# Language Evolves, so should WordNet - Automatically Extending WordNet with the Senses of Out of Vocabulary Lemmas

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Jonathan Rusert

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Ted Pedersen

April 2017

## Acknowledgements

I would like to acknowledge and support those individuals who helped motivate and gave guidance along the way to completing this thesis.

First, I would like to thank my advisor Dr. Ted Pedersen. Dr. Pedersen always gave timely and useful feedback on both the thesis code, and also thesis text. He was also patient but helped keep me on the track to completing my thesis in a timely manner.

Second, I would like to thank Dr. Pete Willemsen, our director of graduate studies. Like Dr. Pedersen, Dr. Willemsen provided a good base of knowledge in the writing of the thesis. He also helped answer any thesis questions and provided a sublime starting motivation when writing the thesis.

Third, I would like to thank Dr. Bruce Peckham for being on my thesis committee and provided support and feedback for my thesis.

Next, I would like to thank my fellow grad students for their help and support throughout the thesis process. They were helpful in completing the classes required, and also for keeping me sane during this process.

Finally, I would like to thank my good friends and family for their support throughout this process. I would specifically like to thank Ankit, Austin, and Mitch for providing motivational support as friends. Also, I would like to thank my parents, Nathan and Lori Rusert, for their continued support.

Thank you.

Dedication

To my parents, Nathan and Lori Rusert. Thank you for your continued love and support.

Abstract

This thesis provides a solution which finds the optimal location to insert the sense of a word not currently found in lexical database WordNet. Currently WordNet contains common words that are already well established in the English language. However, there are many technical terms and examples of jargon that suddenly become popular, and new slang expressions and idioms that arise. WordNet will only stay viable to the degree to which it can incorporate such terminology in an automatic and reliable fashion. To solve this problem we have developed an approach which measures the relatedness of the definition of a novel sense with all of the definitions of all of senses with the same part of speech in WordNet. These measurements were done using a variety of measures, including Extended Gloss Overlaps, Gloss Vectors, and Word2Vec. After identifying the most related definition to the novel sense, we determine if this sense should be merged as a synonym or attached as a hyponym to an existing sense. Our method participated in a shared task on Semantic Taxonomy Enhancement conducted as a part of SemeEval-2016 are fared much better than a random baseline and was comparable to various other participating systems. This approach is not only effective it represents a departure from existing techniques and thereby expands the range of possible solutions to this problem.

# Contents

# List of Tables

# List of Figures

# 1 Introduction

Human language is forever evolving. Fifteen years ago a word like "selfie" was non existent, but is now a commonly used word in the English language. It is impossible to imagine what new words will become common in the next 15 years. As words become commonly used, they tend to be integrated into our dictionaries. Our ever evolving language make dictionaries a valuable part of our society, since new words are not introduced to every person at once. Dictionaries allow humans to look up words, old or new, with which we are not familiar. They also help humans translate between languages.

WordNet[1] is an online dictionary developed at Princeton University. WordNet stores a collection of 155,287 words (or lemmas) along with their definitions (or glosses) organized into 177,659 sets of synonyms (or synsets). WordNet also includes other information including hypernyms and hyponyms. Hypernyms represent an "is a" relationship between words. A *husky* "is a" *dog*, therefore *dog* is the hypernym of *husky*. Hyponyms are the converse of hypernyms, e.g. *husky* is a hyponym of *dog*. WordNet's inclusion of this extra relational information makes it a useful research tool in Natural Language Processing (NLP). In order for dictionaries to continue being useful, they will have to adapt to and add the new vocabulary added to a language. Unfortunately, WordNet has not been updated since December 2006 and no new updates are planned. This thesis aims to add the ability to continuously add

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

new lemmas/senses into WordNet[2].

We hypothesize that lemmas can be inserted based on a combination of the gloss of the lemma and glosses of lemmas already in WordNet. The intellectual merits of this thesis include the discovery of the best algorithms for finding where a lemma should be inserted into WordNet. This thesis also develops an algorithm which determines if a new lemma should be attached as a new hypernym to a word or if it is similar enough to simply merge the new lemma into the existing set of synonyms.

The success of this thesis' hypothesis will have broader impacts as well. If WordNet cannot keep up with new language, it will become deprecated. This could have serious consequences on the NLP research community as a whole. This thesis will help WordNet keep up by implementing an algorithm which will have the ability to insert new lemmas into any user's local WordNet. This will allow researchers to add in terms specific to their research fields, allowing them to continue to use WordNet.

[2]The particulars of WordNet database files and implementation of insertion can be found in Appendix A

# 2 Background

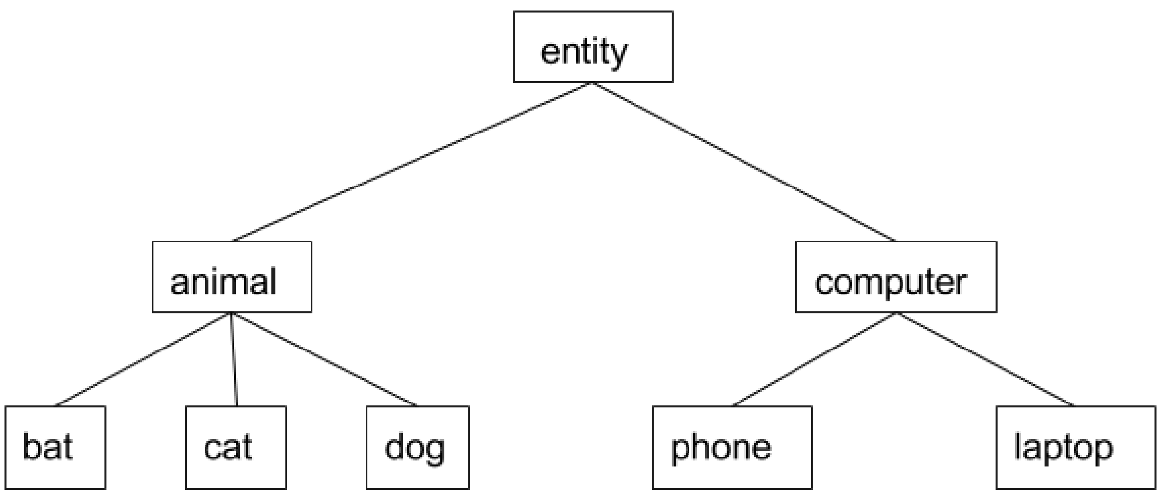
When designing an algorithm for determining the optimal location of an unknown word in WordNet, many factors come up. These include questions such as, "If I determine I want to attach a new lemma to the word *mouse*, how do I know which sense to attach it to?", and also "How similar are the new lemma or words in the new lemma's gloss, and the word I am looking to attach to?" These questions relate to two different areas in Natural Language Processing, Word Sense Disambiguation and Semantic Relatedness.

## 2.1 WSD and Semantic Relatedness

Word Sense Disambiguation (WSD), is one of the core research topics in Natural Language Processing. WSD seeks to answer the question of how to determine what sense of a word is being used. For example, determining what *bat* refers to in the following context: "The bat flew...". *Bat* could refer to both the animal flying, or it could refer to a baseball bat being thrown. The meaning becomes a little more clear once we add more context: "The bat flew out of the batter's hands." With more context, we can now better guess which sense of *bat* is being used (baseball bat).

Semantic Relatedness is connected to WSD. Semantic Relatedness refers to determining how two (or more) terms are related, and also measuring how closely they are related. For example, a *bat* ISA (is a) *animal*, showing that *bat* has an ISA relationship with *animal*. Also, *bat* (the animal) would be more related to *cat* than *phone*, since the first two are both animals. Three important terms that often appear when talking about semantic relatedness are *hypernym*, *hyponym*, and *gloss*. *Hypernym* refers to any ISA relationship, as demonstrated above. *Hyponym*, refers to receiving end of the hypernym. *Hypernym* can also be thought of a more general example while *hyponym* is a more specific example or is a kind of. *Animal* is the hypernym of *bat*, while *bat* is the hyponym of *animal*. Unlike *hypernym* and *hyponym*, *gloss* is not a relation. Instead, *gloss* simply refers to the definition of the sense. One gloss of *bat* would be, "Small rodent-like animal with wings." The *gloss* is useful for differentiating between different senses of the same lemma.

WordNet[1] is an online system that NLP researchers, from Princeton, have created to help solve WSD, semantic relatedness, and many other problems. WordNet contains hierarchies of words and their multiple glosses . WordNet sets up the words in hypernym and hyponym hierarchies. A small (generalized) example of WordNet looks as follows:




Semantic relatedness or more specifically similarity can be determined through WordNet by measuring how far terms are from each other in the WordNet hierarchy.

[1] WordNet can be found:https://wordnet.princeton.edu/

For example in the above figure, *bat* would be more similar to *cat* than *phone* since *bat* is only two steps away from *cat* (*bat*→*animal*→*cat*). It can be seen that this method alone would not be sufficient in measuring similarity, since *bat* and *entity* would have the same similarity value as *bat* and *cat*. Other measures of similarity are introduced in Previous Work, including Extended Gloss Overlaps (2.2.2), and Gloss Vectors (2.2.3).

## 2.2 Previous Work

Since the goal was to find the ideal location of a new lemma only based on the lemma's gloss and glosses in WordNet, it could be seen that finding which sense of a word is being used and how related the out of vocabulary (OOV) lemma and sense locations are would help achieve this goal. It followed that Word Sense Disambiguation and Semantic Relatedness would be featured heavily in our approach. This made it clear that systems and algorithms in these areas would make good assets. The following systems were explored in order to try and form a basis to start our approach. The systems take different approaches to both WSD and Semantic Relatedness, which helps expand the breadth of the basis.

### 2.2.1 Lesk's Algorithm

Michael Lesk first presented a solution to Word Sense Disambiguation when describing what is now commonly referred to as the Lesk Algorithm or Gloss Overlaps [8]. Lesk aimed to tackle WSD by taking the definitions of the senses of a lemma and seeing if words overlap onto definitions of the senses of a different lemma in the same close context.

To demonstrate how this algorithm worked let us revisit our *bat* example first shown in the background section. To determine what sense of *bat* is being used in the sentence "The bat flew out of the batter's hand.", Lesk's algorithm works as follows:

1. Retrieve the glosses for all senses of bat.

- **bat** (nocturnal mouselike mammal with forelimbs modified to form membranous wings and anatomical adaptations for echolocation by which they navigate)
- **bat** (a club used for **swinging** at a ball in various games, as in **baseball**)

2. Retrieve the glosses for other words in the sentence. (Note: To simplify the example, only the first sense of each word is shown.)

- **batter** ((**baseball**) a player who **swings** a **bat** or whose turn it is to **bat**, as in **baseball** or cricket.)
- **hand** (the (prehensile) extremity of the superior limb)
- **flew** (travel through the air; be airborne)

3. Note how many words overlap in each combination of glosses. When comparing the second gloss of bat to the first gloss of batter we see that *bat*, *swing*, *baseball*, and *bat* again all overlap.

4. Presume the sense with the highest overlap value is the correct sense. (In this case the last sense.)

To show the effectiveness of the algorithm, Lesk noted that the program performed in the 50-70% accuracy range when used on short examples from *Pride and Prejudice* and an Associated Press article. However, Lesk acknowledged a problem in his algorithm. The problem arises because glosses can be short and often times might not overlap. Also, determining the gloss source can affect the performance. While words may overlap successfully in the glosses from Webster's Dictionary, they might not overlap from WordNet or vice versa.

Lesk closes his introduction to his algorithm with many questions:

1. What machine readable dictionaries should be used for the glosses?

2. Should examples of the word be used in the algorithm?

3. How should the overlaps be scored compared to each other?

Lesk did not answer these questions outright but believed they offer an exciting future for the possibility of his work. Later work also tried to address these questions.

### 2.2.2 Extended Gloss Overlaps

While Lesk's algorithm was focused on WSD, Extended Gloss Overlaps [1] (EGO) aim to measure semantic relatedness. EGO's function by taking the Lesk Algorithm and incorporating WordNet into the calculation of overlaps by adding in the hypernyms and hyponyms of the lemma and their glosses. The algorithm not only compares the definitions of the two terms, but also the definitions of their respective hypernyms and also hyponyms. The resulting algorithm to calculate the relatedness is as follows:

$$\begin{aligned} relatedness(A,B) = {} & score(gloss(A), gloss(B)) + score(hype(A), hype(B)) \\ & + score(hypo(A), hypo(B)) + score(hype(A), gloss(B)) \\ & + score(gloss(A), hype(B)) \end{aligned} \tag{2.1}$$

$$score(C, D) = numberOfOverlappedWords(C, D) \tag{2.2}$$

Where gloss(), hype(), and hypo() correspond to the gloss of a lemma, the gloss of the hypernym of a lemma, and the glosses of the hyponyms of the lemma respectively.

