Using WordNet and Short-Term Memory for Contextual Disambiguation

William T. F. Strachan

Word Count - 4507

Supervisor - Dr Dimitar Kazakov
Department of Computer Science
University of York
20 November 2016

Contents

1 Literature Review

In order to define a problem, we must first establish the previous works, so as to build upon them effectively.

1.1 Natural Language Processing

The study of natural language processing aims to allow a computer to understand natural language, and formulate a relevant response based upon its input. Within this, the problem of input text analysis has traditionally be broken down into smaller sub-problems [1]:

- Text Preprocessing
- Lexical Analysis
- Syntactic Parsing
- Semantic Analysis

The subsequent subsections will discuss each of these in more detail.

1.1.1 Text Preprocessing

Before any analysis can take place, the inputted raw text must be converted into a usable format. This, once again, can be broken down into multiple steps [1]:

- **Document Triage**
  - Character encoding must be identified.
  - The language can then be identified.
  - Non-useful data, such as images and html formatting must be removed.
- **Text Segmentation**
  - Individual words (tokens) must be separated from one another.
  - Text Normalisation; replacing multiple equivalent tokens with one token (e.g. "Ave." and "Avenue").
  - Identifying sentences, i.e. locating where a sentence begins and ends.

1.1.2 Lexical Analysis

One word can have multiple forms, for example "judge" (the lemma) has the forms {"judge", "judges", "judging", "judged"} (morphological variants). The job of Lexical Analysis is to replace all morphological variants of a word, with their corresponding lemma, a process known as stemming [1].

1.1.3 Syntactic Parsing

Whwn derviving meaning from sentances, the grammatical structure can provide important insight. The Syntactic parsing technique extracts this infomation using two processes [2]:

- **Part-of-speech (PoS) Tagging**
  - Each word is given tags denoting their syntactic role (e.g. noun/adjective/verb).
- **Chunking**
  - Noun phrases and verb phrases are detected and tagged using "begin chunk" and "inside chunk" labels

1.1.4 Semantic Analysis

The semantics of a sentence, is the meaning given by it's tokens [3]. The topic of semantic analysis will be expanded upon on in the Previous Works Section.

1.2 Psycholinguistics

Language understanding is a problem which, it can be stated, is solved by the human brain. From this satatement, it can be derived that a computational solution could be effectively built around knowledge of the processes at work in the brain. The process of language comprehension can be described using the working memory model [4].

According to Baddely et al. [4] there exist multiple, special purpose, memory structures within two main categories, the Short-term Store and the Long-term Store.

1.2.1 Long-term Store

The long-term store (LTS) contains semi-permanent information. Within the LTS, there exist Explicit and Implicit memory structures. The contents of the Implicit memory describe skills and methods of doing things, whereas the Explicit memory contains factual information [4]. When considering these structures, it can be seen that Explicit memory is of greater interest in the context of NLP.

Within the Explicit memory exists knowledge of semantics [4]. The information held here not only defines concepts (meanings of word forms), but also their attributes and rules of use. In 1966, M. Quillain proposed a model of Semantic Memory [5]. The model consists of a graph of nodes, each representing

a concept, connected by edges of differing types, each representing a different syntactic feature (for example, hypernym).

1.2.2 Short-term Store

The short-term store (STS) is a structure of limited capacity, used to store items for periods usually of no more than a few seconds [4]. In 1955, G. Miller, based upon previous experimental results, concluded that the size of the STS existed in the realm of 7 ± 2 items of information [6].

In 1971, R. Atkinson and R. Shiffrin proposed a model of the STS [7]. In this model, the STS can both send information to, and draw information from the LTS. Inputs from the sensory registers (memory structures holding information relating to inputs from senses) are also sent to the STS. Atkinson and Shiffrin proposed that, over time, the activation of items in the STS decreased; they went on to theorise that items could be lost from the STS, only when a new, more highly activated item could take its place. To counter this loss of activation, the authors discussed the control process, rehearsal. This process makes use of repetition to increase the activation of items in memory, decreasing their chance of loss.

1.2.3 Disambiguation Models

In some cases, when assigning meaning to words, ambiguity can arise. Some words have multiple concepts, for example, bank can refer to a building, or a sloped surface alongside a body of water. In such cases, the brain uses some process to select the correct concept. One such model of this disambiguation is the Multiple-access model [8].

According to the multiple-access model, when presented with an ambiguious word, initially all corresponding concepts are activated [9]. The most appropriate concept is then chosen using context and frequency.

Previously, we established that the process of disambiguation begins with the activation of all possible word concepts. The context-sensitive model deals with the use of context and frequency in the selection of the most appropriate of these [8]. In cases where the context is strong, i.e. the correct concept can be chosen using it's surrounding context, the context is primarily relied upon for disambiguation. In the opposite case, i.e. when context gives little indication of which is the correct concept, the most frequenly used concept is used, assuming it fits with the available context.

1.3 Wordnet

In 1990, it was noted by G. Miller et al. that current attemps to organise the english lexicon, i.e. conventional dictionaries, offered few benefits when used in

conjunction with computers [10]. Wordnet was an effort to produce a dictonary, containing more information than a conventional dictionary, that could be useful for computational applications.

