Ideas and notes:

# 26/07

* The first part of an encyclopedia article could constitute a more informative, less strict definition. Or a context to exploit in Paragraph2Vec manner.  
  We can use DBpedia’s introductory paragraph (is it also *rdfs:comment* ?), and possibly also the *dbo:abstract* (English) property to retrieve the equivalent.
* One of the problems of GNNs is that you can not easily stack many layers, because you end up introducing noise from irrelevant distant nodes.  
  Idea: use a GRU-GNN, and expose all the layers, learning a combination of the output of all the layers, not just the last one. (as done in ELMo)
* Autoencoder: it seems to have no place in the architecture at the moment.  
  (Unless we encode the definition, and then recreate it)
* Generally, good non-Wikipedia dictionaries have an API with a price.

For easy reproducibility, the dictionary data base should be free.

* Could it be good to HTML-parse from thefreedictionary.com, that reunites several sources? No, too much effort for too little comparative gain.  
  They do not have an API due to licensing issues.
* More opportune to use the APIs and the link connections between:
  + BabelNet
  + (and/or) WordNet
  + DBpedia
  + Wiktionary
  + OmegaWiki

¨

## Analysis of BabelNet:

Example currently examined: ‘sunlight’, for simplicity. Including some observations from the more complex ‘plant’, as well.

12 results, 4 concepts.

However, 2 of these concepts are “fake concepts” and reasonably they should be named entities.

* **{sunlight • sunshine • sun}** The rays of the sun
* **{light• visible light • visible radiation • sunlight}** (physics) electromagnetic radiation that can produce a visual sensation
* **{Sunlight (Benson)}** Sunlight is an oil painting by Frank Weston Benson currently in the permanent collection at the Indianapolis Museum of Art.
* **{Sunlight (DJ Sammy song)}**

How to differentiate them?

We could look at the number of semantic connections: 1490, 2244, 147, 40. …

*However*, considering the ‘plant’ example, one of the senses ({plant}: An actor situated in the audience whose acting is rehearsed but seems spontaneous to the audience) has only 9 connections while still being relevant.

Alternatively, although it may be more difficult to implement in practice, some categories could be included/excluded.

For instance, “Categories: 2003 singles, DJ Sammy songs” could be excluded, whereas “Atmospheric radiation, Climate forcing, IARC Group 1 carcinogens, Light sources, Solar energy, Sun” would be included…

Let us dive now into the 1st synset : **{sunlight • sunshine • sun}** The rays of the sun

1) Definitions from: WordNet, Wikipedia, Wikidata, Wiktionary, OmegaWiki

In this particular example, all the definitions are relevant.

Question: Are they identical, or do we need to separately access Wiktionary/OmegaWiki to get better/more definitions?

There is a difference in focus: the BabelNet quote is meant for that particular sense of the synset, whereas the Wiktionary entry includes all (e.g. also “(figuratively) Brightness, hope; a positive outlook.”)

And in the cited example, that figurative sense is not represented at all in BabelNet, and not even in OmegaWiki either.

Therefore, yes, it is worth it to separately access Wiktionary. To avoid eccessive overlaps, it is opportune to extract only WordNet and Wikipedia definitions from BabelNet.

2) Several examples. Examples can be useful as a more focused, restricted context. (in the context co-occurrences source for the GNN, they can have a greater weight).

However, the examples here can be about any term of the synset.

So either: filter out all the synset words (“We were warmed by the bright [sunshine].”, “[Sunlight] on the skin gives you vitamin D”), or ignore this entirely and search for examples elsewhere.

3) Semantic relations. Currently not used in this task.

4) Images. “”

5) Translations. “”

6) Sources

- WordNet senses: (for each element of the synset). In the current version, it can be considered redundant. We have the definition and the synset grouping already.

It could be useful to get a link to WordNet’s semantic relations for the specific target word.

- Wikipedia page: the first paragraph is useful as an encyclopedia-style extended definitions. However, it coincides with DBpedia’s introduction.

- OmegaWiki senses: useful. We can get a direct connection on the specific sense of the target word (the others in the synset can be ignored)

- WikiData: useless

7) Categories

8)Compounds: maybe they could be used to locate phrases? (e.g.: collecting sunlight, intense sunlight, Sunlight Solar energy, sensitivity to sunlight)

However: Compounds for ‘plant’ is too long a list, with 30+ phrases (“… plant engineering, flora of Japan, Nuclear power plants, aquatic plant, marine plants, flowering plants, list of plants.”)

It may have another use, instead, not necessarily directly connected to the word embeddings: we may collect compound expressions to build an encyclopedia of phrases and idiomatic expressions

9) External Links

- DBpedia: Sunlight

Extremely useful, direct connection to target word in DBpedia. Once again, it appears that the most useful properties are *dbo:abstract* and *rdfs:comment*, that provide an extended definition.