EGO would handle our *bat* example similarly to the Lesk Algorithm when calculating the relatedness between *bat* and *cat*, except it would add in the glosses of the hypernyms and hyponyms of *bat* and *cat*:

1. Retrieve gloss for *bat* and *cat*.

- **bat** (nocturnal mouselike mammal with forelimbs modified to form membranous wings and anatomical adaptations for echolocation by which they navigate)
- **bat** (a club used for **swinging** at a ball in various games, as in **baseball**)

- **cat** feline mammal usually having thick soft fur and no ability to roar: domestic cats; wildcats
- **cat** a large tracked vehicle that is propelled by two endless metal belts; frequently used for moving earth in construction and farm work

2. Retrieve bat's hypernyms' and hyponyms' glosses. (Only one shown for each as example)

   Hypernym:

- **placental mammal** mammals having a placenta; all mammals except monotremes and marsupials

   Hyponym:

- **fruit bat** large Old World bat of warm and tropical regions that feeds on fruit

3. Retrieve cat's hypernyms' and hyponyms' glosses. (Only one shown for each as example)

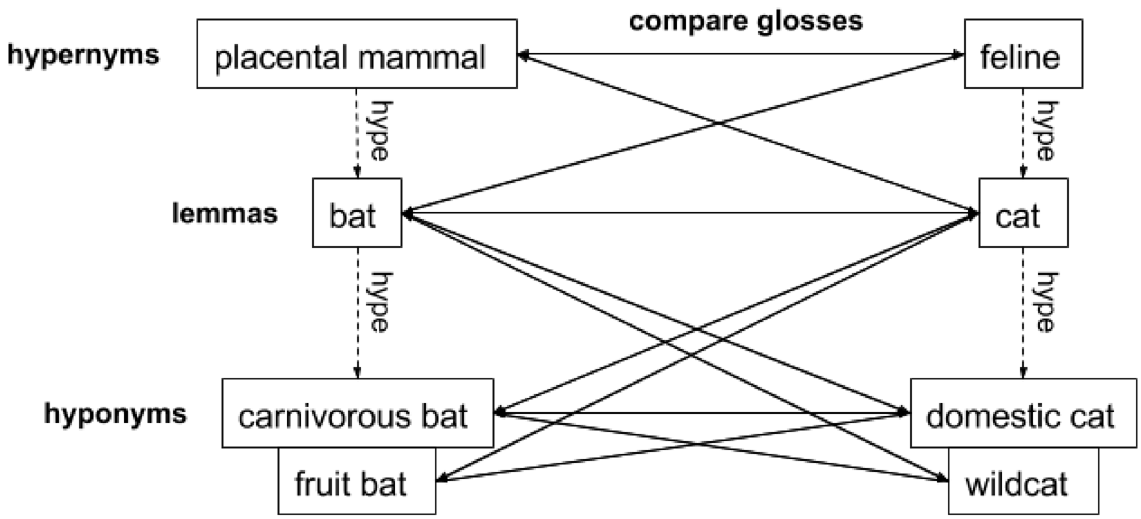
   Hypernym:

   - **feline** any of various lithe-bodied roundheaded fissiped mammals, many with retractile claws

   Hyponym:

   - **wildcat** any small or medium-sized cat resembling the domestic cat and living in the wild

4. Compare relatedness of hypernym glosses, hyponym glosses, and standard glosses. Note: When comparing *bat*'s and *cat*'s hypernyms we observe that *mammal* overlaps. *mammal* also overlaps multiple times when comparing *bat*'s hypernym's gloss with *cat*'s gloss, as well as when comparing *cat*'s hypernym's gloss with *bat*'s gloss.




5. Compute final relatedness with formula 2.1. The structure of all the glosses compared can be seen below.

To evaluate EGO with respect to Semantic Relatedness, the method was compared against human judgment. A 65 word pair set and also a 30 word pair subset were used for the evaluation. The 65 word set was first shown in a study from Rubenstein and Goodenough [14], while the 30 word pair set first appeared in a study by Miller and Charles [5]. For both sets, human subjects were tested on the words by asking them how similar a pair of words were on a 0.0 - 4.0 scale. To find the closeness of the pairs, Spearmans Correlation Coefficient [16] was used. This ranks on values between -1 (opposite ranking) and 1 (exactly the same). When evaluated on the 30 word pair set, EGO has a correlation of .67 to the Miller and Charles study and when evaluated on the 65 word pair set, a correlation of .60 to the Rubenstein and Goodenough study. Banerjee and Pedersen note that EGO correlates well to human judgment.

EGO was also applied to Word Sense Disambiguation (WSD). The formulated algorithm for solving WSD is as follows [11]:

1. Identify a window of context around a target word. Imagine we have the sen-

tence "The player threw the bat into the dugout." and we are trying to determine which sense of *bat* is used. If a window (of surrounding words) of three is chosen, then we examine the words *player* and *dugout*.

2. Assign a score to each sense of the target word by adding together EGO relatedness scores calculated by comparing the sense of the word in question to the senses of each context word. In our example, each sense of *bat* would be scored by EGO relatedness in comparison between *bat* and every sense of *player* and *dugout*. The score between both the animal *bat* (bat#n#1) and the baseball *bat* (bat#n#4) and the senses of *player* and *dugout* can be seen in figure 2.1. Note that "lemma#pos#senseNum" is how WordNet represents lemmas and their senses. In our example, "bat#n#1" shows the first sense of the noun *bat*.

3. The sense with the highest score is chosen as the sense of the target word. Words like *dugout* score higher in relatedness to the baseball *bat* sense and help push the score of *bat#n#4* over *bat#n#1*, which means the baseball sense would be chosen.

EGO was also evaluated on WSD. The data used for evaluation is a 73 word subset from the data of Senseval-2 [3]. EGO was evaluated using three different measures: Precision, Recall, and F-Measure. Precision was calculated by dividing the number of correct answers by the number of answers reported. Recall is calculated by dividing the number of correct answers by number of instances. F-Measure is a measure of the test's accuracy and is calculated by the following algorithm:

$$\text{F-measure} = 2 \times (precision \times recall)/(precision + recall) \tag{2.3}$$

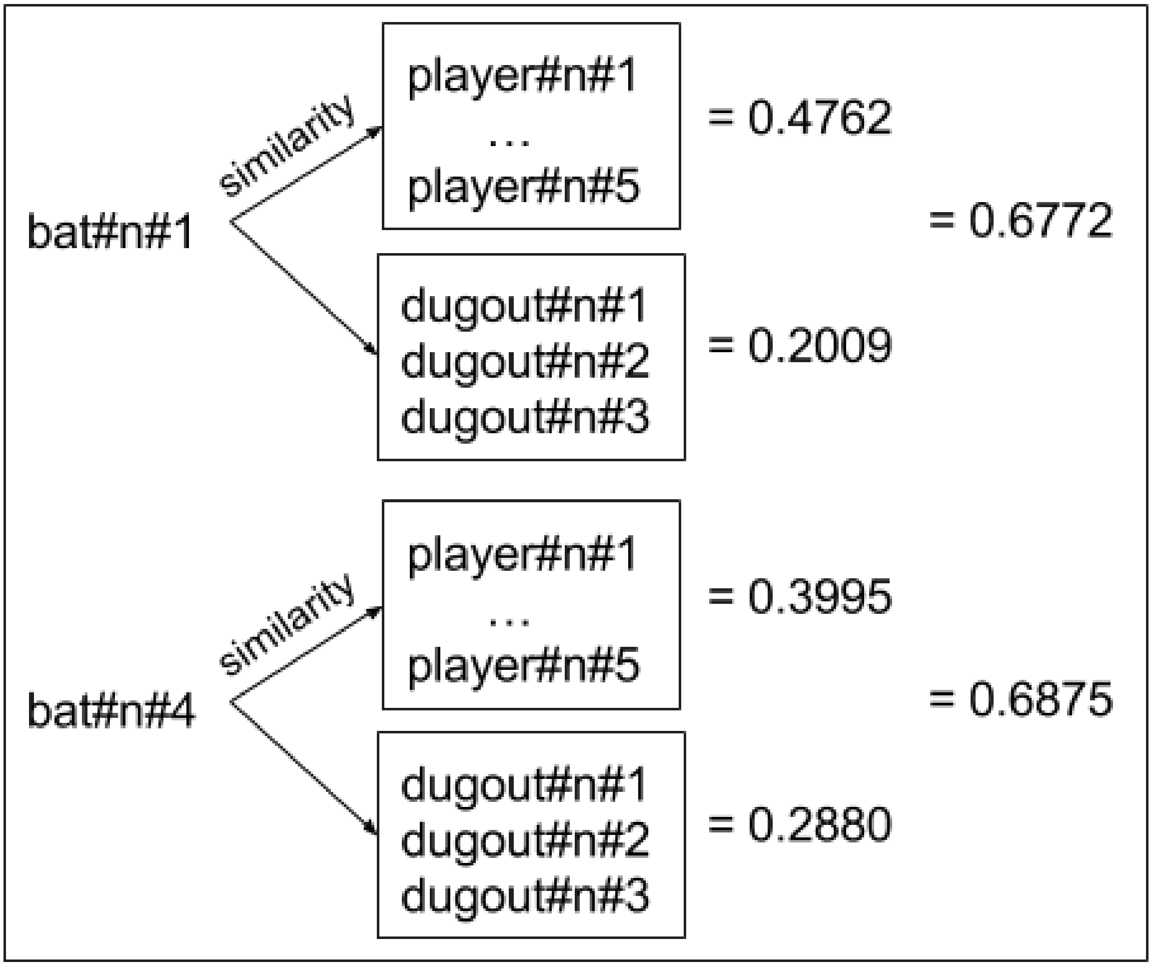



Figure 2.1: Similarity scores between respective *bats*.

Overall EGO scored 0.351 in Precision, 0.342 in Recall, and 0.346 in F-Measure. The original Lesk algorithm, at the same task, scored significantly less 0.183 in each category.

### 2.2.3 Gloss Vectors

Patwardhan and Pedersen [12] aimed at contributing to this area by researching how context vectors combined with WordNet could be used to measure Semantic Relatedness. Normal context vectors [15] function by measuring how two words are related by looking at the words (or context) around those words. By looking at how many times certain words occur with each other in each context, context vectors can weight how closely two terms are related. Gloss vectors address the issue that EGO and the Lesk algorithm had in getting rid of the dependence on glosses and glosses overlapping to be successful. For example, imagine we are trying to find the relatedness between *beer* and *vodka*. In WordNet, *beer*'s gloss is "a general name for alcoholic beverages made by fermenting a cereal (or mixture of cereals) flavored with hops", while *vodka*'s is "unaged colorless liquor originating in Russia". We can observe, that no words in either *beer* or *vodka* overlap, even though we know they would appear in the same contexts.

WordNet-based context vectors (referred to as gloss vectors) implement normal context vectors by using WordNet to obtain the glosses of the terms being compared, then using those glosses to create context vectors. This helps create a more relevant context with which to gain a more accurate context vector, since the glosses are added into the calculations, increasing the overall relevant word count. Gloss vectors are around 50,000 dimensions and are created by finding co-occurrences in the WordNet glosses. To reduce dimensionality, and include more relevant terms, only those words that occur at least five times appear in the 50,000 vector. To calculate gloss vectors, Word Spaces are created as outlined in the gloss vector paper:

1. Initialize the first order context vector to a zero vector v. For example, if we are looking for the word *cat*, *cat*'s vector models:

```
    (bird cave  eat  attack  wings ring  meow screech)
cat (0     0     0      0      0    0     0       0)
```

2. Find every occurrence of v in the given corpus. Continuing with our example,

find all the places where *cat* occurs.

3. For each occurrence of v, increment those dimensions of v, that correspond to the words from the Word Space and are present within a given number of positions around v in the corpus. Again in our example, for each word that occurs around *cat*, add one to that word to increase its occurrences with *cat*. If *meow* occurs in a context with *cat*, we would increment *meow* in *cat*'s vector resulting in:

```
     (bird cave  eat  attack  wings ring  meow screech)
cat  (0     0     0     0      0     0     1       0)
```

   This processes is repeated throughout the whole corpus until we arrive at a resulting vector of cat:

```
     (bird cave  eat  attack  wings ring  meow screech)
cat  (2     0     4     5      1     0     7       3)
```

Gloss vectors tackle our *bat* and *cat* example by use of context vectors. Gloss vectors take a context of *bat* and *cat* and follow the aforementioned steps, creating vectors for *bat* and *cat* and incrementing the dimensions of v for each. Suppose we arrive at the following three vectors:

```
     (bird cave  eat  attack  wings ring  meow screech)
cat  (2     0     4     5      1     0     7       3)
bat  (2     4     5     4      5     0     0       4)
phone(0     0     0     2      0     9     0       0)
```

We can observe that *bat* and *cat*'s vectors are more similar than *cat* and *phone*'s.

As seen in figure 2.2, the *cat* vector is closer to the *bat* vector versus *phone* since *cat* is more like *bat*.