Central to the design of wordnet, is the idea of synsets [10]. The authors began using the assumption that all concepts can be uniquely defined by their set of synonyms (words which share like meaning). In most cases, this assumption holds true, though, in cases where more detail is required, a "gloss" was added [10].

Wordnet builds upon models of the semantic memory, such as that discussed in the Long-term store section [10]. The overall structure relies on four main semantic relations:

- Synonymy
  - If two words are to be called synonyms, they must share at least one like meaning.
- Atonymy
  - Conceptually, Atonymy can be seen as the opposite of Synonymy. Atonymy is difficult to define, as not all words which share opposite meaning can be called atonyms, for example, {up, down} is an atonym pair, but {up, fall} is not.
- Hyponymy
  - If we consider a a synset to be a object-oriented class, its hypernym can be considered its parent class, for example, birch is a type of tree.
- Meronymy
  - Meronymy is relationship between two synsets where one is a part of another, for example, a goat has horns, therefore horn is a meronym of goat.

It is common knowledge that words can fall into one of a number of categories, nouns, adjectives, verbs and adverbs. G. Miller et al. note that, due to the differences in the relations between words in these categories, each type has differs in the structure they produce and are therefore held in different files [10]. The proceeding subsections will go into each of thesecategories in more detail.

1.3.1 Nouns

G. Miller et al. note that a noun can be defined using only its immediate hypernym, and how it differs from its hypernyms other hyponyms [11]. From this, it can be seen that hyponymy is perhaps the most important relation in

the organisation of nouns. For this reason, nouns form a hierachical structure in wordnet.

Wordnet's designers stated the assumption that all nouns can be contained in a single hierachial structure [11]. The issue with having a single word, of which all other words are hyponyms, is that this hypernym is relatively meaningless. It was instead decide to divide all words into 25 separate files, each containing a hierachical tree beginning with one of the following synsets [11]:

{act, action, activity}

{animal, fauna}

{artifact}

{attribute, property}

{body, corpus}

{cognition, knowledge}

{communication}

{event, happening}

{feeling, emotion}

{food}

{group, collection}

{location, place}

{motive}

{natural object}

{natural phenomenon}

{person, human being}

{plant, flora}

{possession}

{process}

{quantity, amount}

{relation}

{shape}

{state, condition}

{substance}

{time}

Other than synonymy, nouns have three other important features [11]:

- Attributes
  - The attributes of a noun consist of adjectives which distinguish it from other hyponyms of its hypernym, for example {huge, green, fluffy}.
- Parts
  - The parts of a noun consist of its meronyms, described previously.
- Functions
  - The functions of a noun consist of verbs which are associated with its actions, for example chair has the functions {sit, rest}.

1.3.2 Adjectives

Adjectives in can be divided into four distinct groups, each implying a different structure of semantic links [12]:

- Descriptive Adjectives
  - As stated in the previous section, nouns have attributes. Descriptive adjectives act as modifiers for these attributes: for example, a building has a height, by saying "tall building", the height attribute is given a value [12]. Atonymy, defined previously, is considered by wordnet's designers to be the most important relation between descriptive adjectives. Unfortuantely, not all adjectives have atonyms, leading to the designer's addition of an indirect antonym" semantic link, between synonyms of a word and its atonym [12]. These semantic links give rise to a structure made up of pairs, linked to one another by their synonyms.
- Reference-Modifying Adjectives
  - Reference-modifying adjectives have an adverb form which can be used to convey the same meaning [12]. For example, the noun-phrase "the former manager", can be modified to become "the man who was formerly a manager", without diverging from its original meaning. There exist relatively few examples of this category, so no overarching structure emerges, that being said, in some cases, the atonym relation does occur [12].
- Colour Adjectives
  - As their name suggests, colour adjctives concern the value of the colour attribute. This definition implies that these words should, in fact, fit into the descriptive adjective category. Their separation is given by colour adjectives lack of true atonym (excluding modifiers such as "light" and "dark") [12]. The lack of clear semantic relations between these words poses a problem for their organisation, leading wordnet's designers to link them using their definitions, i.e. using hue, lightness and saturation.
- Relational Adjectives
  - In the phrase "maternal instinct", it can be seen that the adjective is derived from a noun, in this case "mother"; this is the defining feature of relational adjectives [12]. In wordnet, relational adjectives are linked to their noun form, meaning they don't posess their own structure, instead falling into that described in the Noun section.

1.3.3 Verbs

In C. Fellbaums 1990 paper, "English Verbs as a Semantic Net", she discussed the lack of true synonymy across the verb category [13]. This is an issue for Wordnet's designers, with their reliance on synsets. The author goes on to describe the solution, periphrases, the use of verb phrases to give more meaning to a simple verb. In the paper, the example synset, {swim, travel through water} was given.

The relationships between verbs follow a hierachy shown in Figure 1, with each type elaborated upon in the following paragraphs.

