, and normally they have 300 dimensions. The wildly used distributional word vector collections only contain one vector for each word. In this paper, the pre-trained word representations are used to prepare the initial sense level vectors for each word after gaining the linguistic constraints. But actually, it is still at the word level because all the sense level vectors of each word are the same. The details could be found in the **Figure 3.2**, which is the right part of the **Figure 3.1**.

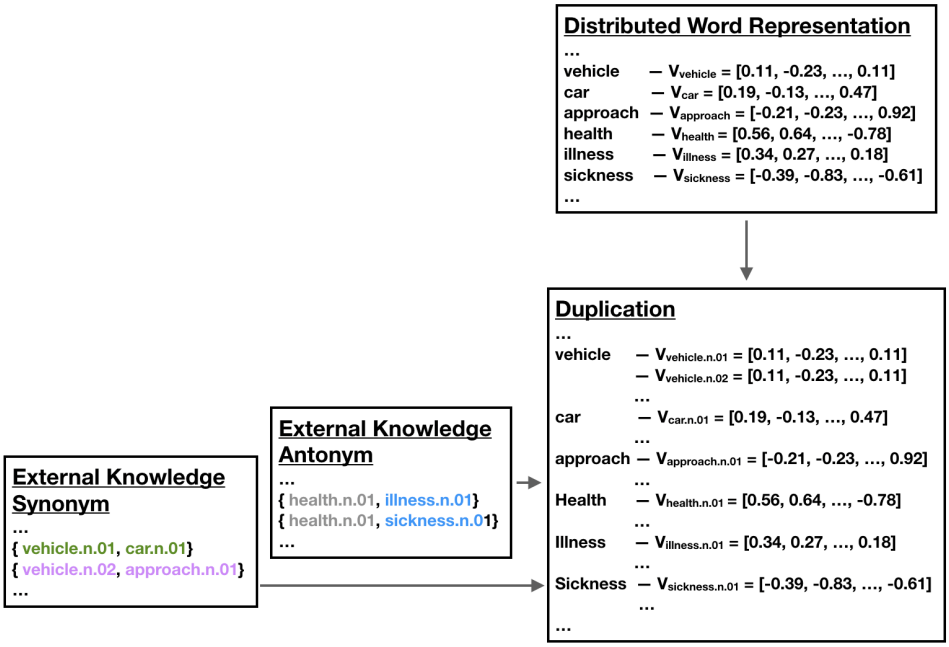

Figure 3.2: Initial Sense Level Vector

Duplicating the pre-trained word vector according to the number of the sense annotations of the words is used to create the initial sense level vectors. V [1] is used to represent the word representation vector. For example, if there are two meanings

[1] The example vectors above are not real vectors from any distributed word representation collection.

of the word **vehicle**, the two same sense level vectors $V_{vehicle\_n\_01}$ and $V_{vehicle\_n\_02}$ are generated at the beginning.

3.3 Linguistic Constraint

The synonym and antonym word pairs extracted from the external lexical resource are used to be the linguistic constraints in our scenario. The details could be found in the **Figure 3.3**, which is the left part of the **Figure 3.1**.

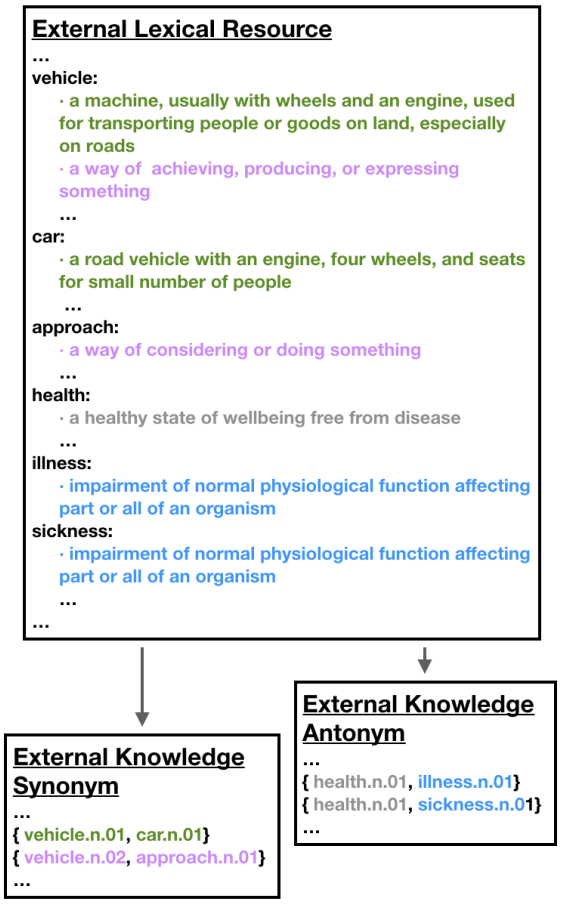

Figure 3.3: Linguistic Constraint Extracted from External Resource

It can be seen that one sense of the word **vehicle** has the same meaning as one

sense of the word **car**, and another sense of the word **vehicle** has the same meaning as one sense of the word **approach**. Thus two synonym pairs could be constructed. And antonym pairs could also be extracted from the lexical resource, such as the {**health.n.01, illness.n.01**}.

3.4 Specialization Model

The specialization model and the retrofitting model refer to the same technique which is used to merge the lexical knowledge into the word level representations to generate the sense level ones. The details could be found in the **Figure 3.4**, which is the bottom part of the **Figure 3.1**.

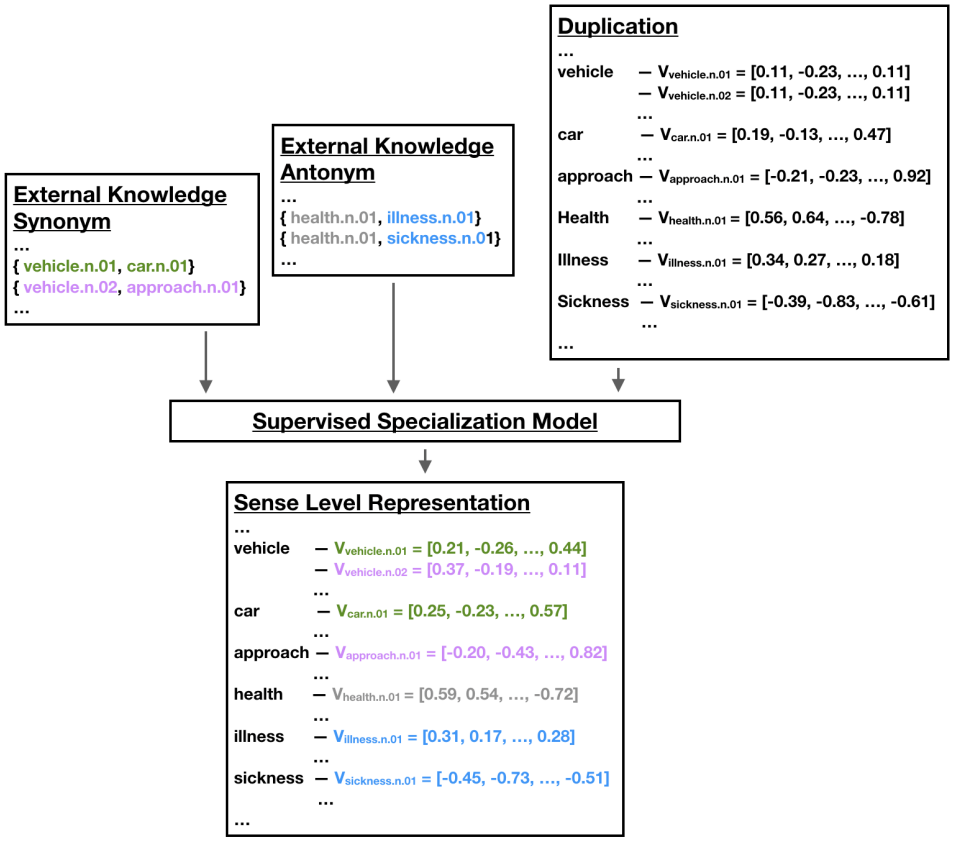

Figure 3.4: Specialization Model

As we can see, the linguistic knowledge constraints and the dummy sense level representations are the input of the specialization model, and after modification, the real sense level representations are the output. One word could has different sense level vectors, such as the different two vectors of the word **vehicle** from the **Figure 3.4**[2].

[2]The example vectors above are not real vectors from any distributed word representation collection.

Chapter 4

Experiment

4.1 Setting

In order to verify the workflow designed above, firstly, we implement it in the English scenario. Therefore, the collection of distributional word vectors and the linguistic constraints discussed below are all in English. As for the retrofitting model, we chose one that could make use of cross-lingual lexicons to build high-quality vector spaces for a number of different languages. Thus, we believe that our workflow can also work on other languages, but we do not test it in this paper but it could be our future work.

4.1.1 Distributional Word Vector Collection

We select two different publicly available and widely used collections of pre-trained distributional word vectors in our experiment, in order to avoid uniformity. They are used to create the dummy sense level representations as the input of the retrofitting model.

- ***Wiki Word Vector – fastText***

  The Wiki Word Vector collection is trained on Wikipedia using ***fastText*** model presented in [3]. These vectors which have dimension 300 are obtained using the Skip-Gram model.

  The word vectors contain the binary and text default formats from ***fastText***. The default format is composed of line which has a word followed by its vector and each value is space separated. In addition, words are sorted in descending order by frequency. The collection can be found **here**[1].

[1] https://fasttext.cc/docs/en/pretrained-vectors.html

In this distributional word vectors collection, there is only one word embedding vector for each word, ignoring the capitalization, which is different from the ***word2vec*** word embeddings collection. And **Figure 4.1** is an example of word **maker** to show the difference.

['maker', '-0.081627 0.04328 -0.10699 0.027693 -0.15899 0.35172 0.053552 ......']

