Summary of WordNet

WordNet is a lexical database that is an attempt to organize language into a hierarchy of words. WordNet provides short definitions and use examples. Furthermore, WordNet groups words into synonym sets known as synsets which we can use to note how similar or dissimilar various words are, among other things

```
#Formatting Code so that the output has text wrapping
from IPython.display import HTML, display

def set_css():
  display(HTML('''
  <style>
    pre {
        white-space: pre-wrap;
    }
  </style>
  '''))
get_ipython().events.register('pre_run_cell', set_css)
```

```
#Import Statements
%%capture
import math
import nltk
nltk.download("all")
from nltk.book import *
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
```

```
#All the synsets of the noun "book"
print("Synsets for the noun 'Book'")
print(wn.synsets('Book'))
```

```
Synsets for the noun 'Book'
[Synset('book.n.01'), Synset('book.n.02'), Synset('record.n.05'),
Synset('script.n.01'), Synset('ledger.n.01'), Synset('book.n.06'), Synset('book.n.07'),
Synset('koran.n.01'), Synset('bible.n.01'), Synset('book.n.10'), Synset('book.n.11')
```

```
#Select one synset from the list of synsets
bible = wn.synset('bible.n.01')
```

```
#Extract its definition, usage examples, and lemmas
print("Definition of the Bible:")
print(bible.definition().title())

print("\nExample of usage examples for the Bible:")
print(bible.examples())

print("\nLemmas for the Bible:")
print(bible.lemmas())
print()

#From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting t
print("Traversing up the WordNet hierarchy for the Bible:")
hyp = bible.hypernyms()[0]
top = wn.synset('entity.n.01')
while hyp:
    print(hyp)
    if hyp == top:
        break
    if hyp.hypernyms():
        hyp = hyp.hypernyms()[0]
```

```
Definition of the Bible:
The Sacred Writings Of The Christian Religions

Example of usage examples for the Bible:
['he went to carry the Word to the heathen']

Lemmas for the Bible:
[Lemma('bible.n.01.Bible'), Lemma('bible.n.01.Christian_Bible'),
Lemma('bible.n.01.Book'), Lemma('bible.n.01.Good_Book'),
Lemma('bible.n.01.Holy_Scripture'), Lemma('bible.n.01.Holy_Writ'),
Lemma('bible.n.01.Scripture'), Lemma('bible.n.01.Word_of_God'),
Lemma('bible.n.01.Word')]

Traversing up the WordNet hierarchy for the Bible:
Synset('sacred_text.n.01')
Synset('writing.n.02')
Synset('written_communication.n.01')
Synset('communication.n.02')
```

How WordNet is organized with nouns

All the nouns in WordNet have the ultimate hypernym of 'entity.n.01'. All nouns are hyponyms of this one synset. Therefore, WordNet nouns are organized into a hierarchy with 'entity.n.01' being on the very top of the hierarchy

```
#For the noun "Bible", we have the following:
#Hypernyms
print("Hypernyms for the Bible:")
print(bible.hypernyms())
```

```
#Hyponyms
print("\nHyponyms for the Bible:")
print(bible.hyponyms())

#Meronyms
print("\nMeronyms for the Bible:")
print(bible.part_meronyms())

#Holonyms
print("\nHolonyms for the Bible:")
print(bible.part_holonyms())

#Antonyms
print("\nAntonyms for the Bible:")
print(bible.lemmas()[0].antonyms())
```

```
Hypernyms for the Bible:
[Synset('sacred_text.n.01')]

Hyponyms for the Bible:
[Synset('family_bible.n.01')]

Meronyms for the Bible:
[Synset('new_testament.n.01'), Synset('old_testament.n.01'),
Synset('testament.n.04'), Synset('text.n.02')]

Holonyms for the Bible:
[]

Antonyms for the Bible:
[]
```

```
#All the synsets for the verb "Believe"
print("Synsets for the verb 'Believe'")
print(wn.synsets('believe'))
```

```
Synsets for the verb 'Believe'
[Synset('believe.v.01'), Synset('think.v.01'), Synset('believe.v.03'),
Synset('believe.v.04'), Synset('believe.v.05')]
```

```
#Select one synset from the list of synsets

believe = wn.synset('believe.v.01')

#Extract its definition, usage examples, and lemmas
print("Definition of the verb 'believe':\n")
print(believe.definition().title())

print("\nExample of usage examples for the verb 'believe':")
print(believe.examples())

print("\nLemmas for the verb 'believe':")
```

```
print(believe.lemmas())
print()

#From your selected synset, traverse up the WordNet hierarchy as far as you can, outputting t
print("Traversing up the WordNet hierarchy for 'believe':")
currentSynset = believe
while True:
  print(currentSynset)
  if currentSynset.hypernyms():
        currentSynset = currentSynset.hypernyms()[0]
  else:
    break
```

```
Definition of the verb 'believe':

Accept As True; Take To Be True

Example of usage examples for the verb 'believe':
['I believed his report', "We didn't believe his stories from the War", 'She
believes in spirits']

Lemmas for the verb 'believe':
[Lemma('believe.v.01.believe')]

Traversing up the WordNet hierarchy for 'believe':
Synset('believe.v.01')
Synset('accept.v.01')
Synset('evaluate.v.02')
Synset('think.v.03')
```