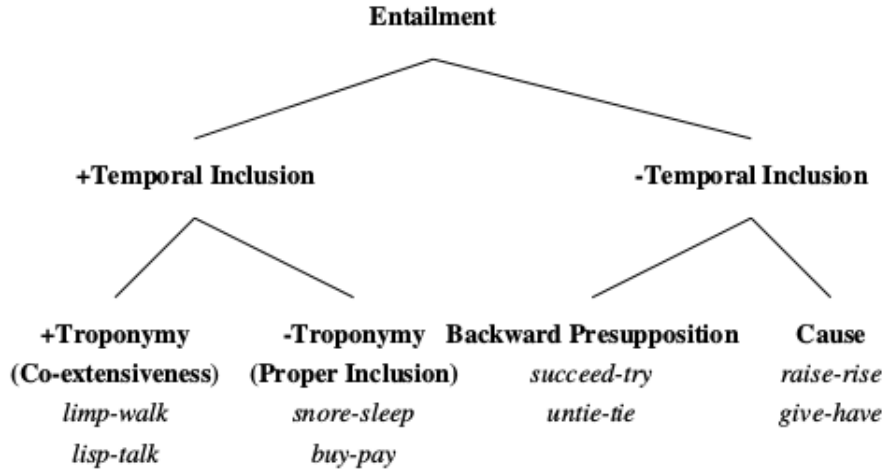

Figure 1: HIERACHY OF SEMANTIC RELATIONS BETWEEN VERBS [13, p. 15]

Entailment is a relation, simlar in nature to hyponymy. A verb (a) is said to entail another (b) if, when b is substituted for a in a sentace, the truth value of the sentance remains the same [13]. For example, "run" entails "move", "she ran" can also be described by "she moved". If a entails b, and b entials a, a and b are synonyms.

Temporal Inclusion is a form of entailment where one verb (a) is temporally included in another (b) if, b can only occur during the same time period as a [13]. For example, "swallow" is temporally included by "consume". If a can also occur without b, the entailment can be called **Proper Inclusion**.

Troponomy is a type of entailment where a verb (a) can be said to be a way of doing another verb (b) [13]. For example, "to nap" is a troponym of "to sleep".

Backward Presupposition is a relation which occurs when one verb (a)

is a precondition of a verb (b) [13]. For example, one must "play" before they can "win".

Cause is a relation where one verb (a) is considered the causative verb, and another (b) is considered the resultative verb, i.e. a causes b to occur [13]. For example, "to teach" is related to "to learn".

The entailment relation leads to a similar structure to that of nouns, a hierachical tree. This structure differs though in its relative lack of depth, caused by the increased average number of concepts per word, when compared to nouns [13].

1.4 Previous Work

1.4.1 Latent Semantic Analysis

In 1997, T. Laundaur and S. Dumais proposed a purely statistical method of semantic analysis, called Latent Semantic Analysis (LSA) [14]. The model can be visualised as a set of nodes, each representing a word, existing in semntic space. Words placed close together can be said to have similar meaning, and when close enough together, can be called synonyms.

When training the model, words which exist in the same context (i.e sentence, paragraph) are linked by a distance derived from their distance in the text. This distance is then adjsted according to similar occurences in more training data. Given enough training data, the distances between each node can be used to give the most likely synonyms of a word, given the context [14].

The authors found that, when presented with a synonym test, the model's best performance gave 64.4% correct answers , which, when compared to US undergraduates for whom English is not their first language (scoring 64.5% on average), could be considered to be reasonably effective [14]. This model only considers likelyhood, which, as discussed in the fDisambiguation Models section, is only one of the two methods the brain is theorised to use for this process. By using a similar method in conjunction with a context focused model, a more accurate and useful model could be produced.

1.4.2 Extended TF-IDF

TF-IDF is an algorithm which, when given a document, outputs its most important words, often used to organise a set of documents into categories. This is done by calculating the importance of each word in the document [15], using the formula:

$$t_i = f_t * (log_2(n) - log_2(f_d) + 1)$$

Where t_i is the importance of the word, f_t is the frequency of the word within the document, n is the number of documents in the corpura, and f_d is the

frequency of the word across all documents in the corpura [16]. The words with the highest importance are then outputted.

In 2004, J. Sedding and D. Kazakov proposed an extension to the TF-IDF algorithm, using Wordnet. The aim of the paper was to include extra information provided by wordnet (synsets and hypernyms) in conjuncton with PoS tagging, to provide an output more suited to the categorisation of documents [16]. The authors ran multiple experiments, in order to compare different usage of additional information. unfortunately, it was found that their method was less effective than TF-IDF alone. As the authors stated, this is likely due to the lack of disambiguation, i.e. all synsets were used for each word, adding noise.

1.4.3 Neural Networks

In recent years the use of Neural Networks, more specifically Deep Neural Networks, in the solving of computational problems has dramatically increased. More recently, this technique has been applied to the problem of natural language processing. As discussed in the Natural Language Processing section, it was shown that assigning varyng labels to words and phrases (i.e. categorisation) made up a large proportion of the processes, a job neural networks are wellsuited to, and often associated with [17].