Figure 4.1: Word Vector of **maker** in ***fastText***

- ***Google News Vector – word2vec***

  The Google News Vector collection is trained on part of Google News data set using the ***word2vec*** tool which contains the continuous ***Bag-of-Words*** and ***Skip-Gram*** architectures for computing vector representations of words. This collection contains 300-dimensional vectors for 3 million words and phrases [22] [23] [24].

  The collection can be found **here**[2].

  In this word embedding vector collection, the capitalization does not be omitted. And **Figure 4.2** is an example of word **maker** to show the difference from the ***fastText*** collection.

['maker', '0.135742 -0.149414 0.15332 0.251953 0.098145 0.239258 0.40625 ......']

['Maker', '-0.080078 -0.183594 -0.048584 -0.03833 0.1875 0.339844 0.15332 ......']

Figure 4.2: Word Vector of **maker** in ***word2vec***

4.1.2 Linguistic Constraint

Injecting lexical knowledge from external resources into the pre-trained word vectors to create sense-level vectors is the main goal of the experiment. Firstly, abstracting the lexical knowledge constraints is the precondition for the other steps. Following the previous work [48] [31], we use the synonym and antonym pairs to be the constraints to train the ***ATTRACT-REPEL*** model, which is illustrated in the next section. Here we choose the ***WordNet*** [3] as our external resource.

[2] https://code.google.com/archive/p/word2vec/

[3] https://wordnet.princeton.edu/

WordNet [12] [43] is a large English lexical database, and contains words with its different senses. Specially, it clustered nouns, verbs, adjectives and adverbs into sets of cognitive synonyms, which is called synsets. Synsets are related through conceptual-semantic and lexical relationships, each representing a unique concept. In addition, the synset can be understood as the index of the word senses. For each word, normally, different senses of the word belong to different synsets. We continue using the word **maker** as the instance, **Figure 4.3**. The {**maker.n.01**} can be understood as the index of the synset, and the string versions of synonyms can be accessed via a Synset's lemma property, **Figure 4.4**.

```
>> wordnet.synsets('maker')

>> Synset('maker.n.01')
   Synset('godhead.n.01')
   Synset('manufacturer.n.01')
```

Figure 4.3: Synsets of **maker** in ***WordNet***

pos, between the word and the number, is one of the module attributes, ADJ, ADJ_SAT, ADV, NOUN or VERB. The **Table 4.1** shows the abbreviations of the part of the speech.

number is the sense number, counting from 0.

Part of Speech	
n	NOUN
v	VERB
a	ADJECTIVE
s	ADJECTIVE SATELLITE
r	ADVERB

Table 4.1: Part of Speech

Here, it can be seen that the word **maker** has three different meanings and they belong to different synsets. However, each synset can contain more than one word. For instance, the synset **maker.n.01** has two word inside, not only one word **maker**, **Figure 4.4**.

```
>> wordnet.synset('maker.n.01').lemmas( )

>> Lemma('maker.n.01.maker')
   Lemma('maker.n.01.shaper')
```

Figure 4.4: Elements of Synset **maker.n.01** in ***WordNet***

lemma is the words morphological stem.

In our experiment, we only composed constraint pairs from NOUN, VERB and ADJECTIVE.

4.1.3 Retrofitting Model

As mentioned in the Introduction **Chapter 1**, there are two main approaches how word embeddings vectors assimilate the external lexical knowledge. In our experiment, we decided to use a post-processing model, ***ATTRACT-REPEL***[4] model which shows boost performance in some downstream tasks.

The ***ATTRACT-REPEL*** algorithm [27] merges in the knowledge from lexical resources to produce semantically specialized word vector spaces. In other word, it injects the similarity and antonym constraints into the pre-trained distributional word vectors to capture the semantic similarity, precisely satisfying our objective.

- It needs the synonym and antonym pairs as input. The negative example pair is chosen from the remaining in-batch vectors for each synonym pair, and each word in the negative pair is the one closest to the word in the synonym pair correspondingly. And as for each antonym pair, the negative example pair is also chosen from the remaining in-batch vectors, but each word in the negative pair is the one furthest away from the word in the antonym pair correspondingly.
- ***ATTRACT-REPEL*** algorithm pulls the synonym pairs to be closer to each other than their negative example pairs and forces the antonym pairs to be further away from each other than their negative example pairs, using two different cost functions below.

$$S(B_S, T_S) = \sum_{i=1}^{k_1} \quad [\tau(\delta_{syn} + x_l^i t_l^i - x_l^i x_r^i) + \tau(\delta_{syn} + x_r^i t_l^i - x_l^i x_r^i)]$$

[4] https://github.com/nmrksic/attract-repel/tree/master/code

$$A(B_A, T_A) = \sum_{i=1}^{k_2} \quad [\tau(\delta_{ant} + x_l^i x_r^i - x_l^i t_l^i) + \tau(\delta_{ant} + x_l^i x_r^i - x_r^i t_r^i)]$$

S and A are the set of synonym and antonym pairs respectively, and we use (x_l, x_r) to represent the synonym and antonym pair. B_S and B_A are the mini-batches, and (t_l, t_r) is the negative example pair. $\tau(x) = max(0, x)$ is the hinge loss function and δ is the similarity margin. k_1 and k_2 are the batch sizes.

- One additional regularization term is involved to keep the high-quality semantic content in the original distributional word vector space.

$$R(B_S, B_A) = \sum_{x_i \in V(B_S \cup B_A)} \lambda_{reg} \parallel \hat{x_i} - x_i \parallel_2$$

$V(B)$ is the set of all word vectors in the mini-batch. The λ_{reg} is the L2 regularization constant and $\hat{x_i}$ indicates the original distributional word vector of x_i.

- The sum of these three functions compose the final cost function.

$$C(B_S, T_S, B_A, T_A) = S(B_S, T_S) + A(B_A, T_A) + R(B_S, B_A)$$

Compared to the previous work, the ***ATTRACT-REPEL*** algorithm updates both the synonym and antonym pair and the negative example pair. Moreover, it combines the L2 regularization to preserve the content from the original pre-trained distributional vector space. These two features make the ***ATTRACT-REPEL*** algorithm perform good.

4.1.4 Environment

The whole experiment is implemented using Python. And because the ***ATTRACT-REPEL*** algorithm is built with Python 2.7, we follow their effort and switch from Python 3 to Python 2.7 when modeling.

Step	Python Version
Constraints Extraction	Python 3
Distributional Word Vector Preparation	Python 3
Model Setup	Python 2.7

Table 4.2: Project Environment

4.2 Implementation

As the main experiment settings above are mature, in this section, we will walk on the shoulder of the giants and will not pay much effort on explaining how they comply. But the implementation of the workflow will be sketched in a detailed way.

Inputting the whole collection of the distributional word vector does not make sense, since only the word vectors which related to constraints will be modified while modeling. Moreover, the bigness of the set of the constraints is much smaller than the bigness of the collection of the distributional word vector. Therefore, we extracted constraints from the external lexical resource firstly, and then fetched the words from the collection of the embeddings vectors. So only a small set of the distributional word vectors will be given to the***ATTRACT-REPEL*** model. Thereby the run time of the model will be reduced to the greatest extent.

Hence, first of all, we extracted the constraints, and next used it to seek the words from the collection of the embeddings vectors. Then we threw both of them into the ***ATTRACT-REPEL*** model to get the sense level word embeddings vectors.

4.2.1 Constraint Extraction

As mentioned in the Settings **Section 4.1**, the ***WordNet*** was chose as the external lexical resource. And it could be easily accessed by Python package **nltk**. Since the ***WordNet*** grouped the synonyms into sets of synsets, the synonym pair was easy to construct. As for the antonym pairs, a bending way were used to constitute them.

Synonym Pairs Composition

The approaches of getting synonym pairs for Noun, Verb and Adjective are the same. Noun synsets are chosen as the instance to explain the process to compose the synonym pairs.

In addition, the potential constraint pairs of Adjective Statement and Adverb are ignored, since they occupy a honestly small percentage compared to constraint pairs of Noun, Verb and Adjective.

First of all, we fetched all noun synsets, and there are 82115 from the ***WordNet***. For each synset, we obtained all possible combination of every two words. The following pseudocode illustrates the algorithm.

Algorithm 1 Noun Synonym Pair Extraction

SYNONYMPAIREXTRACTION()

$noun\_synsets = wordnet.all\_synsets('n')$
$noun\_synpair =$ *empty list*
loop
  for i in $noun\_synsets$ **do**
    $synset = wordnet.synset(i).lemmas()$
    $synonym\_pair = itertools.combinations(synset, 2)$
    $noun\_synpair.append(synonym\_pair)$
  end for
end loop

wordnet.all_synsets(ń) is the function for getting all noun synsets.

wordnet.synset(i).lemmas() is for getting all elements in the synset i.

itertools.combinations(synset, 2) can construct all combinations of each two elements in a certain synset.