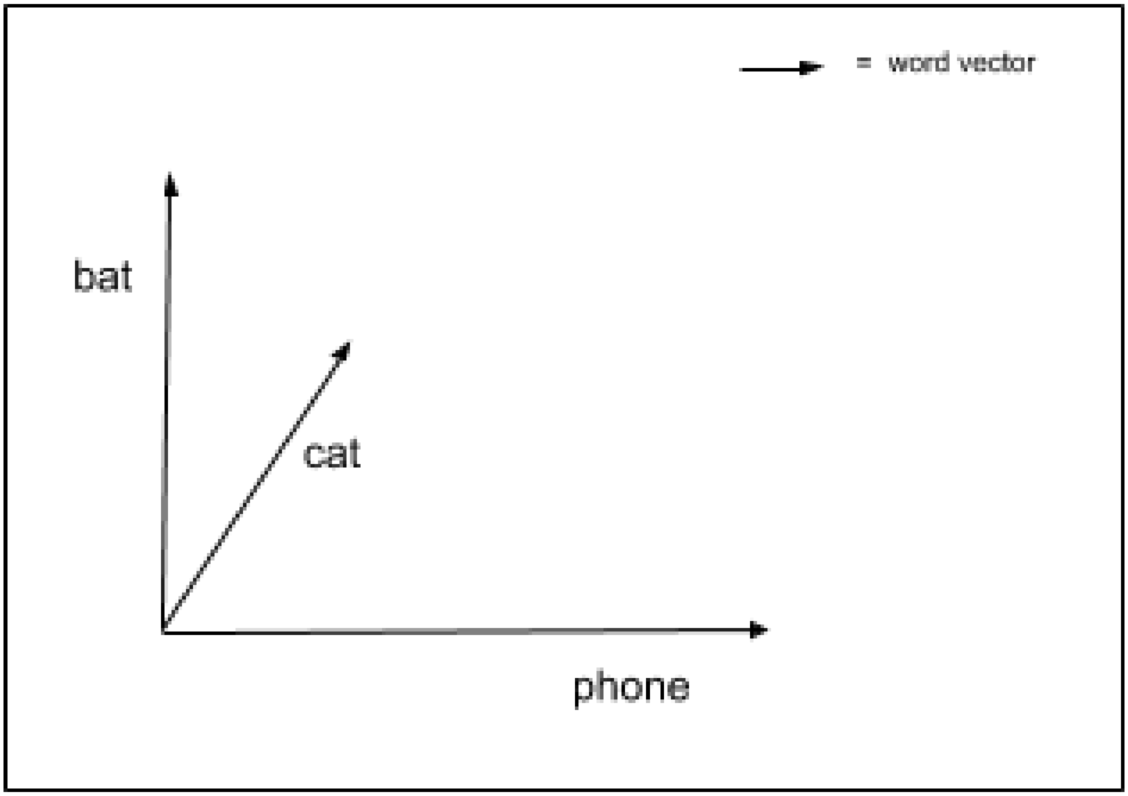



Figure 2.2: *cat*, *bat*, and *phone* represented in a simple vector space

Gloss vectors also include the ability to filter out certain words from appearing in the vectors. Either a stop list can be used or frequency cuttoffs to filter words. Stop lists are lists of common words (a—an—the—...) which many glosses contain and therefore are not overall helpful in determining relatedness. Frequency cuttoffs function similarly to stop lists. However, instead of specifying specific words, frequency cutoffs specify a number which when set as a threshold disqualifies a word from being considered. For example, if the frequency cutoff is 15 and *the* appears 25 times in the contexts, *the* is essentially stop listed. Although gloss vectors include these functions, our systems do not utilize them as the filtering of words is taken care of on a higher level.

Gloss vectors were evaluated similarly to EGO with the word pair set from Miller and Charles and the word pair set from Rubenstein and Goodenough. When being compared to other Semantic Relatedness measures tested on the same data set, Gloss vectors performed the highest in relation to human perception tests. Gloss vectors score 0.91 in Miller and Charles and 0.90 in Rubenstein and Goodenough. Extended Gloss Overlaps [1] was the next highest compared measure, scoring in at 0.81 for Miller and Charles and 0.83 for Rubenstein and Goodenough.

Gloss vectors also performed Word Sense Disambiguation (on SENSEVAL-2 data) well (scoring 0.41), only behind Extended Gloss Overlaps by a few points (scoring 0.45). Patwardhan and Pedersen concluded that the gloss vector scored very well in comparison to human judgments.

### 2.2.4 CROWN

While EGO and gloss vector determined semantic relatedness mainly using WordNet, Jurgens and Pilehvar [6] incorporated Wiktionary[2] into the calculation of semantic relatedness to better find the ideal location for a OOV lemma in WordNet. CROWN aimed to add out of vocabulary (OOV) terms to the existing WordNet synsets. With this method, a new synset was created for the OOV term and added via a hyponym relation to the determined WordNet synset. Jurgens and Pilehvar named this extension of WordNet, CROWN (Community enRiched Open WordNet).

To accomplish the addition of the new synset, CROWN used Wiktionary to look up and add the glosses of similar synsets of the term to the calculation of the term's proper synset location. CROWN ran through 2 steps in its calculation, preprocessing and attachment. In preprocessing, CROWN parsed Wiktionary data to obtain the text associated with each Wiktionary gloss. The glosses were then processed using one of the following two methods to identify which hypernym synset candidates might be

[2]https://www.wiktionary.org/

the correct for the OOV term. First, the glosses were paired with Stanford CoreNLP [9] which extracted words conjucted to the head word as candidates. For example, if we had the sentence "The man and woman were at an impasse.", Stanford CoreNLP would extract *man* and *woman* as candidates, since they are joined together by *and*. Second, more possibilities were added by accessing the first hyperlinked term in the gloss. A hyperlinked term means it also has an accessible entry in Wiktionary. In Wiktionary, *dog*'s first gloss is, "A **mammal**, **Canis lupus familiaris**, that has been **domesticated** for thousands of **years**, of highly variable appearance due to **human** breeding." (where the words in bold are hyperlinks). Since *mammal* is is the first hyperlinked word in *dog*'s gloss, it would be accessed for more possibilities by following the hyperlink to the corresponding Wiktionary entry.

In attachment, CROWN used either structural and lexical attachment or gloss-based attachment. Structural and lexical attachment occurred in one of three ways. First, Wiktionary was used to create mutually-synonymous terms, a common hypernym was then estimated from the aforementioned terms by calculating the "most frequent hypernym synset for all the senses of the set's lemmas that are in WordNet" [6]. The OOV term was then attached to the calculated hypernym. Second, Wiktionary glosses were examined to determine whether they matched a specific predetermined pattern or not. The patterns were inferred from glosses in Wiktionary, since some followed regular patterns. The patterns were matched against the *Person* and *Genus*. *Person* patterns started with the phrase "somebody who" while *Genus* patterns started with "Any member of the" and contained a proper noun later. When *Person* was matched, the term was attached to a descendant using lexical attachment. When *Genus* was matched, the term was "attached to the same hypernym as the synsets with a holonymy relation to the genus's synset." A *holonymy* is a relation that represents a part of a whole, *Felis*(cat family) is a holynym of *cat*. As an example, *cat*'s first gloss is "An animal of the family Felidae." This would match the *Genus* relation since it starts with a variation of "Any member of the" and *Felidae* is a proper noun found later on. *True cat* is a synset of *cat*, while *Felis domesticus* is a holonym of *true cat*. Therefore, *cat* would be attached to *Felis domesticus*. The third structural and lexical attachment used was an *Antonymy* heuristic. *Antonymy* tested the OOV term to determine if it had a prefix that could indicate it was an antonym (e.g., "anti"). If the prefix was removed and the remaining term was in WordNet, the OOV term was attached to the remaining term.

The second attachment, gloss-based attachment, was carried out by using the associated senses of the OOV term which were found from Wiktionary. This method generated possible hypernym synsets by taking each sense and ranking them according to their gloss similarities. The OOV term was then attached to the highest scoring hypernym synset generated.

To show how CROWN works, we will return to bat. Imagine that the term *bat* does not yet exist in WordNet. CROWN would handle inserting *bat* with gloss based attachment, as follows:

- **bat** (nocturnal mouselike mammal with forelimbs modified to form membranous wings and anatomical adaptations for echolocation by which they navigate)

1. Create a possible set of hypernym candidates by looking at their gloss. (To save space we will only display *bat*'s hypernym's previously known gloss.)

- **placental mammal** mammals having a placenta; all mammals except monotremes and marsupials

2. *Placental mammal* would be placed in the set of possible candidates since it matches the first round of preprocessing, in that *mammals* is the first word extracted and that exists in *bat*'s gloss.

3. Gloss based attachment is then used as each term in *bat*'s gloss is analyzed and the highest scoring related term is selected as the hypernym. In this case *mammal*'s gloss makes it the ideal hypernym candidate since *mammal* overlaps several times between the glosses, therefore *bat* is attached to *mammal* in WordNet.

Two evaluations were used to determine the effectiveness of CROWN. The first evaluation used CROWN on terms that were already in WordNet and examining whether or not CROWN determined the correct action. To carry out the evaluation, glosses of 36,605 out of 101,863 nouns and 4,668 out of 6,277 verbs that were "monosemous" (only having one meaning) in WordNet and could be found in Wiktionary were used. CROWN scored well at attaching the terms, scoring 95.4% with nouns and 90.2% with verbs.

The second evaluation calculated "the benefit of using CROWN in an example application". This evaluation used 60 OOV lemmas and 38 OOV slang terms to test CROWN. 51 out of 60 OOV lemmas and 26 out of 38 OOV slang terms were contained in CROWN.

### 2.2.5 Google Similarity Distance

While CROWN aimed to improve the coverage of WordNet and strength of semantic relatedness by adding in Wiktionary to WordNet, Cilibrasi and Vitanyi tried to incorporate the internet as a whole by using Google. Their intuition was that the internet was a vast database, already being updated by humans and Google is a way to search that database. They note that humans naturally use similar words together when writing on the internet, which means similar terms should show up together in a Google search. The result of this effort was the Google Similarity Distance (GSD) [2].

The GSD calculated semantic relatedness by using meta-data obtained about the terms in question when searched in the Google search engine. The GSD worked as follows:

1. Google *term1* and record the number of pages associated with *term1*.
2. Google *term2* and record the number of pages associated with *term2*.
3. Google "term1 term2" and record the number of pages.
4. Record the number of indexed Google pages.
5. Plug in the recorded terms to the GSD formula (where x=term1, y=term2):

$$NGD(x, y) = \frac{G(x, y) - min(G(x), G(y))}{max(G(x), G(y))} \tag{2.4}$$

Calculating the relatedness between *bat* and *cat* is simple following the above formula. We would google *bat* and *cat* separately and record the number of associated pages. Then we would google *bat* and *cat* together and record this number. Finally we would plug these numbers into the NGD(bat, cat) formula.

Several evaluations for how the GSD performed were used. One evaluation, comparable to our previous methods used, was comparing how related different texts are from English novelists. The three novelist whose works were used are William Shakespeare, Jonathan Swift, and Oscar Wilde. The result was a constructed tree of the novels and how closely they were related. The results were positive as all Shakespeare's works were close to the same tree node, as was the same with Swift's and Wilde's works.

It should be noted that while the Google Similarity Distance was a different and new approach to calculating Semantic Relatedness, GSD was not incorporated into our system as it has a possible drawback. Since GSD is based on the web, and the web is constantly changing with new news articles, trends, and other evolving content the GSD score could change in a shorter period of time.

### 2.2.6 Word2Vec

Word2Vec was created by a group of researchers at Google led by Tomas Mikolov [10]. While the previous approaches use different existing structured databases (WordNet, Wiktionary, Google), Word2Vec creates its knowledge by taking in large corpora of text and creating vectors to represent words. Word2Vec uses a two-layer neural network to create these vectors, along with one of two methods, Skip gram or CBOW. Skip gram "learns" the word vectors by predicting the context of a given word, or given one word, Skip gram predicts the words around it. CBOW (Continuous Bag of Words), is opposite of Skip gram in that CBOW predicts the word given its context, or given many words, CBOW predicts the word. The size of vectors for Word2Vec are normally determined by a parameter passed in. This means the size could be anywhere from 100 - 1000 or more depending on what the user chooses. These vectors provide a view of what contexts certain words appear in. This can be useful in both WSD and Semantic Relatedness.

In WSD, knowing the context that appears around a word is the best way to find which sense is being used. As a simple example, imagine we have a word vector that appears as:

```
    (cave batter sonar pitcher wings plate stands screech)
bat (0     4       0      3      0     2     1       1)
```

If we are trying to decide between the baseball bat and animal bat, the vector shows that *bat* has appeared with *batter* 4 times, *pitcher* 3 times, and *plate* 2 times. Using this information, it is easy to infer that this is the baseball bat sense.

In Semantic Relatedness, the context can again provide helpful information in determining how similar two words are, since it is likely that similar words would appear in similar contexts. Imagine we add the following word vectors to our data set:

```
    (cave batter sonar pitcher wings plate stands screech)
base(1      3      0      1      0     3     2       0)
home(2      2      0      1      0     2     0       3)
```

We are then able to observe that *bat* has a more similar vector to *base* than *home*. We can then use these vectors, along with some calculations to find an approximate value of relatedness if we would like.

Although both Word2Vec and Gloss Vectors represent words as vectors, the way the vectors are created differ. Gloss Vectors create vectors by using the glosses from words in WordNet, while Word2Vec creates vectors from large contexts of words.

Like Gloss Vectors, Word2Vec also contains the option for stop lists and frequency cuttoffs. However, again the stop listing is handled on a higher level and these functionalities are not used in our approach.