The first process requrired in the use of neural networks, is the conversion of raw text into a vector-base representation, giving a sentance as a set of vectors. The sentence is then passed into the neural network [17]. The network proposed by R.Collobert and J. Weston in 2008 contained multiple layers, each serving its own purpose [17]. Initially the network identifies word features, before using this data to calculate the probability of each synset of each word.

It was found by the authors that their model provided good results, though, in order to get these results, a large amount of time was deicated to training [17].

1.4.4 Previous use of Wordnet and Short-term Memory for Disabiguation

In 2007, a University of York student, M. Burke, produced a project with a simlar aim to the one this report describes. This project builds upon the work and findings of my predecessor. The model developed made use of memory structures based upon those discussed in the Psycholinguistics section, i.e. Short-term (STM) and Semantic memories [18]. The Short-term memory, was a list, containing synsets, each with its own activation, and Wordnet was used as the Semantic memory, once again each synset has its own activation.

As words are encountered, their activation, and the activation of their hypernyms is increased. The activation increase of the initial synset was found through experimentation, though the activation increase of hypernyms differed

according to the below equation [18].

$$H = \sum_{i=1}^{N} S_i \times A$$

Where A is the attenuation, found through experimentation, S_i is the activation of the hyponyms. This model was used to prevent more general synsets dominating the short-term memory, whilst boosting hypernyms of synsets more if a similar synset also occurs in close proximity [18].

The author noted that, highly activated synsets could remain in the short-term memory indefinitely. To counter this issue, and to remain in line with the memory models discussed in the Short-term store section, the process of forgetting was added to the model, by multiplying the activation value by a number found by experimentation [18]. Forgetting occurs over time, decreasing the activations of items in the Short-term memory and the Semantic memory, the latter by a greater degree.

In the proposed model, the corpus is processed sentence by sentence. Two methods of disambiguation were proposed, with both cases beginning by removing non-useful words, and converting all words into their base forms (e.g. "flies" $\Rightarrow$ "fly"), each model differs in its use of the contents of the short term memory.

Hypernym-first - The hypernyms of all words are activated, with the synsets with the highest activation bieng used to disambguate each word [18].

STM-first - The contents of the Short term memory's hyponyms are searched until one of the synsets present in the sentence are found [18].

As mentioned previously, the values of some variables were found using experimentation. In these experiments, four variables were altered [18]:

- Disambiguation Method

  The author found that STM-first was marginally better (1% difference)

- Short-term memory size

  It was found that a STM size of 5 was optimal, with a 67% accuracy.

- The Attenuation value

  The author found that an attenuation value of between 0.7 and 0.9 gave the best result, with an accuracy of 67%.

- Forgetfulness

It was found that a small amount of forgetfulness (multiply activation by 0.95) in conjunction with a small difference in forgetfulness between Semantic and Short-term memories (multiplied by 1.05), produced the best result [18].

Unfortunately, the model failed to produce the correct synset significantly more accurately than the author's baseline (selecting the most common synset) [18]. As suggested in the report, this may be due to semantic links, which occur in the brain, not being available or utilsed by the model. Though, it may also be the case that the mathematical models used were inaccurate, with less linear functions being required.

2 Implementation

The application was implemented using the Python programming language, chosen for the availability of the Natural Language Toolkit (NLTK) [19]. NLTK provides an interface for use with Wordnet and a number of useful corpura (collections of tagged raw text), to be used as test data.

2.1 Corpus Analysis

Using the corpus reader built into NLTK, an entire document can be analaysed as a series of hierachical lists [19]:

- **Document**, is a list of sentences,
- **Sentences**, each of which is a list of words.

The analyser traverses each of these lists, through a series of loops, processing them a sentance at a time.

2.2 Activation

The corpus is read a sentence at a time, for each word in the sentence a series of activations takes place. Initially, the synsets of each word are activated, followed by their hypernyms following the algorithm outlined in Listing 1.

```
1 FUNCTION activateHypernyms(synset, depth):
2     activationModifier = hypernymModel(depth)
3     IF activationModifier < 0 THEN
4         RETURN
5     ELSE
6         FOR hypernym in synset.hypernyms() LOOP
```

```
7                activateHypernyms(hypernym, depth + 1)
8            END LOOP
9            RETURN
10       END IF
11   END FUNCTION
```

Listing 1: Hypernym Activation

The function recursively traverses the tree of hypernyms an example of which is shown in Figure 2, activating each synset until, the model gives a result of less than zero. As the function traverses through each hypernym, the amount they are activated by decreases, favouring less general synsets.