Here we use the synset **lower_jaw.n.01** as an instance to show the output of each step. **lower_jaw.n.01** can be seen as the index of the synset, and the part following is one word in this synset.

```
wordnet.synset(lower_jaw.n.01).lemmas( )

>> lower_jaw.n.01.lower_jaw
   lower_jaw.n.01.mandible
   lower_jaw.n.01.mandibula
   lower_jaw.n.01.mandibular_bone
   lower_jaw.n.01.submaxilla
   lower_jaw.n.01.lower_jawbone
   lower_jaw.n.01.jawbone
   lower_jaw.n.01.jowl
```

Figure 4.5: Elements of Synset **lower_jaw.n.01** in ***WordNet***

```
itertools.combinations(synset_lower_jaw, 2)

>> ('lower_jaw.n.01.lower_jaw', 'lower_jaw.n.01.mandible')
   ('lower_jaw.n.01.lower_jaw', 'lower_jaw.n.01.mandibula')
   ('lower_jaw.n.01.lower_jaw', 'lower_jaw.n.01.mandibular_bone')
   ('lower_jaw.n.01.lower_jaw', 'lower_jaw.n.01.submaxilla')
   ('lower_jaw.n.01.lower_jaw', 'lower_jaw.n.01.lower_jawbone')
   ('lower_jaw.n.01.lower_jaw', 'lower_jaw.n.01.jawbone')
   ('lower_jaw.n.01.lower_jaw', 'lower_jaw.n.01.jowl')
   ('lower_jaw.n.01.mandible', 'lower_jaw.n.01.mandibula')
   ('lower_jaw.n.01.mandible', 'lower_jaw.n.01.mandibular_bone')
   ('lower_jaw.n.01.mandible', 'lower_jaw.n.01.submaxilla')
   ('lower_jaw.n.01.mandible', 'lower_jaw.n.01.lower_jawbone')
   ('lower_jaw.n.01.mandible', 'lower_jaw.n.01.jawbone')
   ('lower_jaw.n.01.mandible', 'lower_jaw.n.01.jowl')
   ('lower_jaw.n.01.mandibula', 'lower_jaw.n.01.mandibular_bone')
   ('lower_jaw.n.01.mandibula', 'lower_jaw.n.01.submaxilla')
   ('lower_jaw.n.01.mandibula', 'lower_jaw.n.01.lower_jawbone')
   ('lower_jaw.n.01.mandibula', 'lower_jaw.n.01.jawbone')
   ('lower_jaw.n.01.mandibula', 'lower_jaw.n.01.jowl')
   ('lower_jaw.n.01.mandibular_bone', 'lower_jaw.n.01.submaxilla')
   ('lower_jaw.n.01.mandibular_bone', 'lower_jaw.n.01.lower_jawbone')
   ('lower_jaw.n.01.mandibular_bone', 'lower_jaw.n.01.jawbone')
   ('lower_jaw.n.01.mandibular_bone', 'lower_jaw.n.01.jowl')
   ('lower_jaw.n.01.submaxilla', 'lower_jaw.n.01.lower_jawbone')
   ('lower_jaw.n.01.submaxilla', 'lower_jaw.n.01.jawbone')
   ('lower_jaw.n.01.submaxilla', 'lower_jaw.n.01.jowl')
   ('lower_jaw.n.01.lower_jawbone', 'lower_jaw.n.01.jawbone')
   ('lower_jaw.n.01.lower_jawbone', 'lower_jaw.n.01.jowl')
   ('lower_jaw.n.01.jawbone', 'lower_jaw.n.01.jowl')
```

Figure 4.6: Combinations of Every Two Elements form Synset **lower_jaw.n.01**

As for the verb and adjective, they follow the same algorithm as noun. The **Table 4.3** shows the statistics information of the ***WordNet*** and how many synonym pairs we extracted altogether.

Part of Speech	Synset Count	Synonym Pair Count
NOUN	82,115	107,469
VERB	13,767	24,577
ADJECTIVE	18,156	22,730
SUM	114,038	154,776

Table 4.3: Synonym Constraint Pairs

Antonym Pairs Composition

The approach for extracting the antonym constraint pairs is not as simple as synonym, due to the synset structure of the ***WordNet***. Whereupon, it had been divided into two steps. And here, we continue using noun as the instance to explain the composition procedure.

- ***Fetching Antonym Synset Pairs***

  For each element s in a synset S, it could have an antonym synset A. First of all, we fetched antonym synset A for every s from S, and then composed $[S, A]$ as an antonym synsets pair. Hence, for every synset S, it could have several antonym synsets pairs $[S, A]$. The following pseudocode illustrates the algorithm.

Algorithm 2 Noun Antonym Synset Pair Extraction

ANTONYMSYNSETSPAIREXTRACTION()

$noun\_synsets = wordnet.all\_synsets('n')$
$noun\_antonym\_synset\_pair =$ *empty list*
loop
  for i in $noun\_synsets$ **do**
    $synset = wordnet.synset(i).lemmas()$
    for l *in* $synset$ **do**
      $antonym\_synset = l.antonyms().synset()$
      $antonym\_synset\_pair = [i, antonym\_synset]$
      $noun\_antonym\_synset\_pair.append(antonym\_synset\_pair)$
    end for
  end for
end loop

antonyms().synset() is the function for getting antonym synset A of element s from S.

Next, duplication needs to be removed. And here we use the synset **conformity.n.02** as an example to show the output of each step. There are four words in this synset, and the first word and the third word have their own antonym synsets, **Figure 4.7**. Therefore two antonym synsets are matched, **Figure 4.8**.

```
(‘conformity.n.02.conformity').antonyms( ).synset( )
>> Synset(‘nonconformity.n.04')

('conformity.n.02.conformation').antonyms( ).synset( )
>> No antonym synset

('conformity.n.02.compliance').antonyms( ).synset( )
>> Synset(‘disobedience.n.01')

('conformity.n.02.abidance').antonyms( ).synset( )
>> No antonym synset
```

Figure 4.7: Elements from the Synset **conformity.n.02** and Corresponding Antonym Synsets

```
['conformity.n.02', ‘disobedience.n.01']
['conformity.n.02', 'nonconformity.n.04]
```

Figure 4.8: Antonym Synset Pairs of Synset **conformity.n.02**

As for the verb and adjective, they follow the same algorithm as noun. The**Table 4.4** shows how many antonym synsets pairs we extracted altogether.

Part of Speech	Antonym Synsets Pair Count
NOUN	969
VERB	505
ADJECTIVE	1,947
SUM	3,421

Table 4.4: Antonym Synset Pairs

- ***Constructing Antonym Pairs***

  After all antonym synsets pairs are ready, for each synset in the antonym synsets pair, A and B, each word a from A is the antonym of each word b from B. We calculated the Cartesian product to get all antonym pairs for every antonym synsets pair $[A, B]$. The following pseudocode illustrates the algorithm.

Algorithm 3 Noun Antonym Pair Extraction

ANTONYMPAIREXTRACTION()

$noun\_antonym\_pair$ = *empty list*
loop
  for i in $noun\_antonym\_synsets\_pair$ **do**
    $antonym = itertools.product(i)$
    $noun\_antonym\_pair.append(antonym)$
  end for
end loop

itertools.product() is used for calculating the Cartesian product of two synsets to get the antonym pairs. And the following shows how it works, **Figure 4.9**.

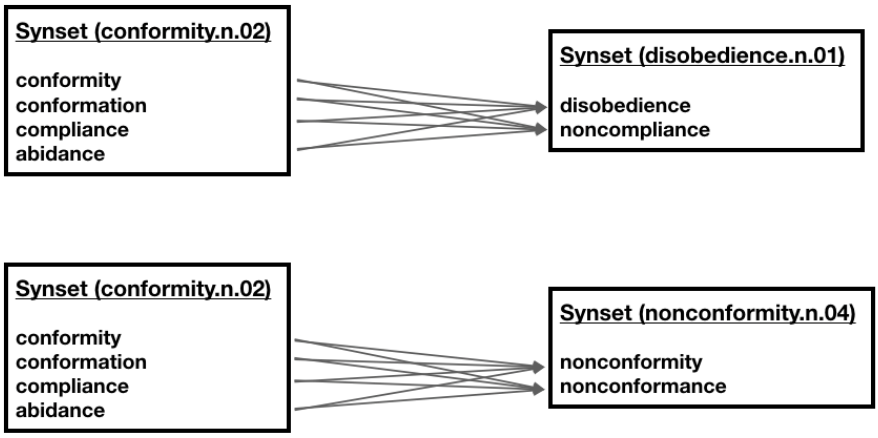

Figure 4.9: Antonym Pairs of Synset **conformity.n.02**

And below, **Figure 4.10** is all antonym pairs for the synset **conformity.n.02**.

('conformity.n.02.conformity', 'disobedience.n.01.disobedience')
('conformity.n.02.conformity', 'disobedience.n.01.noncompliance')
('conformity.n.02.conformation', 'disobedience.n.01.disobedience')
('conformity.n.02.conformation', 'disobedience.n.01.noncompliance')
('conformity.n.02.compliance', 'disobedience.n.01.disobedience')
('conformity.n.02.compliance', 'disobedience.n.01.noncompliance')
('conformity.n.02.abidance', 'disobedience.n.01.disobedience')
('conformity.n.02.abidance', 'disobedience.n.01.noncompliance')

('conformity.n.02.conformity', 'nonconformity.n.04.nonconformity')
('conformity.n.02.conformity', 'nonconformity.n.04.nonconformance')
('conformity.n.02.conformation', 'nonconformity.n.04.nonconformity')
('conformity.n.02.conformation', 'nonconformity.n.04.nonconformance')
('conformity.n.02.compliance', 'nonconformity.n.04.nonconformity')
('conformity.n.02.compliance', 'nonconformity.n.04.nonconformance')
('conformity.n.02.abidance', 'nonconformity.n.04.nonconformity')
('conformity.n.02.abidance', 'nonconformity.n.04.nonconformance')

Figure 4.10: Antonym Pairs of Synset **conformity.n.02**

As for the verb and adjective, they follow the same algorithm as noun. The **Table 4.5** shows how many antonym synset pairs we extracted altogether.

Part of Speech	Antonym Pair Count
NOUN	3,184
VERB	3,307
ADJECTIVE	3,486
SUM	9,977

Table 4.5: Antonym Constraint Pairs

In fact, we also extract hypernym and hyponym constraint pairs, but do not go further in this scenario. It is discussed in the **Chapter 6**.

4.2.2 Distributional Word Vector Preparation

Since the ***ATTRACT-REPEL*** algorithm only modifies the distributional word vectors that exist among synonym and antonym pairs, other word vectors are useless while modeling. And the magnitude of constraints corpus is not at the same level as the magnitude of the distributional word vector collection corpus. The size comparison of these two part can be find in the **Table 4.6**. Therefore, there is no need to load the whole collection and the subset was selected which only contains related word embeddings vectors.

On the other hand, that the model can only retrofit the distributional word vectors which exist among synonym and antonym pairs suggests that our model is powerless on unseen word. But this limitation is resolvable and we discuss it in **Section 6.1**.