## 2.3 Important Lexical Resources and Concepts

As seen in EGO's, Gloss Vectors, and CROWN, lexical resources are an essential tool in WSD and Semantic Relatedness. Our approach is focused heavily on WordNet, while others found Wiktionary to be a great help. In this section we expand on WordNet and Wiktionary, as well as the SemEval task with which our thesis' data originates.

### 2.3.1 WordNet

As was stated in the introduction, WordNet is an online dictionary developed at Princeton University[3]. WordNet was created in 1985 and the idea was formulated by George Miller, a psychologist professor. Miller wanted to create a lexical database which modeled the way humans group words, thus WordNet was born. WordNet stores a collection of 155,287 words organized into 177,659 synsets. WordNet also includes other information including hypernyms and hyponyms. The structure of WordNet, with respect to hypernyms and hyponyms, can be visualized in figure 2.3. This simple representation of WordNet can show the connection between the different hypernyms *placental mammal*, *club* and their respective hyponyms *bat#n#1* and *bat#n#5*. WordNet also includes compound words such as "father-in-law". WordNet's large amount of words and, more importantly, relationships, make it an invaluable asset to many NLP systems. WordNet's valuableness can be seen in the sheer amount of citations WordNet has, over 30,000 different papers alone (at the time of this thesis).

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

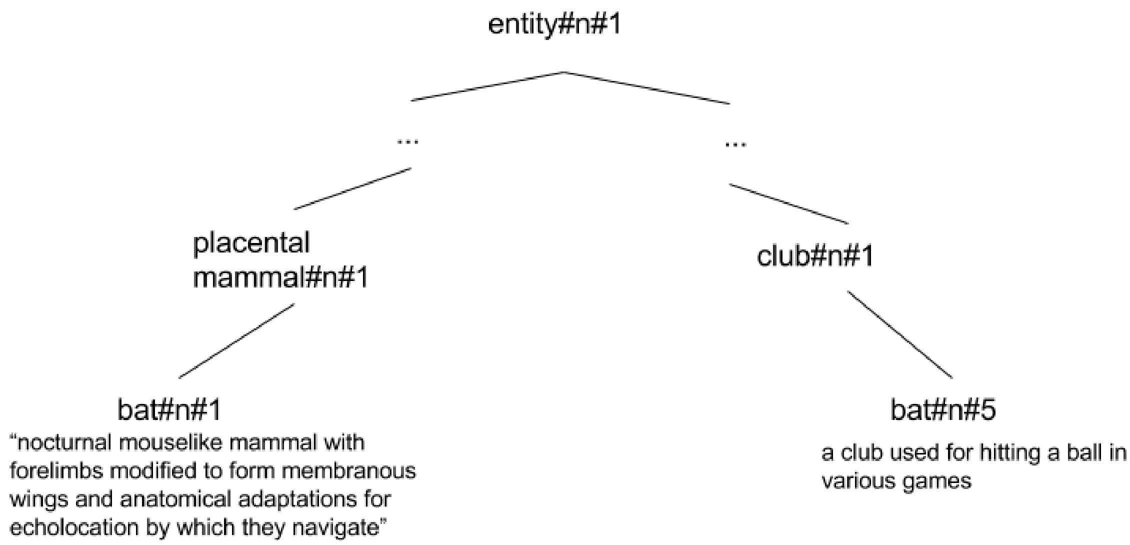


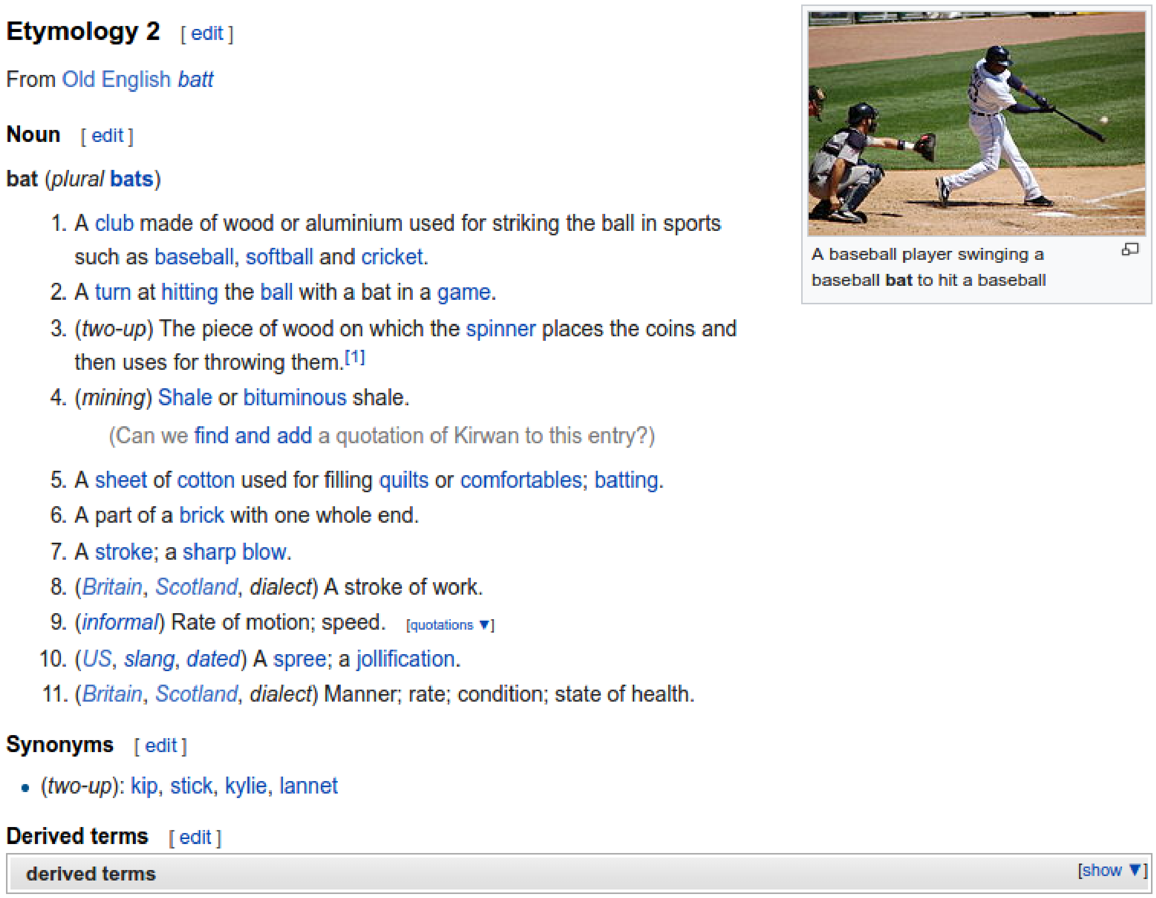

Figure 2.3: Simple representation of WordNet.

### 2.3.2 Wiktionary

Wiktionary[4] is a collaborative project aimed at producing a "free-content multilingual dictionary". Wiktionary contains 5,101,420 entries in 3,200 different languages. Like WordNet, Wiktionary offers more than just the gloss of a word. Wiktionary offers additional information such as etymology, example contexts, derived terms, and more. A sample page of Wiktionary can be seen in figure 2.4.

As can be observed in figure 2.4, Wiktionary also offers hyperlinks to other words within the glosses of the current word being viewed. An example of these hyperlinks being utilized effectively appears in CROWN. As with WordNet, Wiktionary's large database and connections to related words help enrich the data used NLP systems, and subsequently WSD and Semantic Relatedness systems.

[4]https://www.wiktionary.org/


### Etymology 2 [ edit ]

From Old English *batt*

**Noun** [ edit ]

**bat** (*plural* **bats**)

1. A club made of wood or aluminium used for striking the ball in sports such as baseball, softball and cricket.
2. A turn at hitting the ball with a bat in a game.
3. (*two-up*) The piece of wood on which the spinner places the coins and then uses for throwing them.[1]
4. (*mining*) Shale or bituminous shale.
   (Can we find and add a quotation of Kirwan to this entry?)
5. A sheet of cotton used for filling quilts or comfortables; batting.
6. A part of a brick with one whole end.
7. A stroke; a sharp blow.
8. (*Britain*, *Scotland*, *dialect*) A stroke of work.
9. (*informal*) Rate of motion; speed. [quotations ▼]
10. (*US*, *slang*, *dated*) A spree; a jollification.
11. (*Britain*, *Scotland*, *dialect*) Manner; rate; condition; state of health.

A baseball player swinging a baseball **bat** to hit a baseball

**Synonyms** [ edit ]

- (*two-up*): kip, stick, kylie, lannet

**Derived terms** [ edit ]

**derived terms** [show ▼]

Figure 2.4: Wiktionary data for 2nd sense of bat https://en.wiktionary.org/wiki/bat

### 2.3.3 SemEval16 Task14: Semantic Taxonomy Enrichment

SemEval16 Task14 [7] called for a system that could enrich the WordNet taxonomy with new words and their senses. This translates to inserting new lemmas and senses that were previously not in WordNet into their (human perceived) correct place. SemEval16 Task14 was the base starting point for our system.

Task14 also provided the data set on which our system is compared against in Results, Chapter 4. The data set consists of 600 out-of-vocabulary lemmas, along with the part-of-speech, glosses, and Wiktionary references to each lemma. An example from this data is:

```
palliative care noun test.1 A specialized area of healthcare that
focuses on relieving and preventing the suffering of
patients. https://en.wiktionary.org/wiki/palliative_care#English
```

The field *test.1* in the example exists solely for testing purposes. It can be observed that the part-of-speech for each OOV lemma as well, which is used within our approach. Task14 also included an evaluation program which would take in the output from a system and compare that to the gold test key provided along with the test data. This data along with the testing program helped provide a smooth evaluation for the different variations of our system. Along with test data, "trial" and "training" data were included in SemEval16. "Trial" data contains 127 OOV lemmas and "training" data contains 400 OOV lemmas both formatted the same as the test data. SemEval16 Task14 also helped provide ideas which are expanded on in the Implementation (Chapter 3) and Results (Chapter 4) sections.

# 3 Implementation

This chapter aims to explain the multiple algorithms implemented for solving the problem of locating where a word should be inserted in WordNet. We give the general flow of our system and then step through each algorithm in detail.

Our system solves the given problem in three steps:

1. Pre-processing: Acquire all necessary data from WordNet and store it in one step. The data needed includes each word's definitions, hypernyms, hyponyms, and synsets.

2. Location Algorithm: Our system uses one of four algorithms (Overlap, Overlap with Stemming, Similarity, Word2Vec) in this step to determine the location of the OOV lemma.

3. Determining attach or merge: Decide whether or not the new lemma should be attached to the synset of the chosen sense, or merged into it. Attaching should occur when the OOV lemma is the first of its kind, and merging should occur when the OOV lemma is a synonym of the chosen sense.

## 3.1 Pre-processing

As the implementation of our system was underway, it was clear that the system would be making a large number of calls to WordNet. As we accessed more and more data from WordNet[1], our program took longer to finish each time which created a problem for evaluating changes quickly. The pre-processing method was created to speed this process.

Pre-processing consolidates all calls to WordNet at the start of the program, so no duplicate calls need to be made. It does this by first obtaining all nouns and verbs from WordNet and storing them in their respective arrays (one array for nouns and one for verbs). Pre-processing then iterates through each word and retrieves each sense of each word, since the senses are what will determine which synset the new lemma will be merged or attached to later on. The senses are stored in a separate array, which is iterated through, one by one, in the Overlap step to obtain a score for each sense. Next, it iterates through each sense and obtains that senses gloss. Pre-processing cleans each gloss by making all letters into lowercase, removing punctuation, and also removing this list of common stop words *(the—is—at—which—on—a—an—and—or—up—in)* from each gloss. This list of stop words was determined by finding common, less helpful words in the trial/test data. These stop words were found by outputting what words were being overlapped, and these appeared the most frequently even though they rarely added positively to the overlaps scores. Certain stop list words may also have misleading senses in WordNet. *IN* represents inch, indium, and Indiana in WordNet, however, since *in* is a more commonly used word, it could mislead our system by how often it is represented. It then stores the cleaned gloss in a hash that maps the gloss to the corresponding sense. Finally, Pre-processing obtains the hypernyms, hyponyms, and synsets for each sense and stores them in their respective hashes (hypernyms, hyponyms, and synsets).

[1]http://search.cpan.org/dist/WordNet-QueryData/QueryData.pm

## 3.2 Location Algorithms

For the second step, our system uses one of four location determining algorithms: Overlap, Overlap with Stemming, Similarity, Word2Vec.

### 3.2.1 Overlap

The Overlap algorithm aims to find the ideal location for the OOV lemma by using the idea that similar lemmas will have some of the same words in their glosses and therefore there will be overlapping or matching words. Ideas were borrowed from both Lesk[8] and Extended Gloss Overlaps [1] which are covered in Chapter 2, Background.

#### Overlap Implementation

Our Overlap algorithm works by iterating through each sense obtained from WordNet and creating an expanded sense by adding information from each sense. The expanded sense is then compared to the to-be-inserted lemmas creating a score to determine how alike the terms are. It should be noted that only candidates with a corresponding part-of-speech to the OOV lemma were compared as to improve time and not cause nouns to be mapped to verbs and vice versa.