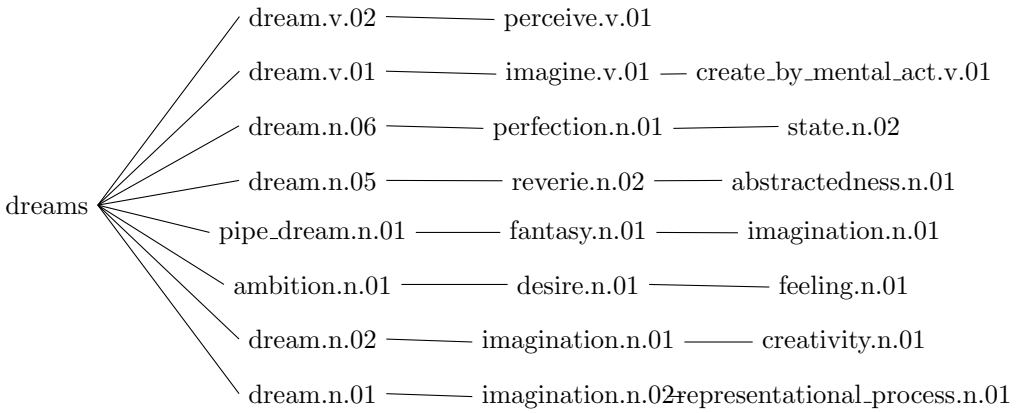

Figure 2: HYPERNYM TREE OF "DREAMS"

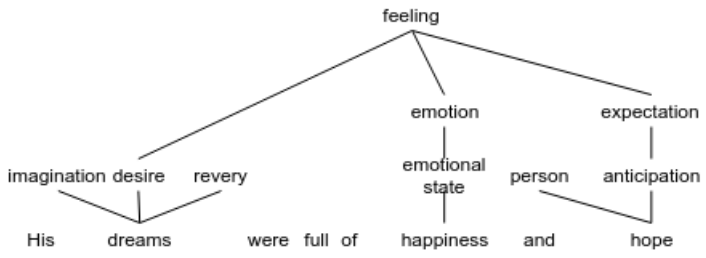

Figure 3: HYPERNYM ACTIVATION

Upon activation, the synset in question's activation is increased by an amount, dependant upon the hypernym model used (discussed later in this section). The same synset can be activated multiple times within the same sentence, an example of which is shown in Figure 3. In this case, a similar model to that proposed

by M. Burke [18] is used:

$$H = \sum_{i=1}^{N} S_i$$

The total acivation increase is equal to the sum of all synset activations within the sentence.

It can be seen that, in Listing 1,there exists a function, hypernymModel. This function is responsible for reducing the activation increase of hypernyms as they become more general (closer to the top of the tree). This function should favour most heavily, less general hypernyms, i.e. those activated by the first few recursions of activateHypernyms.

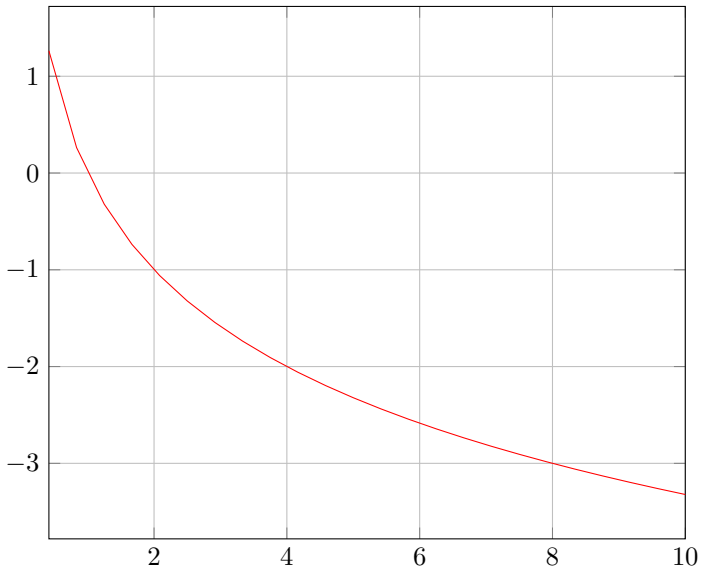

Figure 4: GRAPH OF $-\log(x)$

One function which fits this criteria is $-\log(x)$, shown in Figure 4. The values given by the function, for higher values of x are relatively high, favouring more general hypernyms heavily. For lower values of x, the values given are relatively low, causing more general hypernyms to have a lower activation increase.

It may be necessary to tune this function during experimentation, to provide greater accuracy, meaning more variables must be added. With these additions, the full model is:

$$f(x) = -b\log(ax)$$

where a and b are varied during experimentation, and x is the depth of the hypernym. When the depth = 0 (i.e. the activation of the root synset) the

model would require the calculation of log(0), which is undefined. For this reason, an offset of 0.001 was added, making the final model:

$$f(x) = -b\log(ax + 0.001)$$

The size of the offset needed to be small to reduce its impact on the overall model.

2.3 Memory Structures

Upon activation, the synset in question may enter the memory structures, if it posesses a high enough activation. Both an STS and Episodic Buffer have been implemented, using python classes for each. Interactions between the two memory structures are handled by a Memory Controller. As discussed in the WordNet section, WordNet provides a good model of the semantic memory [10],described in the Long-term Store section. For this reason it has been used heavily.