	Synonym Pair	Antonym Pair	fastText	word2vec
COUNT	154,776	9,977	1,000,000	3,000,000

Table 4.6: Size of Collections

Moreover, the corpus of the distributional word vector cannot cover the corpus of the synonym and antonym pairs. The relationship is shown below, **Figure 4.11**. There is a number of words from the synonym and antonym pairs that cannot be found in the distributional word vector corpus. Which means a subset of synonym and antonym pairs could also be helpful for reducing the model run time.

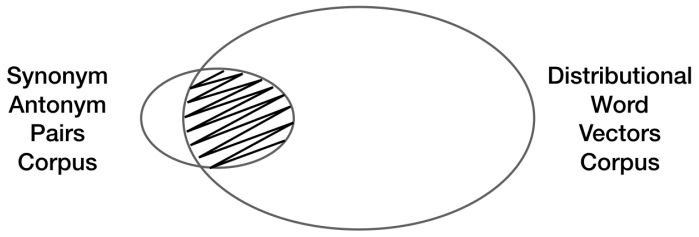

Figure 4.11: The Relationship between Distributional Word Vector Corpus and Synonym Antonym Pair Corpus

It is known that two distributional word vectors collections were employed in this experiment, and they use different way to structure the data and different ways are needed to access the data. For the ***fastText*** distributional word vectors collection, it could be loaded as a dictionary. But for the ***word2vec***, Python package **gensim** is needed to access the data.

The **Algorithm 4** explains the logic how to fetch the coincident subset of the collection of distributional word vectors D and the subset of collection of lexical constraints pairs C. In other word, this algorithm helps to find the shadow part in the **Figure 4.11**. If both words in one synonym or antonym pair appear in the word vector collection, this synonym or antonym pair will be recorded in the subset C. At the same time, two sense level distributional word vectors will be created and

saved into the subset D. To create sense level distributional word vectors, original word embeddings vector was duplicated for each senses of the word. Which means, if a word has three different meanings and all of these meanings are involved in synonym or antonym constraints, there will be three same word embeddings vectors in the subset of the distributional word vector collection, but with different keys. Their keys will contain the synset information. Here, the algorithm only shows how to deal with the synonym constraints subset scenario. For the antonym constraints subset, it followed the same procedure and continued appending the new sense level distributional word vectors into the subset D.

Algorithm 4 Synonym Constraint Subset and Distributional Word Vector Subset Preparation

SUBSETSPREPARATION()

$sense\_collection =$ ***empty list***
$synonym\_subset =$ ***empty list***
loop
  for $pair$ in $synonym\_pairs$ **do**
    $sense_1\ is\ the\ first\ word\ in\ the\ synonym\ pair$
    $sense_2\ is\ the\ second\ word\ in\ the\ synonym\ pair$
    $word_1\ is\ the\ sense_1\ removing\ the\ synset\ information$
    $word_2\ is\ the\ sense_2\ removing\ the\ synset\ information$
    if $word_1$ in $distributional\ word\ vector\ collection$
    and $word_2$ in $distributional\ word\ vector\ collection$ **then**
      $synonym\_subset.append(pair)$
      if $sense_1$ not in $sense\_collection$ **then**
        $sense_collection.append$
        $([sense_1, distributional\ word\ vector\ of\ word_1])$
      end if
      if $sense_2$ not in $sense\_collection$ **then**
        $sense_collection.append$
        $([sense_2, distributional\ word\ vector\ of\ word_2])$
      end if
    end if
  end for
end loop

We continue using the synset **conformity.n.02** as the example to show how the algorithm works and the difference between the two distributional word vector collection, ***fastText*** and ***word2vec***.

- ***fastText***

  The word **nonconformance** does not exist in the ***fastText***, so four antonym constraint pairs that contain the word **nonconformance** are not useful in this scenario and will not be added into the antonym constraint subset. Except this word, all other word can be fetched in the collection. The subset of the synset **conformity.n.02** antonym pairs is shown below, **Figure 4.12**.

```
['conformity.n.02.conformity', 'disobedience.n.01.disobedience']
['conformity.n.02.conformity', 'disobedience.n.01.noncompliance']
['conformity.n.02.conformation', 'disobedience.n.01.disobedience']
['conformity.n.02.conformation', 'disobedience.n.01.noncompliance']
['conformity.n.02.compliance', 'disobedience.n.01.disobedience']
['conformity.n.02.compliance', 'disobedience.n.01.noncompliance']
['conformity.n.02.abidance', 'disobedience.n.01.disobedience']
['conformity.n.02.abidance', ‘disobedience.n.01.noncompliance']

['conformity.n.02.conformity', 'nonconformity.n.04.nonconformity']
['conformity.n.02.conformation', 'nonconformity.n.04.nonconformity']
['conformity.n.02.compliance', 'nonconformity.n.04.nonconformity']
['conformity.n.02.abidance', ‘nonconformity.n.04.nonconformity']
```

Figure 4.12: The Subset of Antonym Pairs of Synset **conformity.n.02**

As for the subset of the distributional word vector, it is better to show from a word perspective instead of from a synset perspective. Thus, we use the word **conformity** as the instance to show the changes. There is only one word embeddings vector in the collection for the word **conformity**. But because the word **conformity** has five different meanings, and all of them are associated with one or more than one synonym or antonym constraint pairs, the vector of the word **conformity** from the original collection has be duplicated five times and saved into the subset of the distribution word vector collection. The **Figure 4.13** shows the duplication. It can be seen that the key of the sense level distributional word vector does not only contain the word itself, but also subjoins the synset information before the word.

['conformity.n.01.conformity', '-0.085332 -0.09469 -0.39808 0.32941 -0.20555 0.066287 ......']
['conformity.n.02.conformity', '-0.085332 -0.09469 -0.39808 0.32941 -0.20555 0.066287 ......']
['conformity.n.03.conformity', '-0.085332 -0.09469 -0.39808 0.32941 -0.20555 0.066287 ......']
['ossification.n.04.conformity', '-0.085332 -0.09469 -0.39808 0.32941 -0.20555 0.066287 ......']
['accord.n.02.conformity', '-0.085332 -0.09469 -0.39808 0.32941 -0.20555 0.066287 ......']

Figure 4.13: The Sense Level Distributional Vectors of Word **conformity** from ***fastText*** before Retrofitting

The following **Table 4.7** is the overview of the subset of the synonym constraint pairs, the subset of antonym constraint pairs and the subset of the distributional word vectors collection of the ***fastText***. It might be confused that the synonym pairs subset is bigger than the distributional word vectors subset. However, there is no connection between the bigness of these two subset. Both of them contain a large amount of duplication. Although it seems like a large number of linguistic constraints, there is only 35.10% synonym pairs and 57.50% antonym pairs that can be fetched in the ***fastText***. This low coverage is one of the core limitation of many retrofitting methods. Moreover, it can be seen that only 5.43% of the distributional word vectors from the collection is useful in our experiment scenario, which shows the importance of creating the subset of the distributional word vectors to decrease the model run time.

	Subset	Original Set	Percentage
Synonym Pair	54,319	154,776	35.10%
Antonym Pair Subset	5,737	9,977	57.50%
Word Vector Subset	54,278	1,000,000	5.43%

Table 4.7: Subsets Overview of the ***fastText***

- ***word2vec***

  The word **nonconformance** exists in the ***word2vec***, so the subset of the antonym constraint pairs of the synset **conformity.n.02** was not shrunken and kept as its original manner, **Figure 4.10**.

  As for the subset of the distributional word vector, compared to the ***fastText***, the only difference is that it has different word embeddings vector, shown in **Figure 4.14**. The vector of the word **conformity** from the original collection

has be duplicated five times and saved into the subset of the distribution word vector collection.

```
['conformity.n.01.conformity', '-0.104004 -0.055908 0.091309 0.143555 -0.269531 0.222656 ......']
['conformity.n.02.conformity', '-0.104004 -0.055908 0.091309 0.143555 -0.269531 0.222656 ......']
['conformity.n.03.conformity', '-0.104004 -0.055908 0.091309 0.143555 -0.269531 0.222656 ......']
['ossification.n.04.conformity', '-0.104004 -0.055908 0.091309 0.143555 -0.269531 0.222656 ......']
['accord.n.02.conformity', '-0.104004 -0.055908 0.091309 0.143555 -0.269531 0.222656 ......']
```

Figure 4.14: The Sense Level Distributional Vectors of Word **conformity** from ***word2vec*** before Retrofitting

The following **Table 4.8** is the overview of the subset of the synonym constraint pairs, the subset of antonym constraint pairs and the subset of the distributional word vectors collection of the ***word2vec***. It can be noticed that the ***word2vec*** can cover more synonym constraint pairs but less antonym constraints pairs than the ***fastText***.

	Subset	Original Set	Percentage
Synonym Pair Subset	57,851	154,776	37.38%
Antonym Pair Subset	5,427	9,977	54.40%
Word Vector Subset	60,840	3,000,000	2.03%

Table 4.8: Subsets Overview of the ***word2vec***

In addition, as the ***word2vec*** also be chosen to do the evaluation, subset of the distributional word vectors collection of the evaluation dictionary also be created using the same way. But we will not paint it in this paper.

4.2.3 Model Setup

The ***ATTRACT-REPEL*** model needs four data inputs and six hyper-parameter inputs.

Except the synonym linguistic constraints subset, antonym linguistic constraints subset and the distributional word vectors collection subset, which were extracted before, the original distributional word vector is still needed as the input. But it is only for the evaluation procedure, not for the modeling. So it is not contrary to our original intention of reducing time cost.

As a matter of fact, the paper [27] tunes the hyper-parameters on the ***WordSim-353*** gold-standard association data set [8] to achieve the best Spearmans correlation score and provided a set of values of the hyper-parameters from their best experiment results. We tried several series of the hyper-parameter values, but found that the hyper-parameter set they furnished is already good enough for our experiment. We tended to increase the iteration times, but the Spearman's score did not have any alteration. The **Table 4.9** below shows the hyper-parameter values they supplied, which were also used in our experiment.