For each sense to be compared, the expanded sense was created. First the sense's gloss was obtained from the hash initialized in pre-processing. Next the sense's immediate hypernyms and their glosses were retrieved and added to the expanded sense. Likewise, the sense's immediate hyponyms and their glosses were retrieved and added to the expanded sense. Finally, the sense's corresponding synset and their corresponding glosses were retrieved and added to the expanded sense. Next before any word overlaps could be processed, the new lemma's gloss needed to be cleaned up.

To provide clarity, we will act as if *ink* (taken from Semeval16 Task14 trial data) is being inserted into WordNet. For reference *ink*'s provided definition was "Tattoo work." The lemma was cleaned up following the same steps as the WordNet glosses followed in the pre-processing step. *Ink*'s definition would now become "tattoo work", since all letters are made lowercase. However, since *ink* did not contain any stop words on the list, none were removed.

Now the system steps through each word in the lemma's gloss and checks for overlaps in the glosses of the expanded sense's gloss, each hypernym's gloss, each hyponym's gloss, and finally each synset's gloss. If the word being checked is part of the lemma of each sense, it receives a bonus score. The reasoning behind the bonus score was that defining a lemma with words it belongs too is a common approach; husky's gloss is "breed of heavy-coated Arctic sled dog", where dog would be the target. The bonus score was originally set to (10 * the length of the lemma) but was found to impact the overallscore when the bonus was changed. An experimental exploration of the impact of this bonus can be found in Chapter 4, Results. This bonus was applied to all single words but limited to compound words of at most two words. The decision to limit the length of compound words was arrived at since larger compounds like *Standing-on-top-of-the-world* would score higher than *World-wide* just because they were much longer compounds, even though they occur less often.

Since *ink*'s definition contains the word *tattoo*, any sense with *tattoo* in its lemma will receive the bonus. This means that *tattoo#n#:.* (i.e. any noun sense of *tattoo*) would receive a bonus. The same holds true for *work#n#:.* (i.e. any noun sense of *work*). The overlapping of words were also weighted by the number of characters present in those words (or more simply length of those words), so longer words carried a heavier weight in the score than shorter ones. As with *ink*, when the word *tattoo*, in the definition of ink, overlaps with another compared word it adds 6 to the score since *tattoo* contains 6 letters, whereas *work* would only add 4 to the score.

The final score of the sense was calculated by dividing the number of overlaps by the total length of words from the new term.

$$\begin{aligned} score = (&SenseLaps + HypeLaps \\ &+ HypoLaps + SynsLaps \\ &+ BonusLapsTotal)/GlossLength \end{aligned} \tag{3.1}$$

The sense with the highest score at the end was presumed to be the chosen sense to either attach or merge to in WordNet.

Our system determined that *ink* belonged to *tattoo#n#3* whose definition from WordNet was, "the practice of making a design on the skin by pricking and staining". Since *ink* had a short definition provided from Wiktionary, the largest score came from the fact that *tattoo* gained the bonus score from overlapping with the definition. The correct answer provided in the key was *tattoo#n#2*, the reason for the differences was thought to likely be the fact that our system did not identify present participle words, since *tattoo#n#2* contained the word "tattooing". This miss brought us to our next algorithm Overlap with Stemming.

### 3.2.2 Overlap with Stemming

Our Overlap with Stemming (OwS) algorithm aimed to address an issue above with overlap, that different tenses of words (tattoo vs. tattooing) and different number representations of words (singular vs. plural) would not overlap. Examples of this missed overlapping might be "pony" vs. "ponies" or even more simply "bat" vs. "bats".

### OwS Implementation

The OwS implementation is essentially the same as our Overlap algorithm, with one change in the PreProcessing step. With OwS the PreProcessing step is carried out with a stemming flag on. This causes a stemmer to be used on the glosses of the words in WordNet before being stored. Stemming translates a word into it's base form (or lemma). For example, *tattooing* would become *tattoo* and *ponies* would become *pony* in the cleaned glosses. We used Lingua::Stem[2] as our stemmer. After PreProcessing, OwS follows the Overlap algorithm exactly, except when cleaning the OOV lemma's gloss, it also stems the gloss unlike the Overlap algorithm.

When running OwS on the *ink* example, the system determines correctly that *ink* should be attached to *tattoo#n#2*. This helps solidify the usefulness of adding stemming to our system.

## 3.2.3 Relatedness

The Relatedness algorithm takes a different approach to finding where the OOV lemma should be inserted. Instead of counting the overlapping words in the glosses, Relatedness uses a relatedness measure to find how related two words are. Relatedness aims to do away with relying on words in glosses matching perfectly to be relevant. We used WordNet::Similarity::vector[3] as our relatedness measure as it utilizes WordNet's structure and allows the comparison between different parts-of-speech. WordNet::Similarity::vector is an implementation of gloss vectors discussed in Background (Chapter 2).

[2] http://search.cpan.org/~snowhare/Lingua-Stem-0.84/lib/Lingua/Stem.pod
[3] http://search.cpan.org/~tpederse/WordNet-Similarity-2.07/lib/WordNet/Similarity/vector.pm

### Relatedness Implementation

Like the previous two methods, Relatedness iterates through each sense, of same part-of-speech (POS) as the OOV lemma, in WordNet. After cleaning the OOV lemma's gloss, the gloss is then processed through a POS tagger. We used Lingua::EN::Tagger[4] as our POS tagger. Lingua::EN::Tagger is a probability based tagger, trained on corpus data, which determines parts-of-speech from a look up dictionary and set of probabilities. After the gloss is tagged, each word from the gloss is then passed against WordNet to find all the senses of the word. Each sense of each word in the OOV lemma's gloss is then compared against the current candidate sense with a similarity measure and added to the candidate's final score. As above, the highest scoring candidate was chosen as the insert location.

When running Relatedness on *ink* the system chooses *work#n#1* as the correct placement. This might be surprising, however, the way Relatedness functions is by finding the relatedness between a sense in WordNet and the lemma's gloss. Unlike the WordNet words, the words in the gloss are not specified to be a sense. This means that *tattoo* could be *tattoo#n#1*, *tattoo#n#2*, or *tattoo#n#3*. The system compares to all these to try and determine the correct candidate. While *tattoo* has three senses, *work* has seven senses, which causes a weight towards *work*'s favor.

## 3.2.4 Word2Vec

Our final algorithm was one that used a large amount of training data with a Word2Vec system. Word2Vec aims to utilize the notion that similar words appear in similar contexts. We used the Gensim[5] implementation of Word2Vec trained on the

[4] http://search.cpan.org/~acoburn/Lingua-EN-Tagger-0.28/Tagger.pm

[5] https://radimrehurek.com/gensim/models/word2vec.html

Google-News-Vectors[6]. Google-News-Vectors is data created from vectors trained on about 100 billion words from Google News.

### Word2Vec Implementation

Instead of examining each sense in WordNet one at a time, as the other algorithms do, Word2Vec takes in all words in WordNet with the same POS as the OOV Lemma. These words are then passed to be evaluated via Word2Vec's similarity measure. Word2Vec is first trained on Google-News-Vectors. This training creates vectors for each word in that data which can be used to determine similarity. Note that the vectors created from Word2Vec differ from the gloss vectors in Relatedness, since Word2Vec uses contextual similarity trained on Google-News-Vectors rather than WordNet when creating the vectors. Word2Vec attempts to find the similarity of each WordNet candidate word and the OOV lemma. If the OOV lemma does not exist in the training data, Word2Vec finds the similarity between the candidate word and the OOV lemma's gloss' words instead. The highest scoring, presumably most similar, candidate word from WordNet is chosen as the insert location. After calculating the highest similarity score, that score is paired against a user chosen confidence value (CV). If the score is below the CV threshold, the OOV lemma and its candidate are not included in the output.

Our Word2Vec system determines that for *ink*, *toner#n#1* is the place in WordNet *ink* should be inserted. The reason for this decision demonstrates a flaw in the chosen Word2Vec algorithm. While new words like *selfie*, would only appear in similar contexts to the optimal hypernym, existing words such as *ink* can be overshadowed by their pre-existing senses, and cause the system to choose a pre-existing sense instead.

[6]https://github.com/mmihaltz/word2vec-GoogleNews-vectors

## 3.3 Merge or Attach

Our system determines whether the term should be merged or attached by looking at the frequency of the chosen sense as obtained from the WordNet::QueryData frequency() function. The frequency function returns the frequency count of the sense from WordNet database tagged text. If the frequency was low (if it was equal to zero), then it was assumed to be a rarer sense so the program would attach the new term. If it was higher (greater than zero), then the opposite was assumed and merge was chosen.

*ink* was chosen to be attached to *tattoo#n#2* which means the frequency was equal to zero.

# 4 Experimental Results

This chapter presents experimental results from the implemented systems outlined in Chapter 3, Implementation. The implemented systems were run on the 600 lemma test data set which was provided for SemEval Task 14 [7] and described in Chapter 2, Background. Our implemented systems scored as shown in Table 4.1.

Several measures are used to evaluate the Task 14 systems. These include *Wu & Palmer Similarity* (Wu&P), *Lemma Match* (LM), *Recall*, and *F1*.

*Wu & Palmer Similarity*, as defined by SemEval16 Task 14 task organizers, is calculated by finding the "similarity between the synset locations where the correct integration would be and where the system has placed the synset."[7]

$$\textit{Wu \& Palmer}(s1, s2) = 2 \times \textit{depth}(\textit{lcs}(s1, s2))/(\textit{depth}(s1) + \textit{depth}(s2)) \tag{4.1}$$

LCS stands for least common subsumer, and is the most specific ancestor in WordNet which sense 1 (s1, the correct/gold sense) and sense 2 (s2 the system discovered sense) share. This score is between 0 and 1.

The *Lemma Match*, again defined by the task organizers, is scored by, "the percentage of answers where the [attach or merge] operation is correct and the gold and system-provided synsets share a lemma." For example, if the system submits "dog#n#1 attach" and the gold system has the answer as "dog#n#1 merge", the answer will be scored zero since the attach and merge do not match even though the lemmas do. Likewise, if the system chooses "cat#n#1 merge", even though both are

| System | Wu&P | LM | Recall | F1 |
|---|---|---|---|---|
| UMNDuluth Sys 1 (2 bonus) | 0.3395 | 0.0984 | 0.9983 | 0.5067 |
| UMNDuluth Sys 1 (10 bonus) | 0.3857 | 0.1467 | 1.0000 | 0.5567 |
| UMNDuluth Sys 1 (25 bonus) | 0.3802 | 0.2117 | 1.0000 | 0.5509 |
| UMNDuluth Sys 1 (50 bonus) | 0.3809 | 0.2100 | 1.0000 | 0.5517 |
| UMNDuluth Sys 1 (100 bonus) | 0.3735 | 0.1550 | 1.0000 | 0.5439 |
| UMNDuluth Sys 1 (500 bonus) | 0.3791 | 0.2083 | 1.0000 | 0.5498 |
| OwS (2 bonus) | 0.2824 | 0.0767 | 1.0000 | 0.4404 |
| OwS (10 bonus) | 0.3539 | 0.1667 | 1.0000 | 0.5227 |
| OwS (25 bonus) | 0.3392 | 0.1467 | 1.0000 | 0.5066 |
| OwS (50 bonus) | 0.3667 | 0.1667 | 1.0000 | 0.5366 |
| OwS (100 bonus) | 0.3707 | 0.1700 | 1.0000 | 0.5409 |
| OwS (500 bonus) | 0.3708 | 0.1700 | 1.0000 | 0.5410 |
| Relatedness | 0.3256 | 0.0783 | 1.0000 | 0.4912 |
| Word2vec (0.0 CV) | 0.3574 | 0.0933 | 1.0000 | 0.5266 |
| Word2vec (0.10 CV) | 0.3591 | 0.0941 | 0.9917 | 0.5273 |
| Word2vec (0.25 CV) | 0.3722 | 0.1092 | 0.7633 | 0.5004 |
| Word2vec (0.50 CV) | 0.4229 | 0.1092 | 0.2650 | 0.3258 |
| Word2vec (0.75 CV) | 0.4256 | 0.0952 | 0.0350 | 0.0647 |
| SemEval16 Baseline: First word, first sense | 0.5139 | 0.4150 | 1 | 0.6789 |
| SemEval16 Baseline: Random synset | 0.2269 | 0.0000 | 1.0000 | 0.3699 |
| *Median of Task14 Systems* | | | | 0.5900 |

Table 4.1: System Scores on SemEval16 Task 14 Test Data

chosen as merge, the lemmas do not match which again causes the score to be zero.

*Recall* refers to the percentage of lemmas attempted by the system. If 600 were attempted out of 600, then recall equals one. The F1 score uses the average of all *Wu & Palmer* scores and is calculated as follows:

$$F1 = 2 \times (Wu\ \&\ Palmer \times recall)/(Wu\ \&\ Palmer + recall) \tag{4.2}$$

.

The SemEval16 Task 14 organizers also included two different baseline scores on the data set, *SemEval16 Baseline: First word, first sense* and *SemEval16 Baseline: Random synset*. These baselines are described below.