How WordNet is organized for verbs

Unlike nouns, WordNet verbs do not have an ultimate hypernym. As such, while various verbs may be grouped into various hierarchies, there is no one single hierarchy that unites them all. In other words, there is no verb that plays the role that 'entity.n.01' plays for nouns

```
#Using morphy to find as many different forms of the word "Believe" as I can
word = 'believe'
print("Using morphy to find various forms of the word '" + word + "':")
print(wn.morphy(word))
print(wn.morphy(word, wn.NOUN))
print(wn.morphy(word, wn.ADJ))
print(wn.morphy(word, wn.ADJ_SAT))
print(wn.morphy(word, wn.VERB))
```

```
Using morphy to find various forms of the word 'believe':
```

```
# "Media" and "Entertainment" are similar words
# Finding the specific synsets for each word
print("Synsets for 'TV':")
print(wn.synsets('TV'))
print()

print("Synsets for 'news':")
print(wn.synsets("News"))
```

```
Synsets for 'TV':
[Synset('television.n.01'), Synset('television_receiver.n.01')]

Synsets for 'news':
[Synset('news.n.01'), Synset('news.n.02'), Synset('news_program.n.01'),
Synset('news.n.04'), Synset('newsworthiness.n.01')]
```

```
# Runing the Wu-Palmer similarity metric and the Lesk algorithm on "television.n.01" and "new
tv = wn.synset("television.n.01")
news = wn.synset("news.n.01")

print("Comparing 'TV' and 'News' using the Wu-Palmer Similarity Metric:")
print(wn.wup_similarity(tv, news))
print()

print("Using the Lesk algorithm to disambiguate the meaning of 'TV' in the sentence 'I am goi
watchingTVSentence = ['I', 'am', 'going', 'to', 'watch', 'some', 'TV']
print(lesk(watchingTVSentence, 'TV'))
print()

print("Using the Lesk algorithm to disambiguate the meaning of 'news' in the sentence 'I am g
watchingNewsSentence = ['I', 'am', 'going', 'to', 'watch', 'the', 'news']
print(lesk(watchingNewsSentence, 'news'))
```

```
Comparing 'TV' and 'News' using the Wu-Palmer Similarity Metric:
0.125

Using the Lesk algorithm to disambiguate the meaning of 'TV' in the sentence 'I
am going to watch some TV':
Synset('television_receiver.n.01')

Using the Lesk algorithm to disambiguate the meaning of 'news' in the sentence 'I
am going to watch the news':
```

Some observations about the Wu-Palmer similarity metric and the Lesk algorithm

Wu-Palmer Similarity Metric

I'm quite surprised that 'TV' and 'News' only had a 0.125 similarity. In my mind, they seem a lot more correlated

Lesk Algorithm

In both cases, the Lesk algorithm failed to disambiguate the words correctly.

For 'TV', the expected output was 'television.n.01' but the Lesk algorithm returned 'television_receiver.n.01'

For 'News', the expected output was 'news.n.01' but the Lesk algorithm returned 'newsworthiness.n.01'