Hyper-parameter	Formula Representation	Value
Similarity Margin	δ_{syn}	0.6
Antonym Margin	δ_{ant}	0.0
Synonym Batch Size	k_1	50
Antonym Batch Size	k_2	50
Regularization Constant	λ_{reg}	10^{-9}
Max Iteration		5

Table 4.9: Hyper-parameter Setting of ***ATTRACT-REPEL*** Model

After retrofitting the input distributional word vectors by injecting the external synonym and antonym constraint pairs, we gained a collection of the sense level distributional embeddings vectors D'. It has the same size of the input subset D, but does not contain the duplication of the embeddings vectors as they have already been retrofitted. The **Figure 4.15** below shows the word embeddings vectors of different senses of the word **conformity** from the collection ***fastText*** after modification. And the **Figure 4.16** shows the retrofitted vectors from the collection ***word2vec***.

```
['conformity.n.01.conformity', '-0.027047 -0.022307 -0.0946 0.082171 -0.055267 0.015942 ......']
['conformity.n.02.conformity', '-0.024492 -0.021477 -0.103277 0.089618 -0.056095 0.009222 ......']
['conformity.n.03.conformity', '-0.032103 -0.031566 -0.110651 0.077412 -0.047048 0.030087 ......']
['ossification.n.04.conformity', '-0.017887 -0.022576 -0.098301 0.092162 -0.061606 0.015916 ......']
['accord.n.02.conformity', '-0.025479 -0.032043 -0.08018 0.080559 -0.054728 0.019057 ......']
```

Figure 4.15: The Sense Level Distributional Vectors of Word **conformity** from ***fastText*** after Retrofitting

['conformity.n.01.conformity', '-0.050863 -0.029527 0.028916 0.040855 -0.081458 0.073897 ......']
['conformity.n.02.conformity', '-0.069294 1.3e-05 -0.000247 -0.000456 -0.066432 0.082264 ......']
['conformity.n.03.conformity', '-0.004857 -0.01759 0.014162 0.049529 -0.06326 0.101392 ......']
['ossification.n.04.conformity', '-0.041196 -0.010966 0.011341 0.046379 -0.099656 0.074704 ......']
['accord.n.02.conformity', '-0.052057 -0.024819 0.045734 0.045791 -0.064724 0.077489 ......']

Figure 4.16: The Sense Level Distributional Vectors of Word **conformity** from ***word2vec*** after Retrofitting

Chapter 5

Evaluation and Discussion

5.1 Evaluation Methodology

It is possible to evaluate the specialized word representations by reproducing the approach presented in paper [33], which could examine the performance of the word representations in the word similarity and the cross-level semantic similarity measurement frameworks. In this section, we depict the evaluation methods and the benchmarks.

5.1.1 Word Similarity

It is easy to evaluate the performance of the retrofitting embeddings vectors at the word level, since there are several gold standard evaluation benchmark data sets from previous work.

Benchmarks

Seven data sets were considered as word similarity measurement benchmark. These data sets contain pairs of entities followed by scores.

- ***SimLex-999***

  SimLex-999 [1] is a standard resource for evaluating distributional semantic models. It contains 999 word pairs, and for each word pair, it attaches with a numerical similarity score between 0 and 10. ***SimLex-999*** demonstratively quantifies similarity rather than association or relatedness so that the word

[1] https://fh295.github.io/simlex.html

pairs in it which have high numerical similarity score, are similar but not associated [13].

SimLex-999 is the main benchmark data set we use, as it covers noun, verb and adjective, not just one of them. Moreover, it has the context free similarity ratings of word pairs of different POS categories and concreteness levels, which enables fine-grained analyses of the performance of our semantic model.

- ***WordSim-353***

  WordSim-353 [2] is the most commonly used evaluation gold standard data set for semantic models. For each word pair, it also attaches with a numerical score between 0 and 10. But it is ambiguous with respect to similarity and association. It actually measures the association rather than similarity. Thus, the ***WordSim-353*** is used as an evaluation reference due to its low performance of measuring similarity [8] [13].

- ***SimVerb-3500***

  SimVerb-3500 [3] is a verb evaluation resource with numerical ratings for the similarity of 3,500 pairs. However, as a matter of fact, there is only 24,577 verb synonym constraint pairs over 154,776 synonym constraint pairs, around 15.88%, and 3,307 verb antonym constraint pairs over 9,977 antonym constraint pairs, around 33.15%. Since the verb is not the main composition of the constraints collection, the ***SimVerb-3500*** can only become an evaluation reference of our experiment results [9].

- ***MEN-3K***

  MEN-3K [4] consists of 3,000 word pairs, randomly selected from words that have high frequency of occurrence, which consists of word pairs with similarity score in a 50 scale. Instead of asking annotators an score reflecting how much a word pair is semantically associated, each pair was randomly matched with a comparison pair and then given a score based on the comparison pair. Different from other benchmark data set, the word pairs in the ***MEN-3K*** have different parts of speech. For instance, it could contain a word pair $(noun, verb)$ [4].

- ***RG-65***

[2] http://www.cs.technion.ac.il/ gabr/resources/data/wordsim353/
[3] http://people.ds.cam.ac.uk/dsg40/simverb.html
[4] https://staff.fnwi.uva.nl/e.bruni/MEN

RG-65 is a data set containing 65 English noun pairs with the similarity score between 0 and 4, from highly synonymous pairs to semantically unrelated pairs. As it only has noun word pairs and limited amount of data, we also considered it as an evaluation reference [37].

- ***YP-130***

  YP-130 is a data set which has 130 word pairs following with the numerical scores, which focuses on verb similarity, and we also used it as an evaluation reference [46].

- ***SCWS-2003***

  Stanford Contextual Word Similarity [5] is a data set with human judgments on 2003 pairs of words in sentential context,and we also used it as an evaluation reference [36].

Evaluation at the Word Level

Correlation refers to the association between the observed values of two variables. Variables may have a positive association, which means that if the value of one variable increases, the value of the other increases. On the other hand, the association can also be negative, which means that if the value of one variable increases, the values of other variables decrease. Correlation weights this relationship, quantifying it between the values -1 to 1 for sheer negatively correlated and perfectly positively correlated.

Rank correlation is a method of measuring the association between variables using an ordinal relationship between values rather than a specific value. It can be calculated to quantify the association between the rankings of two variables.

The Spearman's Rank Correlation Coefficient ρ is used to discover the strength of a link between two sets of data, assessing monotonic relationships. The Spearman correlation between two variables will be high when observations have a similar rank between the two variables, and will be low when observations have a dissimilar rank between the two variables. In our experiment scenario, the higher Spearman's score means that the rank of the similarities and the rank of the human-assigned scores are more similar.

The Spearman's Rank Correlation Coefficient is wildly used in NLP tasks. Utilizing the benchmark data sets above, we calculate the Spearman's score between the vector of the word pairs similarities and the vector of the assigned scores following the word pairs to evaluate the improvement that our experiment achieved.

[5] http://www.bigdatalab.ac.cn/benchmark/bm/dd?data=Stanford

In all our evaluation approaches, the cosine distance is used as the similarity measure.

The D is used to represent the distributional word vectors subset, the input of the ***ATTRACT-REPEL*** model, while D' is the output of the model, obtained from the D by injecting the constraints knowledge. Moreover, W is the word in the vocabulary of D (D' has the same vocabulary), and S is the sense level vector of the word W.

We note that the subset D has the format of the sense level distributional embeddings vectors collection, but as a matter of fact, it is at the word level because all senses of one word have the same vectors as we duplicate them when prepare the subset of the distributional word vectors collection. And all the Spearman's score of the data set D are considered as the baseline of the experiment.

- ***Average Similarity***

  For each word pair (W_1, W_2) from the benchmark data set, we fetch all sense level vectors $\{S_{11}, S_{12}...\}$ for W_1 and $\{S_{21}, S_{22}...\}$ for W_2 and calculate the similarities for all possible combinations of the sense vectors sets $\{S_{11}, S_{12}...\}$ and $\{S_{21}, S_{22}...\}$. Then the average of all combinations' similarities s computed as the similarity of the word pair (W_1, W_2).

- ***Maximum Similarity***

  For each word pair (W_1, W_2) from the benchmark data set, we fetch all sense level vectors $\{S_{11}, S_{12}...\}$ for W_1 and $\{S_{21}, S_{22}...\}$ for W_2 and calculate the similarities for all possible combinations of the sense vectors sets $\{S_{11}, S_{12}...\}$ and $\{S_{21}, S_{22}...\}$. Unlike the **Average Similarity** method, **Maximum Similarity** uses the maximum similarity of all combinations' similarities as the similarity of the word pair (W_1, W_2).

The similarity of each word pair gained via above approaches is used to calculate the Spearman's Rank Correlation Coefficient score. In addition, the Spearman's score is computed twice, once for the word level distributional word vectors subset D, and once for the sense level distributional word vectors D'. These two scores describe how well the approach actually capture the meanings of particular word.

5.1.2 Cross-Level Semantic Similarity

In addition to the word similarity evaluation tactics, we also consider the cross-level semantic similarity evaluation strategy which is presented in the paper [33] [16].

Benchmark

For the cross-level semantic similarity evaluation strategy, only one benchmark data set is considered.

- ***Test Set of SemEval-2014 Task 3***

  Test Set of SemEval-2014 Task 3 [6] has 500 instances. For each instance, it contains a $(word, sense)$ pair following by a human-assigned scores between 0 and 4. And the following is an instance from it

  $$(project\#n,\ enterprise\#n\#1)$$

  Each instance pair in this data set contain a word and a sense of another word. Therefore, evaluation on this data set is not purely measuring at the word level or the sense level. This is the reason why this method is named as cross-level semantic similarity.

Evaluation at the Cross Level

Same as the word similarity evaluation strategy, the Spearman's Rank Correlation Coefficient is used to measure the performance of the retrofitting distributional word vectors. And in all our evaluation approaches, the cosine distance is used as the similarity measure. Let (W, W_s) be the $(word, sense)$ pair, and S_w be the sense vector of the word W and S be the sense vector of W_s.