*SemEval16 Baseline: First word, first sense* starts by finding the first word with the same part of speech as the OOV lemma, in the gloss of the OOV lemma. This found word is chosen as the lemma's target hypernym. *First word, first sense* always chooses the first sense of that target word. For example, if *puppy*'s gloss is "A young dog.", the first noun is *dog*, which will cause *puppy* to be mapped to "dog#n#1". *SemEval16 Baseline: Random synset* simply assigns the lemma's hypernym randomly. As can be seen in Table 4.1, *First word, first sense* scored well which proved to be a hard baseline to overcome. The success of this baseline can most likely be attributed to the notion that the OOV lemma's gloss often contained the hypernym to which it should be attached. This can be seen in many common lemmas' definitions. Imagine we are trying to define what an iPhone is. There is a good chance we would include the fact that it is a cellphone, which is the exact place we'd want to attach it to in WordNet. Another example is defining what a Husky is. Again, we would include that a Husky is a dog, as well as the features that distinguish it.

Our first system *UMNDuluth Sys 1* was based on the concept of extended gloss overlaps[1] described in Background (Chapter 3). This system finds the candidate, for which the OOV lemma should be attached or merged to, by overlapping words in the OOV lemma's gloss with words in each candidate's gloss as well as each candidate's hypernym and hyponyms' glosses. This system participated in SemEval16 Task 14 and placed 12th out of 13. Table 4.2 shows the systems and their respective scores and rankings. As can be seen, even the #1 ranked system for Task 14 was only slightly above the *SemEval16 Baseline: First word, first sense*.

After *UMNDuluth Sys 1* was submitted, we noticed a last minute change in how large of a bonus a candidate sense should receive for appearing in the OOV lemma's gloss. Our submitted system had a bonus of 2, however, our development system had a bonus of 10 (the bonus had been reduced to test other affects of overlaps when testing).

| Rank | Team | System | Wu&P | LM | Recall | F1 |
|---|---|---|---|---|---|---|
| 1 | MSerjrKU | System2 | 0.523 | 0.428 | 0.973 | 0.680 |
| 2 | MSerjrKU | System1 | 0.518 | 0.432 | 0.968 | 0.675 |
| 3 | TALN | test_cfgRun1 | 0.476 | 0.360 | 1.000 | 0.645 |
| 4 | TALN | test_cfgRunPickerHypos | 0.472 | 0.240 | 1.000 | 0.641 |
| 5 | TALN | test_cfgRun2 | 0.464 | 0.353 | 1.000 | 0.634 |
| 6 | VCU | Run3 | 0.432 | 0.161 | 0.997 | 0.602 |
| 7 | VCU | Run2 | 0.419 | 0.171 | 0.997 | 0.590 |
| 8 | VCU | Run1 | 0.408 | 0.124 | 0.997 | 0.579 |
| 9 | Duluth | Duluth2 | 0.347 | 0.043 | 1.000 | 0.515 |
| 10 | JRC | MainRun | 0.347 | 0.066 | 0.987 | 0.513 |
| 11 | Duluth | Duluth3 | 0.345 | 0.017 | 1.000 | 0.513 |
| **12** | **UMNDuluth** | **Run1** | **0.340** | **0.098** | **0.998** | **0.507** |
| 13 | Duluth | Duluth1 | 0.331 | 0.023 | 1.000 | 0.498 |
| Baseline: First word, first sense | | | 0.5139 | 0.415 | 1.000 | 0.6789 |
| Baseline: Random synset | | | 0.2269 | 0 | 1.000 | 0.3699 |

Table 4.2: SemEval16 Task 14 Participating System Scores

When running our submitted system with a bonus of 10, we noted an improvement in the F1 score, which is shown as *UMNDuluth Sys 2*. This improvement led us to run more experiments on the improvement of the F1 score with respect to the bonus. We found that continuing to increase the bonus resulted in a plateau in the F1 score. This can be seen in *UMNDuluth Sys 2-6* in Table 4.1. The improvements in results with increased bonuses most likely has to do with the intuition behind *Baseline: First word, first sense*, which is the OOV lemma's gloss will often contain the hypernym it should be attached or merged to. As the bonus increases this causes the system to pick one of these included words, since the bonus word begins to completely outweigh any non bonus overlap.

Another common NLP technique omitted from our first system (*Sys 1*) was stemming. To stem a word is to remove suffixes and return that word to its base form. *Flying* would be stemmed to *fly*. We believed that adding stemming to our original system would cause more words to overlap which would in turn arrive at a better candidate. We used Lingua::Stem[1] as our stemmer. *OwS* (Overlap with Stemming) shows the scores received when adding stemming to our first system, which have decreased slightly. The decrease in score is likely to do with the low encounter rate of relevant different verb tenses and noun plurals.

After the SemEval16 Task 14 was completed, other participating researchers (JRC in table 4.2), released descriptions of their own approaches. One such approach used relatedness measures, instead of only overlaps, to determine the optimal candidate for each OOV lemma [17]. We believed a relatedness measure may offer a more rounded score for each candidate hypernym as similar words such *dog* and *puppy* would score well and contribute to the score even though they may not have overlaps in their glosses. This inspired us to approach the problem in a similar manner by utilizing WordNet::Similarity [2] [13]. WordNet::Similarity includes tools for measuring how related two words are by factoring in their distance apart in WordNet. We used the co-occurrence vector implementation[3] to find the relatedness between each candidate and each word in the OOV Lemma's gloss, then determined the optimal candidate by choosing the highest scoring. It should be noted that WordNet::Similarity::vector is an implementation of the gloss vectors described in Background (Chapter 2). *Relatedness* shows the results for this system.

Our final idea was to incorporate machine learning into our system. Word2Vec is a system, originally developed at Google [2], which is trained on a large corpus of text. This creates word vector spaces which represent each word as a large vector populated with high counts of words that appear in the same context. We believed Word2Vec's extra knowledge of large contexts could help find which WordNet words OOV lemmas were most similar to. This follows the same idea as *Relatedness* as similar words would

[1] http://search.cpan.org/~snowhare/Lingua-Stem-0.84/lib/Lingua/Stem.pod
[2] http://search.cpan.org/~tpederse/WordNet-Similarity-2.07/lib/WordNet/Similarity.pm
[3] http://search.cpan.org/~tpederse/WordNet-Similarity-2.07/lib/WordNet/Similarity/vector.pm

appear in similar contexts. After the initial implementation of our Word2Vec system, we added a confidence value (CV) which would filter out chosen candidates if their calculated similarity value was not above the CV threshold. Our Word2Vec system is represented as *Word2Vec (x CV)* where x is the set confidence value with range 0 - 1. As can be observed, as the CV increases, the Wu & Palmer score increases, however, the recall decreases. This makes sense, as we are factoring out the words we are less confident about, which decreases the total number tested but increases the score of those tested. While the bonus and CV both have the ability to be adjusted in different systems, the bonus was left out of this initial implementation as the application of the bonus would cause the need for all words to be normalized based on the bonus. This could reduce the effectiveness of any CV above 0.1 or lower. For example, suppose a bonus is set to 10. In order to keep the score below 1, we would need to divide each score by an additional 10, since when a bonus word is found, we want it to both be worth 10 times the amount of non bonus words and remain below 1. If the bonus was too large, even the most confident candidate could fall below a CV of 0.1.

Table 4.3 shows how our systems performed on the individual parts-of-speech (POS). *Total Recall* refers to the amount of that parts-of-speech in the 600 word test data from SemEval16 Task 14. As can be seen, nouns composed most of the test data with 86%, while verbs composed the final 14%. *POS F1* refers to the F1 score for that particular POS. This means that instead of using the *Total Recall* in the F1 score, we use a flat recall of 1.0000 to show the F1 score for each POS. The separation between nouns and verbs reveals some things about the results. Nouns composing most of the test data, means that if a system performs better at one POS over the other, it could be affected by this. The Wu & Palmer for each of our systems, is very similar in each the noun and verb category. However, the Lemma Match (LM) is significantly higher in the verb data compared to the noun data. This may be due to the lower amount of verbs, which would mean getting a few correct would have a larger impact on the LM score compared to nouns. In order to fully test this theory though, we would need to test an equal amount of verb tests.

Regardless of which system was used, all of our systems approached the problem of determining how to merge and attach the same way (by using the frequency amount of a candidate from WordNet). This approach is described in Implementation. Our results for the 600 word test set are seen in the contingency table, Table 4.4. These results show that the data itself is weighted towards attaching an OOV lemma over merging it. Also, our system correctly chose 447/568 attaches, and 7/32 merges, for a total of 454/600 or 75.67% accuracy.

To see how much of an effect the missed attach/merge choices had on our *Lemma Match* score we combined the key answers for attach/merge with our lemma answers. The results of select systems are shown in Table 4.5. *LM System* shows the lemma match calculated with our systems' attach/merge answers, while *LM Key* shows the lemma match calculated with the keys' attach/merge answers. As can be seen, *LM Key* does not score that much higher than *LM System*. This means that the LM score is brought down more by our system not choosing the correct lemma, rather than the system not choosing the correct attach/merge action.

| System | POS | Wu&P | LM | Total Recall | POS F1 |
|---|---|---|---|---|---|
| UMNDuluth Sys 1 (2 bonus) | Noun | 0.3552 | 0.0677 | 0.8617 | 0.5242 |
| | Verb | 0.3509 | 0.3373 | 0.1383 | 0.5195 |
| UMNDuluth Sys 1 (10 bonus) | Noun | 0.3792 | 0.1199 | 0.8617 | 0.5499 |
| | Verb | 0.3545 | 0.3614 | 0.1383 | 0.5234 |
| UMNDuluth Sys 1 (25 bonus) | Noun | 0.3839 | 0.1219 | 0.8617 | 0.5548 |
| | Verb | 0.3474 | 0.3373 | 0.1383 | 0.5157 |
| UMNDuluth Sys 1 (50 bonus) | Noun | 0.3813 | 0.1257 | 0.8617 | 0.5521 |
| | Verb | 0.3552 | 0.3494 | 0.1383 | 0.5242 |
| UMNDuluth Sys 1 (100 bonus) | Noun | 0.3777 | 0.1277 | 0.8617 | 0.5483 |
| | Verb | 0.3614 | 0.3253 | 0.1383 | 0.5309 |
| UMNDuluth Sys 1 (500 bonus) | Noun | 0.3831 | 0.1277 | 0.8617 | 0.5539 |
| | Verb | 0.3626 | 0.3614 | 0.1383 | 0.5322 |
| OwS (2 bonus) | Noun | 0.2815 | 0.0522 | 0.8617 | 0.4393 |
| | Verb | 0.2884 | 0.2289 | 0.1383 | 0.4477 |
| OwS (10 bonus) | Noun | 0.3547 | 0.1354 | 0.8617 | 0.5237 |
| | Verb | 0.3591 | 0.3735 | 0.1383 | 0.5284 |
| OwS (25 bonus) | Noun | 0.3406 | 0.1199 | 0.8617 | 0.5081 |
| | Verb | 0.3306 | 0.3133 | 0.1383 | 0.4969 |
| OwS (50 bonus) | Noun | 0.3683 | 0.1393 | 0.8617 | 0.5383 |
| | Verb | 0.3569 | 0.3373 | 0.1383 | 0.5261 |
| OwS (100 bonus) | Noun | 0.3729 | 0.1431 | 0.8617 | 0.5432 |
| | Verb | 0.3571 | 0.3373 | 0.1383 | 0.5263 |
| OwS (500 bonus) | Noun | 0.3729 | 0.1431 | 0.8617 | 0.5432 |
| | Verb | 0.3579 | 0.3373 | 0.1383 | 0.5271 |
| Relatedness | Noun | 0.3299 | 0.0832 | 0.8617 | 0.4961 |
| | Verb | 0.2986 | 0.0482 | 0.1383 | 0.4599 |
| Word2vec (0.0 CV) | Noun | 0.3581 | 0.0754 | 0.8617 | 0.5274 |
| | Verb | 0.3534 | 0.2048 | 0.1383 | 0.5222 |
| Word2vec (0.10 CV) | Noun | 0.3598 | 0.0760 | 0.8550 | 0.5292 |
| | Verb | 0.3549 | 0.2073 | 0.1367 | 0.5239 |
| Word2vec (0.25 CV) | Noun | 0.3748 | 0.0909 | 0.6417 | 0.5452 |
| | Verb | 0.3588 | 0.2055 | 0.1217 | 0.5281 |
| Word2vec (0.50 CV) | Noun | 0.4203 | 0.0889 | 0.2250 | 0.5918 |
| | Verb | 0.4371 | 0.2917 | 0.0400 | 0.6083 |
| Word2vec (0.75 CV) | Noun | 0.4385 | 0.0000 | 0.0217 | 0.6097 |
| | Verb | 0.4048 | 0.2500 | 0.0133 | 0.5763 |

Table 4.3: Verb/Noun Scores on SemEval16 Task 14 Test Data

| | | system | | |
|---|---|---|---|---|
| | | merge | attach | |
| **key** | merge | 7 | 25 | 32 |
| | attach | 121 | 447 | 568 |
| | | 128 | 472 | 600 |

Table 4.4: Contingency table of test data word placements.

| **System** | **LM System** | **LM Key** |
|---|---|---|
| UMNDuluth Sys 1 (2 bonus) | 0.0984 | 0.1050 |
| UMNDuluth Sys 1 (10 bonus) | 0.1467 | 0.1533 |
| UMNDuluth Sys 1 (100 bonus) | 0.1550 | 0.1550 |
| OwS (2 bonus) | 0.0767 | 0.0767 |
| OwS (10 bonus) | 0.1667 | 0.1683 |
| OwS (100 bonus) | 0.1700 | 0.1700 |
| Relatedness | 0.0783 | 0.0783 |
| Word2vec (0.0 CV) | 0.0933 | 0.0933 |

Table 4.5: Lemma Match Scores on SemEval16 Task 14 Test Data

# 5 Conclusions

In this chapter we re-examine the discoveries of our system and their importance. We continue on to examine possible future directions in which our system can be research and applied.