2.3.1 STS

The STS (Stm in psuedocode) can, in a simplified sense, be described as a list containing memory items. These memory items have been implemented in python as an object with two attributes, synset (immutable) and activation (mutable), based upon the contents of the STS in the working memory model. These memory items can be replaced, if a synset with greater activation appears. The process of synset replacement is handled by the swapLowestItem function shown in Listing 2.

```
1  FUNCTION swapLowestItem(newItem):
2          IF Stm.size < self.maxStmSize THEN
3              Stm.add(newItem)
4              RETURN None
5          ELSE
6              self.lowestItem = self.getLowestActivation()
7              IF newItem.Activation < stm.lowestActivationItem THEN
8                  RETURN None
9              ELSE
10                 Stm.remove(lowestActivationItem)
11                 Stm.add(newItem)
12                 RETURN stm.lowestActivationItem
13             END IF
14         END IF
15 END FUNCTION
```

Listing 2: the swapLowestItem function

A notable feature of the STS is its limited capacity, as described by G. Miller in 1955 [6]. It is for this reason that the ability to swap memory items is required. In the case that the STS isn't full, the new memory item is simply added with no swap taking place.

2.3.2 Episodic Buffer

The purpose of the episodic buffer, is to maintain a list of synsets which have occurred in the STS previously. This allows future activations of synsets which have already occured to receive a boost when activated in the future. Interactions with the Episodic buffer only occcur during the activation of synsets, therefore, all interactions are handled by the Memory Controller.

2.3.3 Semantic Memory

The Semantic Memory is responsible for maintaining all the semantic knowledge of the system. It was decided that WordNet provided the best solution, with its in depth knowledge of synsets and their relations to eachother [11, 13].

NLTK's built in WordNet reader [19] was used to interact with WordNet. Throughout the system, synsets are used. Each of these is a reference to a synset present in WordNet, allowing their semantic relationships (accessed through their methods) to be used.

2.3.4 Memory Controller

A memory controller has been implemented to handle interactions between the STS and episodic buffer. The most notable of its uses is the activation of synsets. In order to activate a synset, the Memory Controller must take into account the whether the synset is present in any of the afformentioned memory structures, and handle the activation accordingly. The activateSynset function, shown in Listing 3 is responsible for handling this process.

```
1  FUNCTION activateSynset(self, synset, activationModifier):
2          IF stm.inContents(synset) THEN
3              synset.activate(activationModifier)
4              RETURN
5          ELSE IF episodicBuffer.inContents(synset) THEN
6              newMemItem = MemItem(synset, 1)
7              newMemItem.activate(activationModifier)
8              sendToStm(newMemItem)
9              RETURN
10         ELSE
11             newMemItem = MemItem(synset, 0)
12             newMemItem.activate(activationModifier)
```

```
13                  sendToStm(newMemItem)
14                  RETURN
15              END IF
```

Listing 3: the activateSynset function

If the activateSynset finds the synset in the STS, the synset is simply activated using, and remains there. If the synset is not present in the STS, it is activated, and sent to the STS, through the sendToStm function, shown in Listing 4, recieving a boost if it has previously existed in the STS (i.e. it is in the Episodic Buffer).

```
1  FUNCTION sendToStm(inputSynset):
2           returnedItem = self.stm.swapLowestItem(inputItem)
3       IF returnedItem is None THEN
4           RETURN
5       ELSE IF NOT episodicBuffer.inContents(returnedItem) THEN
6           episodicBuffer.addSynset(returnedItem)
7           RETURN
8       END IF
9  END FUNCTION
```

Listing 4: the sendToStm function

The sendToStm function, shown in Listing 4, is responsible for changing the contents of the STS and updating the Episodic buffer accordingly. The swapLowestItem function, shown in Listing 2 is used to edit the STS. As discussed previously, this function can return a synset, if the swap is successful. In this case, sendToStm will add the synset previously in the STS, to the episodic buffer (if it is not already present there).

Throughout the analysis of a corpus, the activation of a specific memory item can change by either being forgotten or activated.

2.4 Forgetting

After each sentence is read, all items in the STS must be forgotten (have their activation reduced), so as to comply with Atkinson and Shiffrin's STS model discussed previously [7]. The forgetting of the entire contents of the STS is done using a loop, as shown in Listing 5.

```
1  FOR item IN Stm LOOP
2           item.forget()
3  END LOOP
```

Listing 5: Forget loop

The forgetting process occurs after every sentence, giving synsets in the next sentence the opportunity to enter the STS.

When a synset is forgotten, its activation may fall below a threshold, varied in experimentation. If this occurs, the synset is removed from the STS and, as with the sendToStm function, is added to the Episodic Buffer. This prevents synsets irrelevant to the current context from existing in the STS.

2.5 Disambiguation

Up to this point, we have only discussed the processes involved with changing the contents of the memory structures. The purpose of these memory structures is to provide the surrounding context of a word, to aid in its disambiguation. There exists a delay between the processes described previously (i.e. the initial reading of the sentence) and the disambiguation phase. This delay allows inappropriate synsets to leave the STS before they are used. The duration of this delay is to be adjusted during experimentation.