- ***Average Similarity***

  For each word sense pair (W, W_s) from the benchmark data set, we fetch all sense level vectors $\{S_{w_1}, S_{w_2}...\}$ for W, and calculate the similarities for all possible combinations of the sense vectors sets $\{S_{w_1}, S_{w_2}...\}$ and S. Then the average of all combinations' similarities is computed as the similarity of the word sense pair (W, W_s).

- ***Maximum Similarity***

  For each word sense pair (W, W_s) from the benchmark data set, we fetch all sense level vectors $\{S_{w_1}, S_{w_2}...\}$ for W, and calculate the similarities for all possible combinations of the sense vectors sets $\{S_{w_1}, S_{w_2}...\}$ and S. Unlike the **Average Similarity** method, **Maximum Similarity** uses the maximum similarity of all combinations' similarities as the similarity of the word pair (W, W_s).

[6] http://alt.qcri.org/semeval2014/task3/index.php?id=data-and-tools

- ***Sense to Word Similarity***

  For each word sense pair (W, W_s) from the benchmark data set, the original distributional word vector is used to represent the word W, which means the similarity is calculated between the word level vector of W and the sense level vector of W_s directly.

- ***Sense to Aggregated word senses Similarity***

  For each word sense pair (W, W_s) from the benchmark data set, we fetch all sense level vectors $\{S_{w_1}, S_{w_2}...\}$ for W and compute the centroid vector of $\{S_{w_1}, S_{w_2}...\}$. Then the similarity is calculated between the centroid vector of word W and the sense vector S.

The same as the procedure in word similarity evaluation, the similarity of each word pair gained via above approaches is used to calculate the Spearman's Rank Correlation Coefficient score. And the Spearman's score is also computed twice, once for the distributional word vector data set D, and once for the data set D'. These two scores describe how well the approach actually capture the meanings of particular word.

5.2 Results and Discussion

As duplication are used to represent different senses for each word in the original distributional word vectors subset D as the input of the ***ATTRACT-REPEL*** model, the Spearman's Rank Correlation Coefficient score of D is the delegate of the monotonic relationship of the word pair similarities and the human-assigned scores at the word level. As for the output data set D' which contains different sense vectors for each word after modeling, the Spearman's score is at the sense level.

5.2.1 Evaluation Result on *fastText*

The **Table** 5.2, 5.3 and 5.4 below show the evaluation results on the distributional word vectors collection ***fastText***.

data set	Fetched word pairs	Word Fetched in the Subset D	Word Fetched in the Original Collection
SimLex-999	999 (100%)	1788 (89.49%)	210 (10.51%)
WordSim-353	332 (94.05%)	540 (81.33%)	124 (18.67%)
SimVerb-3500	3498 (99.94%)	6887 (98.44%)	109 (1.56%)
MEN-3K	3000 (100%)	4909 (81.82%)	1091 (18.18%)
RG-65	65 (100%)	124 (96.92%)	6 (3.08%)
YP-130	130 (100%)	259 (99.62%)	1 (0.38%)
SCWS-2003	1973 (98.50%)	3522 (89.25%)	424 (10.75%)
TestSet-500	361 (77.20%)	377 (52.22%)	345 (47.78%)

Table 5.1: Word Coverage of Each Standard Data Set in ***fastText***

The **Table 5.1** shows the overview of the vocabulary coverage in the collection ***fastText*** of seven standard data sets. Each element from the pairs is considered as an individual word, which means we ignore the repeat of the word. And we need to note that the percentage of the word fetched in the distributional word vectors collection subset D is calculated on the amount of the fetched word pairs.

Except the ***Test Set of SemEval-2014 Task 3***, the word pairs from other seven data sets could be covered well in the vectors collection ***fastText***, and more than 80% of the words could be found in the sense level collection D and D'. As for the data set ***Test Set of SemEval-2014 Task 3***, 77.20% word pairs can be fetched in the ***fastText***, and only 52.22% words are found in the sense level distributional word vector collection D and D', which could influence the evaluation results under cross-level semantic similarity strategy.

Word Similarity

data set	Word-Level Spearman Score ($\rho \times 100$)	Sense-Level Spearman Score ($\rho \times 100$)	Change $\Delta\rho$ ($\Delta\rho \times 100$)
SimLex-999	38.1	44.7	+ 6.6
WordSim-353	66.6	69.6	+ 3
SimVerb-3500	26.1	33.4	+ 7.3
MEN-3K	76.3	77.3	+ 1
RG-65	80.1	81.5	+ 1.4
YP-130	53.1	59.8	+ 6.7
SCWS-2003	67	67.5	+ 0.5

Table 5.2: Word Similarity (Average) Evaluation Result of ***fastText***

The **Table 5.2** shows that the Spearman's score of every standard data set is increased, which means the rank of the average similarities of the each word pair is much closer to the rank of the human-assigned scores. In other words, the sense representations specialization achieve significant improvements over seven standard data sets. And this highlights the fact that the sense specialization can benefit the word representation.

We can see that the Spearman's score of the ***SimLex-999*** increases 6.6, which is larger than the increments of the Spearman's score of ***WordSim-353***, ***MEN-3K***, ***RG-65*** and ***SCWS-2003***. When introducing the standard data set ***SimLex-99***, we mention that it quantifies similarity rather than association or relatedness, which means it assigns relatively low scores to antonym word pairs. In fact, the ***ATTRACT-REPEL*** model tries to keep the antonym pairs as far as possible so that the antonym pairs from the standard data set will get relatively low similarity scores. That is the reason why the significant performance improvement on ***SimLex-999*** could be observed.

It could be also noticed that the increments on the standard data sets ***SimVerb-3500*** and ***YP-130*** which focus on the verb pairs, are larger than others. We conjecture that our model performs well on the verb senses specialization.

Moreover, we can see that the distributional word vectors collection ***fastText*** already achieves high Spearman's score at the word level on the gold standard data set ***WordSim-353***, ***MEN-3K***, ***RG-65*** and ***YP-130***.

data set	Word-Level Spearman Score ($\rho \times 100$)	Sense-Level Spearman Score ($\rho \times 100$)	Change $\Delta\rho$ ($\Delta\rho \times 100$)
SimLex-999	38.1	45.3	+ 7.4
WordSim-353	66.6	70	+ 3.4
SimVerb-3500	26.1	33.9	+ 7.8
MEN-3K	76.3	76.7	+ 0.4
RG-65	80.1	80.5	+ 0.4
YP-130	53.1	56.1	+ 3
SCWS-2003	67	66.8	- 0.2

Table 5.3: Word Similarity (Maximum) Evaluation Result of ***fastText***

The **Table 5.3** shows the Spearman's scores of each evaluation data set under the maximum word similarity setting. Except the the Spearman's score of ***SCWS-2003*** data set, all Spearman's scores of other data sets are raised. Unfortunately, our sense specialization vectors are weak on this data set following the maximum word similarity evaluation strategy.

Cross-Level Semantic Similarity

Similarity	Word-Level Spearman Score ($\rho \times -100$)	Sense-Level Spearman Score ($\rho \times -100$)	Change $\Delta\rho$ ($\Delta\rho \times -100$)
Average	7.8	8.2	+ 0.4
Maximum	7.8	7.6	- 0.2
Sense to Word	7.8	7.6	- 0.2
Sense to Aggregated word sense	7.8	7.6	- 0.2

Table 5.4: Cross-level Semantic Similarity Evaluation Result of ***fastText***

The **Table 5.4** shows the cross-level semantic similarrity evaluation results of ***fastText***. Only the Spearman's score under the average strategy slightly increases. The low performance could be cased by the low coverage of the word sense pairs of the ***Test Set of SemEval-2014 Task 3*** in the collection ***fastText***.

5.2.2 Evaluation Result on *word2vec*

The **Table** 5.5, 5.6, 5.7, 5.8, 5.9 and **??** below show the evaluation results on the distributional word vectors collection ***word2vec***.

It is different from the distributional word vectors collection ***fastText*** that there is a great quantity of previous work on ***word2vec*** where comparisons could be available.

data set	Fetched word pairs	Word Fetched in the Subset D	Word Fetched in the Original Collection
SimLex-999	999 (100%)	1793 (89.74%)	205 (10.26%)
WordSim-353	350 (99.15%)	569 (81.29%)	131 (18.71%)
SimVerb-3500	3500 (100%)	6889 (98.41%)	111 (1.59%)
MEN-3K	2946 (98.2%)	4867 (82.6%)	1025 (17.4%)
RG-65	65 (100%)	123 (94.62%)	7 (5.38%)
YP-130	130 (100%)	259 (99.62%)	1 (0.38%)
SCWS-2003	1982 (98.95%)	3524 (88.9%)	440 (11.1%)
TestSet-500	390 (78%)	393 (50.38%)	387 (49.62%)

Table 5.5: Word Coverage of Each Standard data set in ***word2vec***

The **Table** 5.5 shows the overview of the vocabulary coverage in the collection ***word2vec*** of seven standard data sets. Each element from the pairs is considered as an individual word, which means we ignore the repeat of the word. And we need to note that the percentage of the word fetched in the distributional word vectors collection subset D is calculated on the amount of the fetched word pairs.

It is almost the same as the **Table** 5.1. Except the Test Set of SemEval-2014 Task 3, the word pairs from other seven data sets could be covered well in the vectors collection ***word2vec***, and more than 80% of the words could be found in the sense level collection D and D'. As for the data set Test Set of SemEval-2014 Task

3, 78% word pairs can be fetched in the ***fastText***, which is slightly higher than that in the ***fastText***, and only 50.38% words are found in the sense level distributional word vector collection D and D', which could also influence the evaluation results under cross-level semantic similarity strategy.