## 5.1 Contributions of Thesis

This thesis investigated the problem of locating the optimal location for an Out -of- Vocabulary lemma to be inserted in WordNet. We applied various Natural Language Processing approaches to the problem, including: Extended-Gloss-Overlaps, Stemming, Similarity Measures, and Word2Vec. Our results, and the results of other SemEval16 Task 14 participants, demonstrate the strength of the idea that lemmas' glosses often contain the location where that lemma belongs. Even though our results did not exceed the second baseline from SemEval16 Task 14 (the system which chose it's hypernym by picking the first same part-of-speech word in the gloss), our variations explore other possibilities than choosing only a word from the OOV lemma's gloss. Our research expands the range of examined approaches and allow us to contribute the following discoveries:

1. Extended-Gloss-Overlaps: EGO's helped increase the amount of training data without increasing the overall data required, by incorporating in the glosses of the hypernyms and hyponyms of the candidate lemmas. This approach performed better than the random word baseline from SemEval16 Task 14, but did

not surpass the second baseline.

2. Stemming: Stemming helped to increase the number of relevant overlaps, by transforming data into similar roots. While stemming helped fix specific data examples, it did not surpass the EGO's. We believe this is tied to the data in the gloss, since the number of different tenses and plural nouns was not large enough to be relevant.

3. Relatedness: Relatedness measures use WordNet's inherent structure to naturally find the relatedness between a hypernym and the OOV lemma's gloss. Relatedness performed above the random word baseline, but not above EGO's, this was due to the fact that when calculating WordNet relatedness from the OOV lemma's gloss, the measure does not automatically know which WordNet sense each word from the gloss is.

4. Word2Vec: Word2Vec applied contextual similarity to the problem and we discovered that the Wu & Palmer score increased as the confidence value (CV) increased, but recall performed the opposite. This occurred since as the CV rises, less OOV lemmas are tested (decreasing the recall) but more confident ones are chosen (increasing the Wu & Palmer score).

5. WordNet::Extend: We coded our experiments and released them as open sourced in WordNet::Extend[1]. WordNet::Extend::Locate contains the code and experiments from this thesis for other developers to freely download. WordNet::Extend::Insert[2] contains code which allow a user to insert OOV lemmas into WordNet.

[1] http://search.cpan.org/~jonrusert/WordNet-Extend-0.052/lib/WordNet/Extend.pm
[2] Implementation found in Appendix A

These different variations increase the breadth of possible approaches to tackling this problem.

## 5.2 Future Work

Our results open a window into the variations that our systems could explore. For example, we experimented with the bonus in our initial system and also the stemming system to examine the effect on the chosen candidates. Likewise, we also tried variations of the confidence value on the word2vec system. These micro variations (changing a single system value) along with larger, macro variations (changing or combining new or current systems) lead to possible extended examination on the problem, by combining approaches already examined and reviewing the outcomes (E.g. confidence value with EGO's).

### 5.2.1 Micro Variations

For micro variations, it would be interesting to incorporate the confidence value into the other algorithms. Correspondingly, applying the bonus to word2vec candidate words could have a large effect on the output, since more relevant words (which are highlighted by the bonus) could cause a gloss to more easily pass the confidence value. Another variation on word2vec would be to experiment with different training data, by either changing the source altogether, or even increasing the amount of data. Another set of data that Word2Vec could be trained on is the English Wikipedia[3].

Stop lists were also consistent throughout the testing of the different systems. It could be beneficial to test the effectiveness of different sizes of stop lists to determine how great of impact they have on the systems.

[3] https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2

### 5.2.2 Macro Variations

Possible macro variations include combining the systems or adding in new data. As was used in CROWN (introduced in Background Chapter 2), Wiktionary data could be a large help as it would not only increase the overall data available, but also have links to words that might not appear in WordNet. Another idea is generalizing the words in the glosses to find an optimal location. This could be combined with Wiktionary to find a same part-of-speech word that doesn't appear in WordNet, then continue to generalize it until a WordNet word is found. This would take the *First sense, first baseline* approach and extend it to Wiktionary. A final macro variation deals with extending the research of *Relatedness*. As was noted, *Relatedness* was hindered by not knowing which sense of a word occurs in each OOV lemma's gloss. If the system could first apply Word Sense Disambiguation to the gloss, then calculate the relatedness, we believe the system would score better overall.

### 5.2.3 Beyond Nouns, Verbs, and Hypernyms

Since SemEval16 Task14 only examined nouns and verbs which can be attached or merged, our system followed suite. Since adjectives and adverbs do not have hypernyms, attach is irrelevant to them. However, other operations/relations exist in WordNet that incorporate adjectives, adverbs, as well as different relations for nouns and verbs. These different relations include: holonym/meronym(a part of a whole), derivationally related form, antonym, and more. Future work could include how these relations are determined by some of the previously examined methods.

### 5.2.4 No Location Found

Another challenge for finding the location of a new lemma in WordNet occurs if an OOV lemma is found not to be related to any existing word in WordNet, which means it cannot be inserted which a traditional operation but requires its own start in WordNet. Future work could determine if a lemma is not similar enough to any existing word and how that should be handled (whether in a new hypernym structure, or another variation).

### 5.2.5 Merge/Attach

A final idea for future research exists with merge/attach. Our systems currently uses the frequency of a candidate hypernym to determine whether to merge or attach. An increased exploration of this subject would allow future systems to increase the number of correct merge/attach guesses.

As shown, many possibilities still exist, and would provide compelling, continued research in this field and this problem. Increased research would only lead to a better, longer lived WordNet. This is important because WordNet is widely used, and will lose its viability if it cannot grow and adapt.

# A Appendix A

## A.1 Inserting New Words into WordNet

As seen the thesis focused on locating where a lemma should be inserted in WordNet. It became apparent early on that while finding a location is useful, if we cannot then proceed to insert that lemma into that location in WordNet we would not end up improving WordNet too much overall. Thus we arrived at the conclusion early on that the ability to insert new lemmas into WordNet would be a great help to the continuation of WordNet. When searching for Perl based utilities which allowed the inserting of a new lemma, only one was found[1]. However, this was deprecated, as it only worked with older versions of WordNet. This drove us to develop our own WordNet insertion functionality.

### A.1.1 WordNet Data Files

When researching how WordNet data is stored, we found that the WordNet database was created especially for WordNet in 1998 [4]. Wordnet stores its data in a low level representation with hard coded byte offset addresses representing the location of a specific sense. This made the inserting of a new lemma into WordNet much more difficult as once new bytes were added or changed in a file, all the following byte offsets had to be changed in all files they appeared in. WordNet's original source

[1]http://search.cpan.org/ dbrian/Lingua-Wordnet-0.74/Wordnet.pm

files were written by lexicographers. The source files were divided into four parts-of-speech (pos), noun, verb, adjective, adverb. The lexicographer files stored each word in WordNet, as well as their gloss and also pointers to other words. The pointers marked relations to other WordNet words (e.g. hypernyms). In order to convert the lexicographer files to a more portable/distributable format, the files were rewritten into several pos based files including index.pos, index.sense, and data.pos. It should be noted that the morphology (converting words to base lemma) of WordNet words occurs outside the database, in a function called morphy[2]. Morphy allows words such as *threw* to be search and WordNet to return *throw* (the correct base form).

### A.1.2 WordNet Data Format

Of the files mentioned above, we found that the aforementioned index and data files needed to be modified in order to insert a new word into WordNet. These include: index.pos and data.pos, where pos is one of four parts-of-speech (verb, noun, adj, adv) and contain information only for the specific pos; index.sense which contains senses for every lemma no matter which pos it is.

#### Index.pos

In index.pos each unique lemma is represented by a line of data formatted as:

`lemma pos synset_cnt p_cnt [ptr_symbol...] sense_cnt tagsense_cnt synset_offset [synset_offset...]`[3]

Where: *synset_cnt* refers to how many senses that lemma has; *p_cnt* and *ptr_symbol...* refer to the different relations it posses to other words (hypernym, hyponym, etc.); *sense_cnt* is the same as *synset_cnt* but was "preserved for compatibility issues";

[2]https://wordnet.princeton.edu/man/morphy.7WN.html
[3]https://wordnet.princeton.edu/wordnet/man/wndb.5WN.html

*tagsense_cnt* refers to the number of senses that are ranked according to their frequency in lexicographer texts; *synset_offset* refers to the byte offset in data.pos which contains the specific senses of lemma (there are *synset_cnt* number of *synset_offsets*.

As an example for index.pos, in index.noun *cocaine* is stored as

```
cocaine n 1 3 @ ~ \#s 1 1 03060294
```

This translates to: *lemma* is *cocaine*; *pos* is *n* or noun; *synset_cnt* and *sense_cnt* are both 1, which means there is only one sense of *cocaine*; *p_cnt* is 3 and is followed by three pointers "@   #s", which are hypernym, hyponym, and substance holonym respectively[4]; *tagsense_cnt* is 1 since only one sense exists it must be the only ranked; *synset_offset* is 03060294, which is the byte location where we will find *cocaine#n#1* in data.noun (since only one synset offset appears and *synset_cnt* is 1, cocaine only has one sense).

### Data.pos

In index.pos each **lemma** is represented on a single line. However, in data.pos each **sense** is represented on a single line formatted as:

```
synset_offset  lex_filenum  ss_type  w_cnt  word  lex_id
[word  lex_id...]  p_cnt  [ptr...]  [frames...]  | gloss
```

Where: *synset_offset* corresponds to the a same unique offset as was describe in index.pos; *lex_filenum* corresponds to the lexicographer file name which contains the synset; *ss_type* is a one character code which corresponds to the pos; *w_cnt* refers to the number of words in that synset; *word* and *lex_id* refer to the word in ASCII format and identifies the sense in the lexicographer files respectively; *p_cnt* is the number of pointers to different relationships; *ptr* refers to the data which points to each relation of the synset; *frames* (which only exists in data.verb) refers to the

[4] https://wordnet.princeton.edu/wordnet/man/wninput.5WN.html

generic verb sentence frames for a word in the synset; *glos* refers to the gloss of that synset.

As an example for data.pos, we follow up *cocaine* in data.noun. It is stored as

```
03060294 06 n 02 cocaine 0 cocain 0 007 @ 03492717 n 0000 #s 03060074
n 0000 + 00021679 v 0201 + 00021679 v 0202 ~ 02806274 n 0000 ~ 03066743
n 0000 ~ 03125184 n 0000 | a narcotic (alkaloid) extracted from coca
leaves; used as a surface anesthetic or taken for pleasure; can become
powerfully addictive
```

This translates to: *synset_offset* is 03060294 (the same as it was in index.noun); *lex_filenum* is “06”, which means *cocaine#n#1* is found in lexicographer file “06”; *ss_type* is ‘n’, the code for noun; *w_cnt* is “02”, meaning there are two words in this synset; *word* and *lex_id* correspond to both “cocaine 0” and “cocain 0”, showing *cocaine* and *cocain* as words in this synset with sense “0” in the lexicographer file; *p_cnt* is “007”, meaning *cocaine* has relations with seven different words; *ptr* covers the seven different pointers from “@ 03492717 n 0000” to “ 03125184 n 0000”, translating “@ 03492717 n 0000” to mean *cocaine* has a hypernym (@) located at byte offset “03492717” in the data.noun (n) file which is the current file (0000); *gloss* is everything after the “—” symbol.

#### Index.sense

The index.sense file differs from each of the previous files as instead of focusing on one pos, it includes all parts-of-speech. Index.sense contains all 206,941 senses within WordNet arranged alphabetically. Each line is formatted,

```
lemma%ss_type:lex_filenum:lex_id:head_word:head_id synset_offset
sense_number  tag_cnt
```
[5]

[5] https://wordnet.princeton.edu/wordnet/man/senseidx.5WN.html

Where: *lemma* refers to the lemma; *ss_type* refers to an integer which corresponds to the pos; *lex_filenum* refers to the lexicographer file name which contains the synset; *lex_id* refers to the sense in the lexicographer files; *head_word* appears only if the sense is in an adjective satellite synset and refers to the lemma of the first word of the satellite's head synset; *head_id* acts the same as *lex_id* but references the *head_word* in the lexicographer files; *synset_offset* refers to the byte offset in the data.pos file; *sense_number* refers to the sense of the word; *tag_cnt* refers to the number of times the sense has been tagged in WordNet texts.

Continuing our example for *cocaine* we find it stored in index.sense as

`cocaine%1:06:00:: 03060294 1 1`

Comparing to the format we find: *lemma* is *cocaine*; *ss_type* is 1, which corresponds to noun; *lex_filenum* is "06", meaning it is found in lexicographer file "06"; *lex_id* is "00", since it is the only sense, it need not be uniquely identified in the lexicographer file; *head_word* and *head_id* are both empty since this is not an adjective satellite synset; *synset_offset* is "03060294", which corresponds to the byte offset in data.noun; *sense_number* is "1", meaning this is the first sense of *cocaine*; *tag_cnt* is "1", which shows *cocaine* has been tagged once in WordNet texts.