Functionality and possible use cases for SentiWordNet

SentiWordNet is built on top of WordNet and is used to analyze the sentiment of a word/sentence. This is accomplished by giving each word a positive/negative/objective score. Some possible use cases could be seeing if a tweet is toxic, checking how positive and/or negative a news article about a certain person is, and seeing if an email you wrote comes off as positive or negative

```
# "Death" is an emotionally charged word
# Finding its senti-synsets and output the polarity scores for each word
death = swn.senti_synset('death.n.03')
print(death)
print("Positive score = ", death.pos_score())
print("Negative score = ", death.neg_score())
print("Objective score = ", death.obj_score())
print()

# Sentence: "For God so loved the world that He gave His only begotten Son that whosoever bel
# Outputting the polarity for each word in the sentence
john316 = "For God so loved the world that He gave His only begotten Son that whosoever belie
print("Outputting the polarity for each word in the sentence: '" + john316 + "'")
neg = 0
pos = 0
tokens = john316.split()
for token in tokens:
    syn_list = list(swn.senti_synsets(token))
    if syn_list:
        word = syn_list[0]
        neg += word.neg_score()
        pos += word.pos_score()
        print("For the word '" + token + "' we have the following polarity:")
        print("\t" + str(word))
        print("\tPositive score = ", word.pos_score())
```

```
            print("\tNegative score = ", word.neg_score())
            print("\tObjective score = ", word.obj_score())
            print()

    print("Overall Negative and Positive Score for the sentence: '" + john316 + "'")
    print("neg\tpos")
    print(neg, '\t', pos)
```

```
        Objective score =  1.0

For the word 'Son' we have the following polarity:
        <son.n.01: PosScore=0.0 NegScore=0.0>
        Positive score =  0.0
        Negative score =  0.0
        Objective score =  1.0

For the word 'believes' we have the following polarity:
        <belief.n.01: PosScore=0.125 NegScore=0.0>
        Positive score =  0.125
        Negative score =  0.0
        Objective score =  0.875

For the word 'on' we have the following polarity:
        <on.a.01: PosScore=0.0 NegScore=0.0>
        Positive score =  0.0
        Negative score =  0.0
        Objective score =  1.0

For the word 'not' we have the following polarity:
        <not.r.01: PosScore=0.0 NegScore=0.625>
        Positive score =  0.0
        Negative score =  0.625
        Objective score =  0.375

For the word 'perish' we have the following polarity:
        <die.v.01: PosScore=0.0 NegScore=0.0>
        Positive score =  0.0
        Negative score =  0.0
        Objective score =  1.0

For the word 'but' we have the following polarity:
        <merely.r.01: PosScore=0.0 NegScore=0.0>
        Positive score =  0.0
        Negative score =  0.0
        Objective score =  1.0

For the word 'have' we have the following polarity:
        <rich_person.n.01: PosScore=0.0 NegScore=0.0>
        Positive score =  0.0
        Negative score =  0.0
        Objective score =  1.0

For the word 'everlasting' we have the following polarity:
        <everlasting.n.01: PosScore=0.0 NegScore=0.375>
        Positive score =  0.0
        Negative score =  0.375
        Objective score =  0.625

For the word 'life' we have the following polarity:
        <life.n.01: PosScore=0.0 NegScore=0.0>
        Positive score =  0.0
        Negative score =  0.0
        Objective score =  1.0

Overall Negative and Positive Score for the sentence: 'For God so loved the
```

```
Overall Negative and Positive Score for the sentence:  For God so loved the
world that He gave His only begotten Son that whosoever believes on Him should
```

Observations of the scores and the utility of knowing these scores in an NLP application

In terms of observations, I was really shocked and somewhat confused. John 3:16 was analyzed and it was calculated that it was more negative than positive! And individual words were interesting as well like how 'God' and 'life' have a 0 positive rating and that 'everlasting' actually has a somewhat negative score

I believe a utility of knowing these scores in an NLP application could be in training AIs to understand how language can communicate positive/negative emotions and thereby help the AIs to write more realistically

What a collocation is

Language isn't a random grouping of various words. There is a structure and ordering and grouping to words and phrases and sentences. As such, there are certain pairs or sets of words that appear next to each other more often than not. For example, 'good' and 'enough' are commonly used together as 'good enough'. These pairs or sets of words is what is known as a collocation

```
# Output collocations for text4, the Inaugural corpus
print("Collocations for the Inaugural Corpus:")
print(text4.collocations())
print()

text = ' '.join(text4.tokens)
vocab = len(set(text4))

#Mutual Information for 'Almighty God'
print("Mutual Information for the phrase 'Almighty God':")
almightyGod = text.count('Almighty God')/vocab
print("p(Almighty God) =", almightyGod)

almighty = text.count('Almighty')/vocab
print("p(Almighty) =", almighty)
```

```
God = text.count('God')/vocab
print('p(God) =', God)

pmi = math.log2(almightyGod / (almighty * God))
print('pmi =', pmi)

#Mutual Information for 'of the'
print("\nMutual Information for the phrase 'of the':")
ofThe = text.count('of the')/vocab
print("p(of the) =", ofThe)
of = text.count('of')/vocab
print("p(of) =", of)
the = text.count('the ')/vocab # space so it doesn't capture 'their' etc.
print('p(the) =', the)
pmi = math.log2(ofThe / (of * the))
print('pmi =', pmi)
```

```
Collocations for the Inaugural Corpus:
United States; fellow citizens; years ago; four years; Federal
Government; General Government; American people; Vice President; God
bless; Chief Justice; one another; fellow Americans; Old World;
Almighty God; Fellow citizens; Chief Magistrate; every citizen; Indian
tribes; public debt; foreign nations
None

Mutual Information for the phrase 'Almighty God':
p(Almighty God) = 0.0014962593516209476
p(Almighty) = 0.002793017456359102
p(God) = 0.011172069825436408
pmi = 5.583495367722955

Mutual Information for the phrase 'of the':
p(of the) = 0.20089775561097256
p(of) = 0.7487281795511221
p(the) = 0.9533167082294264
pmi = -1.8290080938996587
```

Write commentary on the results of the mutual information formula and your interpretation

The PMI for 'Almighty God' is 5.58

The PMI for 'of the' is -1.83

The phrase 'Almighty God' is far more likely to be a collocation than the phrase 'of the', at least with respect to the Inaugural Address Corpus