Word Similarity

data set	Word-Level Spearman Score ($\rho \times 100$)	Sense-Level Spearman Score ($\rho \times 100$)	Change $\Delta\rho$ ($\Delta\rho \times 100$)
SimLex-999	44.3	53.6	+ 9.3
WordSim-353	66.6	69.9	+ 3.3
SimVerb-3500	36.7	47.2	+ 10.5
MEN-3K	77.1	77.3	+ 0.2
RG-65	76	77.2	+ 1.2
YP-130	55.8	65.9	+ 10.1
SCWS-2003	66.9	67.4	+ 0.5

Table 5.6: Word Similarity (Average) Evaluation Result of ***word2vec***

The **Table 5.6** shows that the Spearman's score of every standard data set is increased, which means the sense representations specialization achieve significant improvements over seven standard data sets.

Compared to the **Table 5.2**, in general, the collection ***word2vec*** achieves higher Spearman's score than the collection ***fastText*** even at the word level. This means the distributional word vector collection ***word2vec*** performs better than the ***fastText***.

The more important point is that the increments of the Spearman's scores on the standard data set ***SimLex-999***, ***SimVerb-3500*** and ***YP-130*** are bigger than those of the collection ***fastText***. It shows the strongness of our approach on the collection ***word2vec***.

Model	Word-Level Spearman Score ($\rho \times 100$)	Sense-Level Spearman Score ($\rho \times 100$)	Change $\Delta\rho$ ($\Delta\rho \times 100$)
ATTRACT-REPEL	44.3	53.6	+ 9.3
DECONF	44.2	51.7	+ 7.5
Pilehvar and Navigli (2015)	-	43.6	-

Table 5.7: Word Similarity (Average) on ***SimLex-999*** Evaluation Result Comparison of ***word2vec***

The word level Spearman's scores of the ***ATTRACT-REPEL*** model and the ***DECONF*** model are slightly different. As only Noun, Verb and Adjective synonym and antonym constraint pairs are extracted from the lexical resource, Adjective Statement and Adverb have been ignored. This could cause the **0.1** ($\rho \times 100$) deviation, but it cannot have much impact on analyzing our evaluation results.

Compared to the evaluation results of previous models : the ***DECONF*** model [33] and the ***Pilehvar and Navigli (2015)*** [34], from the **Table 5.7**, we can see that our model reaches the highest Spearman's score **53.6** ($\rho \times 100$) over three models on the standard data set ***SimLex-999***.

This highlight the fact that the specialization approach highly beneficial the sense level distributional embeddings vectors. It outperforms the two recent state-of-the-art sense representation techniques. The Spearnman's score **53.6** ($\rho \times 100$) proves that our specialization approach make better use of the external lexical knowledge from the ***WordNet*** and effectively capturing the semantic meanings of the words.

We note that it is impossible to compute the Spearman's score for the ***Pilehvar and Navigli (2015)*** model, since this model does not train on any pre-trained distributional word embeddings vectors and totally dependent on the semantic network of ***WordNet*** [34].

data set	Word-Level Spearman Score ($\rho \times 100$)	Sense-Level Spearman Score ($\rho \times 100$)	Change $\Delta\rho$ ($\Delta\rho \times 100$)
SimLex-999	44.3	53.1	+ 8.8
WordSim-353	66.6	68.6	+ 2
SimVerb-3500	36.7	44.5	+ 7.8
MEN-3K	77.1	74.6	- 2.5
RG-65	76	73.5	- 2.5
YP-130	55.8	57.8	+ 2
SCWS-2003	66.9	65.1	- 1.8

Table 5.8: Word Similarity (Maximum) Evaluation Result of ***word2vec***

The **Table 5.8** shows the Spearman's scores of each evaluation data set under the maximum word similarity setting. There are three Spearman's scores which are decreased after the specialization of the embeddings vectors. Unfortunately, our sense specialization vectors are weak on the data set ***MEN-3K***, ***RG-65*** and ***SCWS-2003***, following the maximum word similarity evaluation strategy. It is not abnormal that the sense representation techniques could not achieve good performance than the baseline, and precedents can be found from the paper [33].

Cross-Level Semantic Similarity

Similarity	Word-Level Spearman Score ($\rho \times -100$)	Sense-Level Spearman Score ($\rho \times -100$)	Change $\Delta\rho$ ($\Delta\rho \times -100$)
Average	9	9.3	+ 0.3
Maximum	8.9	8.4	- 0.5
Sense to Word	8.9	8.4	- 0.5
Sense to Aggregated word sense	8.9	8.4	- 0.5

Table 5.9: Cross-level Semantic Similarity Evaluation Result of ***word2vec***

The **Table 5.9** shows almost the same evaluation results as ***fastText***. Only the Spearman's score under the average strategy slightly increases. The low performance could be cased by the low coverage of the word sense pairs of the ***Test Set of SemEval-2014 Task 3*** in the collection ***word2vec***.

Compared to the evaluation results in the **Paper** [33], our outcomes differ greatly from theirs. They fetch 474 word sense pairs of original 500 pairs, but only 390 pairs are found by our model. Comparison on different basic is meaningless.

Chapter 6

Conclusion

6.1 Future Work

Although the combination of the past work that we design already outperform recent state-of-the-art models on the standard gold evaluation data set ***SimLex-999***, such as the ***DECONF*** model [33] and the ***Pilehvar and Navigli (2015)*** [34], there is still a number of works could be interesting to go further. In this section, we discuss several future works that could be implemented later.

Except the synonym and antonym, there are some other relationships between words which can be also employed as the constraints that can be exploited in the specialization model. For instance, there is an association between hypernym and hyponym although it is not as strong as that between synonym words, but still considerable.

And it could be noticed that there are some limitations of our approach. For now, only words that appear in the lexical constraints vocabulary can have sense level representation, but it is not possible for other words. Moreover, the experiment and the evaluation only have the English collection as the input. It would be great if our approach can be extend to other languages. However, fortunately, there are also some solutions which could ameliorate the situation.

As for the evaluation, more complicated measurements could be implemented to show the power of the sense vector space that our approach generated.

All the possible amelioration discussed above are looking forward to implemented later.

6.1.1 Hypernym and Hyponym

As for the linguistic constraints, not only the synonym and antonym pairs could be extracted from the WordNet, but also the hypernym pairs which contain a hyper-

nym word and a hyponym word. A hyponym word is a word which is included in another word, its hypernym word. In simple terms, there is a type-of relationship between the hypernym and the hyponym. The hypernym and hyponym pairs could be generated as a type-of relationship constraints which could be also inserted into the specialization model to train the sense level representations.

For example, the word **color** could be a hypernym, and then the word **green** is a hyponym word. The hypernym and hyponym pair $[color, green]$ which contains a type-of relationship could be constructed as a linguistic constraint to retrofit the word level representations.

Actually, we also generate the hypernym and hyponym pairs when preparing the synonym and antonym constraints pairs from the ***WordNet***. The same algorithm for creating the antonym constraint pairs is also used here to get the hypernym and hyponym pairs, all combinations of the words from the hypernym set and the words from the hyponym set. The same as other lexical constraints, only the constraints of noun, verb and adjective are concocted. The **Table 6.1** shows how many hypernym and hyponym pairs we extracted altogether.

Part of Speech	Hypernym and Hyponym Pair Count
NOUN	261,308
VERB	68,088
ADJECTIVE	0
SUM	329,396

Table 6.1: Hypernym and Hyponym Pairs

The relationship between hypernym and the hyponym is not as strong as the synonym, but could also be considered as an association. Which means it could also be used as the lexical constraints from the external lexical knowledge to be the input of the retrofitting model. We look forward to the result after injecting the hypernym and the hyponym constraint pairs together with the synonym and the antonym constraint pairs.

6.1.2 Unseen Words

As it can be seen from the **Section 4.2.2**, only the dummy sense level representations which occur in the lexical constraint vocabulary, are created, not all meanings of all words from the collection of the pre-trained distributed word representations. We call the words which are included in the lexical constraint vocabulary as seen words, and other words as unseen words.

The specialization model we select can only work well on the seen words but do nothing with the unseen words. This is the limitation of the ***ATTRACT-REPEL*** retrofitting model. But when it is evaluated by the Dialogue State Tracking downstream task [27], the delexicalization-based ***DST*** models are used to overcome this limitation by displacing the occurrences of the ontology values with generic tags [11] [12] [25] [26].

Inspirited by the explicit retrofitting model ***ER*** [10], we could use a model which can learn the behavior of the retrofitting model and then instigate the retrofitting model to do the same action to the unseen words. Then not only the seen words can gain the sense level representations, but also the unseen words can obtain sense level representations by the learning model. This could be implemented later which can enhance the capability of specialization model.

6.1.3 Multiple Languages

It is mentioned at the beginning of the **Chapter 4**, English is chosen as the target language of the whole experiment, and the collections of the pre-trained distributional word vectors and the lexical constraints are all in English.

Since the retrofitting model ***ATTRACT-REPEL*** we select can facilitate the use of the lexical constraints not only form just single language resources but also cross-lingual resources. And the cross-lingual vector space produced by this model performs well in the downstream tasks [27].

Therefore, it is not difficult to expand the implementation from single language to multiple languages even cross languages. This could be implemented later which can enhance the capability of specialization model.

6.1.4 Downstream Task Evaluation

The evaluation methods introduced in the **Chapter 5** are basic ones, and there is a number of downstream tasks that could be implemented as the measurement to survey the ability of the model.

Two downstream tasks introduced in the paper [10], the Lexical Text Simplification and the Dialog State Tracking are used to distinguishing semantic similarity from semantic relatedness. The Lexical Text Simplification targets to replace the complex words with the simple ones without changing the meaning of the original text. Our work which aims to generate sense level representations for individual word, might provide improvement when creating true synonym candidate replacements. As for the Dialog State Tracking, it aims to understand the language by capturing the objective from the user and updating the dialog state. Our sense level

representations for individual word could be helpful when discriminating similarity from relatedness.

However, all the visions above need further works to prove, and the improvement of the results of each downstream tasks could be worth expecting.

6.2 Summary

In this paper, we focus on ameliorating the limitation of the word embeddings technique. Simply putting, one word can have several different meanings in reality, but the word representation technique cannot delineate the it as it only provide one word representation vector for each word. Fortunately, it could be solved by the sense embeddings technique, also referred as the sense level specialization.