Note how in each example, the files share some same information, showing the cohesivity of the files with each other. Each time a new word/sense is inserted into WordNet, each of these files would need to be updated to match.

### A.1.3 Implementation

Inserting a word into WordNet can either mean attaching it to an existing lemma as its hyponym or merging it into the synset, if it is a synonym. We split implementation between attach and merge.

**attach**

A word is attached into WordNet via the following process (visualized in figure A.1):

1. For the new lemma, calculate the new offset by traversing to the end of data.pos.
2. In data.pos:
   (a) Add new lemma to end of file.
   (b) Update lemma's hypernym to add the lemma's offset to the hypernym.
   (c) All other following offsets must be updated since more bytes of data have been added before them. Change all affected offets.
3. In index.pos:
   (a) If new word:
      i. Add new lemma alphabetically.
      ii. Change all updated offsets from data.pos.
   (b) If existing word:
      i. Update existing line with new offset and other information.
      ii. Change all update offsets from data.pos.
4. In index.sense:
   (a) Add new lemma sense alphabetically.
   (b) Change updated offsets that correspond to the same pos.
5. In all other data.pos files:
   (a) Change updated offsets that correspond to the new lemma's pos.

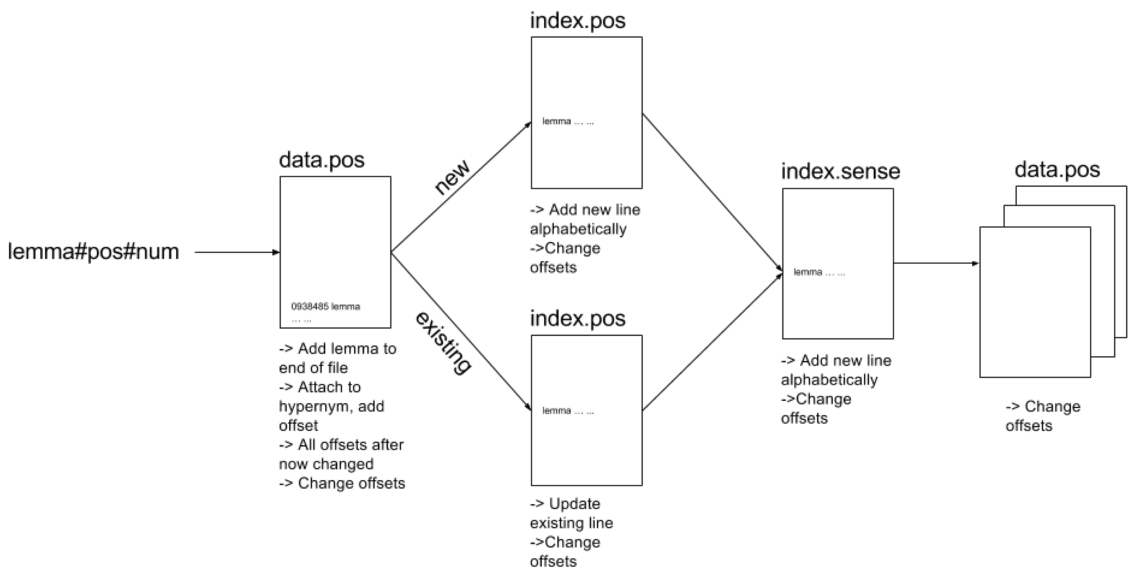



Figure A.1: Visualization of attaching a new lemma

### merge

A word is merged into WordNet as a synonym via the following process (visualized in figure A.2):

1. In data.pos:
   (a) Update existing data line, adding in the word to the synset.
   (b) All other following offsets must be updated since more bytes of data have been added before them. Change all affected offsets.
2. In index.pos:
   (a) If new word:
      i. Add new lemma alphabetically.
      ii. Change all updated offsets from data.pos.
   (b) If existing word:

        i. Update existing line with new offset and other information.

        ii. Change all update offsets from data.pos.

3. In index.sense:

    (a) Add new lemma sense alphabetically.

    (b) Change updated offsets that correspond to the same pos.

4. In all other data.pos files:

    (a) Change updated offsets that correspond to the new lemma's pos.

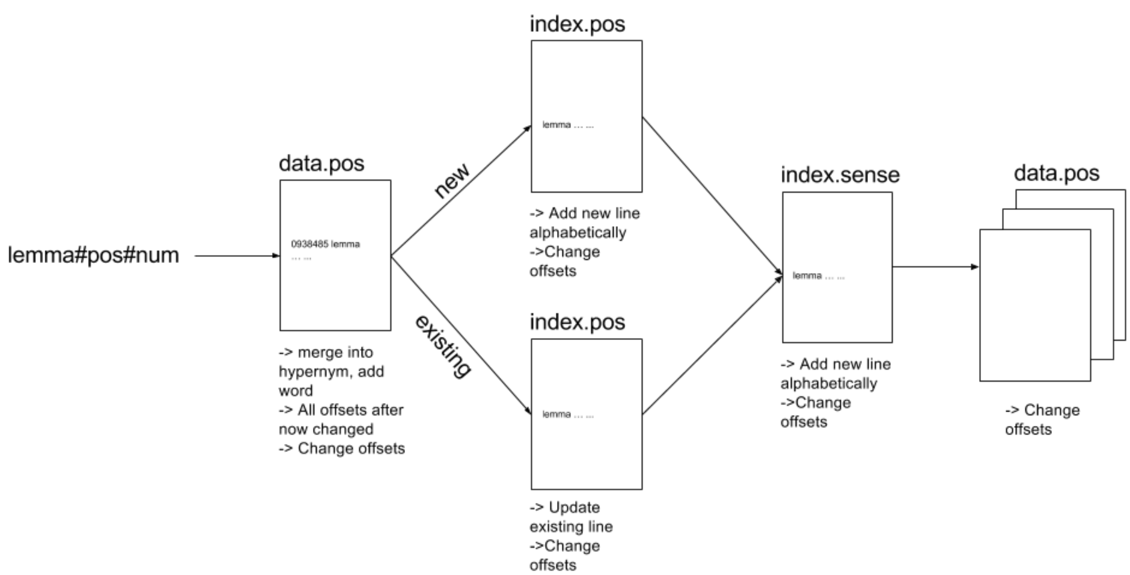



Figure A.2: Visualization of merging a new lemma

### A.1.4 Discussion

Since our system has the option to attach or merge, two operations that deal with hypernyms/hyponyms and synsets, our system has the ability to add in new nouns and new verbs. Our system has been run against tests of WordNet::QueryData[6] to ensure our insertions would not negatively affect other systems that use WordNet. Our system is currently being hosted on cpan as open-source code[7].

### A.1.5 Future Work

This system right now focuses primarily on inserting hypernym relationships. Adjectives and adverbs do not have hypernym relations and therefore cannot be added in by this method. Our initial system was fine with leaving these out however, as nouns and verbs make up a bulk of WordNet's data. Future work on this system would then work to include adjective and adverb insertions.

Besides hypernyms and hyponyms, WordNet stores many other relations between words. These include: holonym/meronym (a part of a whole), derivationally related form, antonym, and more. Future work in this area could work to incorporate these relationships in, either when inserting initially, or adding relations in as they are noticed later.

Since words change, the meaning of words may change overtime as well and wished to be changed. Future work could include the ability to change existing glosses. Subsequently, if a relationship between two words is discovered after a word has been inserted, the ability to add in relationships would be useful.

Currently, the system runs under the assumption that any OOV lemma would share a relation with an existing WordNet word. However, we might discover an OOV lemma that does not cleanly have a relation to any current WordNet word (e.g. proper names). Future work would work to include the ability to insert an OOV lemma into WordNet without having to choose a specific hypernym.

[6] http://search.cpan.org/~jrennie/WordNet-QueryData-1.49/QueryData.pm
[7] http://search.cpan.org/~jonrusert/WordNet-Extend-0.052/lib/WordNet/Extend.pm

# References

[1] S. Banerjee and T. Pedersen. "Extended Gloss Overlaps As a Measure of Semantic Relatedness". In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI'03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003, pp. 805–810. URL: http://dl.acm.org/citation.cfm?id=1630659.1630775 (cit. on pp. 7, 16, 29, 38).

[2] R. L. Cilibrasi and P. M. B. Vitanyi. "The Google Similarity Distance". In: *IEEE Trans. on Knowl. and Data Eng.* 19.3 (Mar. 2007), pp. 370–383. ISSN: 1041-4347. DOI: 10.1109/TKDE.2007.48. URL: http://dx.doi.org/10.1109/TKDE.2007.48 (cit. on pp. 20, 40).

[3] P. Edmonds and S. Cotton. "SENSEVAL-2: Overview". In: *The Proceedings of the Second International Workshop on Evaluating Word Sense Disambiguation Systems*. SENSEVAL '01. Toulouse, France: Association for Computational Linguistics, 2001, pp. 1–5. URL: http://dl.acm.org/citation.cfm?id=2387364.2387365 (cit. on p. 11).

[4] C. Fellbaum, ed. *WordNet: an electronic lexical database*. MIT Press, 1998 (cit. on p. 50).

[5] J. J. Jiang and D. W. Conrath. "Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy". In: *International Conference Research on Computa-

*tional Linguistics*. 1997. URL: http://arxiv.org/pdf/cmp-lg/9709008.pdf (cit. on p. 10).

[6] D. Jurgens and M. T. Pilehvar. "Reserating the awesometastic: An automatic extension of the WordNet taxonomy for novel terms". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL HLT 2015)*. Denver, Colorado, 2015 (cit. on pp. 16, 17).

[7] D. Jurgens and M. T. Pilehvar. "SemEval-2016 Task 14: Semantic Taxonomy Enrichment". In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, California: Association for Computational Linguistics, June 2016, pp. 1092–1102. URL: http://www.aclweb.org/anthology/S16-1169 (cit. on pp. 25, 36).

[8] M. Lesk. "Automatic Sense Disambiguation Using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone". In: *Proceedings of the 5th Annual International Conference on Systems Documentation*. SIGDOC '86. Toronto, Ontario, Canada: ACM, 1986, pp. 24–26. ISBN: 0-89791-224-1. DOI: 10.1145/318723.318728. URL: http://doi.acm.org/10.1145/318723.318728 (cit. on pp. 5, 29).

[9] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, June 2014, pp. 55–60. URL: http://www.aclweb.org/anthology/P/P14/P14-5010 (cit. on p. 17).

[10] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Curran Associates, Inc., 2013, pp. 3111–3119. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf (cit. on p. 21).

[11] S. Patwardhan, S. Banerjee, and T. Pedersen. "Using Measures of Semantic Relatedness for Word Sense Disambiguation". In: *Proceedings of the 4th International Conference on Computational Linguistics and Intelligent Text Processing*. CICLing'03. Mexico City, Mexico: Springer-Verlag, 2003, pp. 241–257. ISBN: 3-540-00532-3. URL: http://dl.acm.org/citation.cfm?id=1791562.1791592 (cit. on p. 10).

[12] S. Patwardhan and T. Pedersen. "Using WordNet-based Context Vectors to Estimate the Semantic Relatedness of Concepts". In: *Workshop on Making Sense of Sense at the 11th Conference of the European Chapter of the Association for Computational Lingustics*. 2006, pp. 1–8. URL: http://www.aclweb.org/anthology/W06-2501 (cit. on p. 12).

[13] T. Pedersen, S. Patwardhan, and J. Michelizzi. "WordNet::Similarity: Measuring the Relatedness of Concepts". In: *Demonstration Papers at HLT-NAACL 2004*. HLT-NAACL–Demonstrations '04. Boston, Massachusetts: Association for Computational Linguistics, 2004, pp. 38–41. URL: http://dl.acm.org/citation.cfm?id=1614025.1614037 (cit. on p. 40).

[14] H. Rubenstein and J. B. Goodenough. "Contextual Correlates of Synonymy". In: *Commun. ACM* 8.10 (Oct. 1965), pp. 627–633. ISSN: 0001-0782. DOI: 10.

1145/365628.365657. URL: http://doi.acm.org/10.1145/365628.365657 (cit. on p. 10).

[15] H. Schtze. "Automatic Word Sense Discrimination". In: *Comput. Linguist.* 24.1 (Mar. 1998), pp. 97–123. ISSN: 0891-2017. URL: http://dl.acm.org/citation.cfm?id=972719.972724 (cit. on p. 12).

[16] C. Spearman. "The Proof and Measurement of Association between Two Things". In: *The American Journal of Psychology* 15.1 (1904), pp. 72–101. ISSN: 00029556. URL: http://www.jstor.org/stable/1412159 (cit. on p. 10).

[17] H. Tanev and A. Rotondi. "Deftor at SemEval-2016 Task 14: Taxonomy Enrichment using Definition Vectors". In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, California: Association for Computational Linguistics, June 2016, pp. 1218–1221. URL: https://www.aclweb.org/anthology/S/S16/S16-1210.pdf (cit. on p. 40).