As discussed in the Disambiguation Models section, the multiple access model relies upon context and frequency, using whichever is stronger for disambiguation [8]. The system discussed in this section makes use of both of these, through the disambiguate function, shown in Listing 6.

```
1  FUNCTION disambiguate(synsetList):
2      FOR item in stm LOOP
3          IF item in synsetList THEN
4              RETURN item
5          END IF
6      END LOOP
7      FOR item in stm LOOP
8          returnedSynset = hyponymSearch(synsetList, item)
9          IF returnedSynset is not None THEN
10             RETURN returnedSynset
11         END IF
12     END LOOP
13     RETURN mostLikelySynset(synsetList)
14 END FUNCTION
```

Listing 6: The disambiguate function

2.5.1 Context

The context used for disambiguation its taken from two sources, the STS and knowledge of noun-verb relationships (the latter is discussed in the Sanity Checking Section).

Initially, the context contained in the STS is used. Two algorithms for disambiguation using this context have been implemented, one using hyponyms and one using hypernyms.

As discussed in the Previous Works section, M. Burke found that hyponym based searching (previously called STM-First), using the Stm is the most effective method of disambiguation [18]. This method works upon the assumption that the contents of the Stm are relatively general (i.e. exist high up in the Wordnet hierarchy). The implementation of this search is given by the hyponymSearch function, shown in Listing 7.

```
1  FUNCTION hyponymSearch(synsetList, searchItem):
2      hyponymList = searchItem.hyponyms()
3      IF len(hyponymList) == 0 THEN
4          RETURN None
5      END IF
6      FOR item in hyponymList LOOP
7          IF item in synsetList THEN
8              RETURN item
9          END IF
10     END LOOP
11     FOR item in hyponymList LOOP
12         returnedItem = hyponymSearch(synsetList, item)
13         IF returnedItem is not None THEN
14             RETURN returnedItem
15         END IF
16     END LOOP
17     RETURN None
```

Listing 7: The hyponymSearch function

This function traverses through all hyponyms of a given synset (searchItem) until either, a match with a synset in the synsetList is found, or the function reaches the base of the tree. If a match is found, the matched synset is returned as the correct word meaning.

Another algorithm, shown in Listing 8, for disambiguation has also been implemented, using the lowest_common_hypernym method implemented in NLTK's WordNet reader [19]. This algorithm instead searches the hypernyms of both the word to disambiguate and the item in the STS, to find whether a common hypernym exists. If a common hypernym does exist, it is checked to ensure it is not too general (e.g. the common hypernym could be "entity" which is a hypernym of all nouns), and the synset is returned as the correct meaning.

```
1  FUNCTION hypernymSearch(synsetList, searchItem):
2      FOR synset in synsetList LOOP
3          common_hypernym = synset.lowest_common_hypernym(searchItem)
4          IF common_hypernym.depth > 4 THEN
5              RETURN synset
6          END IF
7          END LOOP
```

Listing 8: THE HYPERNYMSEARCH FUNCTION

Using either method, a meaning will only be found if the surrounding context of a word gives enough indication of its meaning, complying with the multiple access model, where if strong context exists, it is used for disambiguation [8].

2.5.2 Frequency

The previously described hyponym-based system will only find a meaning if a word's surrounding context is strong. When this is not the case, another system must be used. The multiple access model, described in the Disambiguation Models section, states that in such cases, the likelihood of a proposed word meaning is used [8].

In order to calculate the likelihood of each synset in the synsetList, Wordnet's lemmas are used. Lemmas in Wordnet are the word forms a synset can take. For each lemma, there exists a frequency value, which describes how common that form is, relative to other Wordnet lemmas. For each synset, the sum of all its lemmas' frequency values, is used to calculate the most likely synset, as shown in Listing 9.

```
1  FUNCTION synsetFrequency(synset):
2      outputFrequency = 0
3      FOR lemma in synset.lemmas() LOOP
4          outputFrequency += lemma.count()
5      END LOOP
6      RETURN outputFrequency
7  END FUNCTION
8
9  FUNCTION mostLikelySynset(synsetList):
10     outputSynset = synsetList[0]
11     FOR synset in synsetList LOOP
12         IF synsetFrequency(outputSynset) < synsetFrequency(synset) THEN
13             outputSynset = synset
14         END IF
15     END LOOP
16     RETURN outputSynset
17 END FUNCTION
```

Listing 9: The synsetFrequency and mostLikelySynset functions

As stated previously, in cases where context cannot provide a viable synset, the above is used for disambiguation.

2.5.3 Sanity Checking

Given the sentence:

He fixed the bug

The reader knows that "bug" refers to a computer bug, due to its use with the verb "fixed". So far, the system has no way of modelling the contextual effects of nouns on verbs, so the system is likely to produce invalid sentences. For this reason, post-disambiguation sanity checks occur.

Though a proposed feature, WordNet does not currently contain relationships between nouns and verbs [13]. For this reason, the information had to be extracted from the available corpus. It was decided that half of the test corpus would be used for the extraction, so that test data used for evaluation would be completely new to the system.