We propose a combination of the previous work to generate sense level representations to solve the meaning conflation deficiency of the word level representation. Generally, there are two main solutions, the Joint Specialization Approach and the Post-Processing Approach. Since the second one performs better under most circumstanceswe choose this one and design the combination of different passed works from different domains.

For achieving the objective of creating the sense embedding, the collection of the pre-trained distributional word vectors, external linguistic knowledge and the specialization model are needed. The ***fastText*** and the ***word2vec*** are chosen as the word embeddings collections. And the ***WordNet*** is used to generate synonym and antonym pairs as the lexical constraints. The ***ATTRACT-REPEL*** model is selected as the specialization model.

We evaluate the experiment results on several gold standard data set, such as ***SimLex-999*** , the ***WordSim-353*** and ***Test Set of SemEval-2014 Task 3***, under the word similarity and the cross-level semantic similarity strategies. Although not all of the evaluation results on different standard data set make improvements, as we are more valued the data set ***SimLex-999*** and the experiment result on it outperforms over the other specialization models, our work is valuable.

We highlight a combination of the previously excellent works to achieve higher performance on the standard gold evaluation data set ***SimLex-999***, compared to the state-of-the-art sense specialization approaches, such as the ***DECONF*** model [33] and the ***Pilehvar and Navigli (2015)*** [34]. However, this is not the end of the work, and improvements and extensions discussed in the **Section 6.1** could be implemented later to stretch the capacity of our approach.

We also want to compare our evaluation result with the ***DECONF*** model and the ***Pilehvar and Navigli (2015)*** over the ***Test Set of SemEval-2014 Task 3***, but unfortunately we cannot fetch the same amount of the word sense pair as them.

We release our code of experiment and evaluation at `https://github.com/EffyLuLifei/Sense_Representations_Generation`, and the whole experiment can be re-implemented following the README.

Bibliography

[1] Eneko Agirre and Aitor Soroa. Semeval-2007 task 02: evaluating word sense induction and discrimination systems. In *Proceedings of the 4th International Workshop on Semantic Evaluations - SemEval '07*, pages 7–12, Prague, Czech Republic, 2007. Association for Computational Linguistics.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A Neural Probabilistic Language Model. page 19.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. *arXiv:1607.04606 [cs]*, July 2016. arXiv: 1607.04606.

[4] E. Bruni, N. K. Tran, and M. Baroni. Multimodal Distributional Semantics. *Journal of Artificial Intelligence Research*, 49:1–47, January 2014.

[5] Jose Camacho-Collados and Mohammad Taher Pilehvar. From Word To Sense Embeddings: A Survey on Vector Representations of Meaning. *Journal of Artificial Intelligence Research*, 63:743–788, December 2018.

[6] Devendra Singh Chaplot and Ruslan Salakhutdinov. Knowledge-based Word Sense Disambiguation using Topic Models. page 8.

[7] Manaal Faruqui, Jesse Dodge, Sujay Kumar Jauhar, Chris Dyer, Eduard Hovy, and Noah A. Smith. Retrofitting Word Vectors to Semantic Lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1606–1615, Denver, Colorado, 2015. Association for Computational Linguistics.

[8] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing Search in Context: The Concept Revisited. page 9.

[9] Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. SimVerb-3500: A Large-Scale Evaluation Set of Verb Similarity. *arXiv:1608.00869 [cs]*, August 2016. arXiv: 1608.00869.

[10] Goran Glavaš and Ivan Vulić. Explicit Retrofitting of Distributional Word Vectors. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 34–45, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[11] Matthew Henderson, Blaise Thomson, and Steve Young. Robust dialog state tracking using delexicalised recurrent neural networks and unsupervised adaptation. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 360–365, South Lake Tahoe, NV, USA, December 2014. IEEE.

[12] Matthew Henderson, Blaise Thomson, and Steve Young. Word-Based Dialog State Tracking with Recurrent Neural Networks. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 292–299, Philadelphia, PA, U.S.A., 2014. Association for Computational Linguistics.

[13] Felix Hill, Roi Reichart, and Anna Korhonen. SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation. *Computational Linguistics*, 41(4):665–695, December 2015.

[14] Thomas Hofmann. Unsupervised Learning by Probabilistic Latent Semantic Analysis. page 20.

[15] Michael P. Jones and James H. Martin. Contextual spelling correction using latent semantic analysis. In *Proceedings of the fifth conference on Applied natural language processing* -, pages 166–173, Washington, DC, 1997. Association for Computational Linguistics.

[16] David Jurgens, Mohammad Taher Pilehvar, and Roberto Navigli. SemEval-2014 Task 3: Cross-Level Semantic Similarity. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 17–26, Dublin, Ireland, 2014. Association for Computational Linguistics.

[17] Alberto H F Laender and Berthier A Ribeiro-Neto. A Brief Survey of Web Data Extraction Toolst. *SIGMOD Record*, 31(2):10, 2002.

[18] Siwei Lai, Kang Liu, Shizhu He, and Jun Zhao. How to Generate a Good Word Embedding. *IEEE Intelligent Systems*, 31(6):5–14, November 2016.

[19] D.L. Lee, Huei Chuang, and K. Seamons. Document ranking and the vector-space model. *IEEE Software*, 14(2):67–75, April 1997.

[20] Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, June 1996.

[21] Fuli Luo, Tianyu Liu, Qiaolin Xia, Baobao Chang, and Zhifang Sui. Incorporating Glosses into Neural Word Sense Disambiguation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2473–2482, Melbourne, Australia, 2018. Association for Computational Linguistics.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*, January 2013. arXiv: 1301.3781.

[23] 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. Curran Associates, Inc., 2013.

[24] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. page 6.

[25] Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Multi-domain Dialog State Tracking using Recurrent Neural Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 794–799, Beijing, China, 2015. Association for Computational Linguistics.

[26] Nikola Mrkšić, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. Neural Belief Tracker: Data-Driven Dialogue State Tracking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1777–1788, Vancouver, Canada, 2017. Association for Computational Linguistics.

[27] Nikola Mrkšić, Ivan Vulić, Diarmuid Ó Séaghdha, Ira Leviant, Roi Reichart, Milica Gašić, Anna Korhonen, and Steve Young. Semantic Specialization of Distributional Word Vector Spaces using Monolingual and Cross-Lingual

Constraints. *Transactions of the Association for Computational Linguistics*, 5:309–324, 2017.

[28] Roberto Navigli. Word sense disambiguation: A survey. *ACM Computing Surveys*, 41(2):1–69, February 2009.

[29] Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. Efficient Non-parametric Estimation of Multiple Embeddings per Word in Vector Space. *arXiv:1504.06654 [cs, stat]*, April 2015. arXiv: 1504.06654.

[30] Kim Anh Nguyen, Sabine Schulte im Walde, and Ngoc Thang Vu. Integrating Distributional Lexical Contrast into Word Embeddings for Antonym-Synonym Distinction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 454–459, Berlin, Germany, 2016. Association for Computational Linguistics.

[31] Masataka Ono, Makoto Miwa, and Yutaka Sasaki. Word Embedding-based Antonym Detection using Thesauri and Distributional Information. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 984–989, Denver, Colorado, 2015. Association for Computational Linguistics.

[32] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics.

[33] Mohammad Taher Pilehvar and Nigel Collier. De-Conflated Semantic Representations. *arXiv:1608.01961 [cs]*, August 2016. arXiv: 1608.01961.

[34] Mohammad Taher Pilehvar and Roberto Navigli. From senses to texts: An all-in-one graph-based approach for measuring semantic similarity. *Artificial Intelligence*, 228:95–128, November 2015.

[35] Alok Ranjan Pal and Diganta Saha. Word Sense Disambiguation: A Survey. *International Journal of Control Theory and Computer Modeling*, 5(3):1–16, July 2015.

[36] Joseph Reisinger and Raymond J Mooney. Multi-Prototype Vector-Space Models of Word Meaning. page 9.

[37] Herbert Rubenstein and John B. Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, October 1965.

[38] David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams. Learning representations by back-propagating errors. page 4, 1986.

[39] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, November 1975.

[40] Gerard Salton. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983. Google-Books-ID: u980AAAACAAJ.

[41] Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nădejde. Nematus: a Toolkit for Neural Machine Translation. *arXiv:1703.04357 [cs]*, March 2017. arXiv: 1703.04357.

[42] Richard Socher, Christopher D Manning, and Andrew Y Ng. Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks. page 9.

[43] Dagobert Soergel. WordNet. An electronic lexical database. page 8.

[44] Julien Tissier, Christopher Gravier, and Amaury Habrard. Dict2vec : Learning Word Embeddings using Lexical Dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263, Copenhagen, Denmark, 2017. Association for Computational Linguistics.

[45] Ivan Vulić, Goran Glavaš, Nikola Mrkšić, and Anna Korhonen. Post-Specialisation: Retrofitting Vectors of Words Unseen in Lexical Resources. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 516–527, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[46] Dongqiang Yang and David M W Powers. Measuring Semantic Similarity in the Taxonomy of WordNet. page 8.

[47] Mo Yu and Mark Dredze. Improving Lexical Embeddings with Semantic Knowledge. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 545–550, Baltimore, Maryland, 2014. Association for Computational Linguistics.

[48] Jingwei Zhang, Jeremy Salwen, Michael Glass, and Alfio Gliozzo. Word Semantic Representations using Bayesian Probabilistic Tensor Factorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1522–1531, Doha, Qatar, 2014. Association for Computational Linguistics.

Acknowledgement

I would first like to thank my thesis advisor Prof. Dr. Goran Glavaš of the Data and Web Science Group at University of Mannheim. The door to Prof. Glavaš office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it. I would also like to thank all the authors of every cited literature. Without their works, my thesis could not have been conducted.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Master-/Bachelorarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Er- klärung rechtliche Folgen haben wird.

Mannheim, den 02.09.2019 Unterschrift