Listing 10 shows the algorithm used for extracting this information. Nouns which occur in the same sentence as a specific Verb are considered valid partners to the Verb, and are added to a dictionary for quick lookup.

```
1  FUNCTION verbDistance(verb, sentence):
2          FOR word in sentence LOOP
3                  IF word is a noun THEN
4                          outputList.append(word.synset)
5                  END IF
6          END LOOP
7          RETURN outputList
8  END FUNCTION
9
10 FOR sentence in inputCorpus LOOP
11         FOR every verb in sentence LOOP
12                 update contents of verbDict[verb.synset]
13                 to include verbDistance(verb, sentence)
14         END LOOP
15 END LOOP
```

Listing 10: The noun-verb relationship learning algorithm

Initially, the sentence is disambiguated, using the method described previously. The outputted sentence is then checked for correctness by the function sanityCheck, as shown in Listing 11. If a synset is found to be incompatible with others in the sentence, it is added to a blacklist, and its corresponding word is disambiguated again, ignoring the previous meaning.

```
1  FUNCTION sanityCheck(inputSentence, nounDict, verbDict):
2      sane = False
3      FOR word in inputSentence LOOP
4          IF word is a noun THEN
5              nounList.append(word)
6          ELSE IF word is a verb THEN
7              verbList.append(word)
8          END IF
9      END LOOP
10     WHILE not sane LOOP
```

```
11          sane = True
12          FOR verb in verbList LOOP
13              IF verb in verbDict THEN
14                  plausibleNouns =  verbDict[verb]
15                  IF nounList and plausibleNouns share common words THEN
16                      verbList.remove(verb)
17                  ELSE
18                      blackList.append(verb)
19                      verbList.remove(verb)
20                      Run disambiguation again with blacklist
21                  END IF
22              END  IF
23                  END LOOP
24          END LOOP
25  END FUNCTION
```

Listing 11: The sanityCheck function

It may be noted that sanityCheck only considers verb incorrectness. As stated in the WordNet section, for each word, there is likely to be a larger set of synsets when compared to nouns. With this knowledge, we can assume that the likelyhood of an incorrect result for a verb is higher than that of a noun.

References

[1] F. Damerau. N. Indurkhya, *Handbook of Natural Language Processing*, 2nd ed. Taylor and Francis Group, 2010.

[2] R. Collobert. et al., "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. 76, pp. 2493–2537, Aug 2011.

[3] C. Goddard, *Semantic Analysis: A Practical Introduction*, 2nd ed. Oxford University Press, 2011.

[4] M. Eysenck. A. Baddeley and M. Anderson, *Memory*, 2nd ed. Psychology Press, 2015.

[5] M. Quillain, "Semantic memory," *BBN Report*, vol. 675, no. 2, pp. 227–270, Oct 1966.

[6] G. Miller, "The magical number seven, plus or minus two some limits on our capacity for processing information," *Psycological Review*, vol. 101, no. 2, pp. 343–352, May 1955.

[7] R. Shiffrin. R. Atkinson, "The control processes of short term memory," *Technical Report*, vol. 173, Apr 1971.

[8] T. Harley, *The Psychology of Language: From Data to Theory*, 3rd ed. Psychology Press, 2008.

[9] D. Swinney. W. Onifer, "Accessing lexical ambiguities during sentence comprehension: Effects of frequency of meaning and contextual bias," *Memory & Cognition*, vol. 9, no. 3, pp. 225–236, May 1981.

[10] G.Miller. et al., "Introduction to wordnet: An on-line lexical database," *International Journal of Lexicography*, vol. 3, no. 4, pp. 235–244, Aug 1990.

[11] G.Miller, "Nouns in wordnet: A lexical inheritance system," *International Journal of Lexicography*, vol. 3, no. 4, pp. 235–244, Aug 1990.

[12] C. Fellbaum. et al., "Adjectives in wordnet," *International Journal of Lexicography*, vol. 3, no. 4, pp. 235–244, Aug 1990.

[13] C. Fellbaum, "English verbs as a semantic net," *International Journal of Lexicography*, vol. 3, no. 4, pp. 235–244, Aug 1990.

[14] S. Dumais. T. Landauer, "A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge," *Psychological Review*, vol. 104, no. 2, pp. 211–214, Apr 1997.

[15] J. Ramos, "Using tf-idf to determine word relevance in document queries," *Proceedings of the first instructional conference on machine learning*, vol. 1, Dec 2003.

[16] D. Kazakov. J. Sedding, "Wordnet-based text document clustering," *Proceedings of the 3rd Workshop on RObust Methods in Analysis of Natural Language Data*, vol. 3, pp. 104–113, Aug 2004.

[17] J. Weston. R.Collobert, "A unified architecture for natural language processing: deep neural networks with multitask learning," *Proceedings of the 25th international conference on Machine learning*, vol. 25, pp. 160–167, Jul 2008.

[18] M. Burke, "Contextual disambiguation using wordnet and a short-term memory model," BEng dissertation, Dept. of Computer Science, University of York, York, UK, 2007.

[19] E. Klein. S. Bird and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, 2009.