Note: Are these 2 properties identical. Checking on ‘plant’…

It appears that the *rdfs:comment* is a core subset of the *dbo:abstract*

## Onwards

Observation: Generally, the definitions “from WordNet”*in the synsets that contain the word* constitute a superset of the definitions *for the target word* (e.g. “plant”) in WordNet.

Possibilities:

- Restrict: Examine only the BabelNet synsets which correspond (word-by-word) to a WordNet definition

- Widen: Examine all synsets, and include everything, thus pulling the WordNet definitions of words different from the target word as well

- Choose by rank: we include the synsets where the target word is 1st or 2nd. If it is 3rd or beyond, ignore.

Note: maybe a better WordNet – BabelNet correspondence is determined by whether the elements in the synset are 100% coinciding? They do not \*always\* coincide, so synset + definition is probably a better idea to get the correspondence…

It would appear that the “Choose by rank” policy is the most promising, as the most likely to include relevant additional senses (see for ‘plant’ without adding too much unimportant information / noise from other words (e.g. including ‘pan’ into ‘pot’, or ‘light’ into ‘sunlight’)

* We could also use different weights for the “rank/core” contributions…
* Or we could use a “restrict-WordNet” policy, while including the relevant (eg. Close, or key concepts) synsets among the synonims

# 27/07

Describing again the task:

A Graph NN can be applied over:

1. dictionary definition co-occurrence graph
2. context-window occurrences
3. (synonims & antonyms)

The aim is to combine the input of context co-occurrence methods and dictionary information

1)

BabelNet: Select the relevant synsets, and extract the WordNet and Wikipedia definitions

DBpedia: From BabelNet>External links, go to the DBpedia page and get *rdfs:comment*

Wiktionary: Go to the Wiktionary page of the target word, and either use the API or parse to get English > Noun / Verb > List of definitions

OmegaWiki: get the English definitions from the page/content of the target word, all of them.

## Selecting the relevant synsets from BabelNet

We review the examples ‘sea’, ‘plant’, ‘high’:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Synset for: Sea | Definition | Key | In WordNet | Connections | Relevant? |
| sea | (a division of an ocean or a large body of salt water partially enclosed by land) | Y | Y | 4.4K + | Y |
| ocean • sea | Anything apparently limitless in quantity or volume | N | Y | 5 | Y |
| sea | Turbulent water with swells of considerable size | N | Y | 7 | Y |
| ocean • body of water • bounding main • sea | A large body of water constituting a principal part of the hydrosphere | Y | C | 3.1K + | Y |
| Lunar mare • mare • lunar maria • sea | The lunar maria are large, dark, basaltic plains on Earth's Moon, formed by ancient volcanic eruptions. | N | C | 842 | C |
| Sea (advertisement) | Sea is an advertising campaign launched by Diageo in 2007 to promote Smirnoff brand vodka. | N | N | 76 | N |
| Seamester • mester Global Programs • Sea | Sea|mester Global Programs is an organization which offers academic, study abroad programs on board two sailing vessels, Ocean Star and Argo. | N | N | 43 | N |
| seah (unit) • Sea (unit) | The se'ah or seah is a unit of dry measure of ancient origin used in Halakha, which equals one third of an ephah, or bath | N | N | 42 | N |
| Sea | Genus of insects | N | N | 29 | N |
| sea | Heraldic figure | N | N | 13 | N |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Synset for: Plant(name) | Definition | Key | In WordNet | Connections | Relevant? |
| industrial plant • plant • works | Buildings for carrying on industrial labor | Y | Y | 66 | Y |
| flora • plant • plant life | (botany) a living organism lacking the power of locomotion | Y | Y | 4.2K + | Y |
| plant | An actor situated in the audience whose acting is rehearsed but seems spontaneous to the audience | N | Y | 9 | C |
| plant | Something planted secretly for discovery by another | N | Y | 4 | C |
| factory • manufactory • manufacturing plant • plant | A plant consisting of one or more buildings with facilities for manufacturing | Y | C | 1.6K + | C |
| assembly line • production line • line • plant | Mechanical system in a factory whereby an article is conveyed through sites at which successive operations are performed on it | N | C | 428 | N |
| factory (trading post) • factorij • manufactory • plant | "Factory" was the common name during the medieval and early modern eras for an entrepôt – which was essentially an early form of free-trade zone or transshipment point. | N | N | 392 | N |
| Glossary of cue sports terms • Plant (snooker) | The following is a glossary of traditional English-language terms used in the three overarching cue sports disciplines: carom billiards referring to the various carom games played on a billiard table without pockets; pool, which denotes a host of games played on a table with six pockets; and snooker, played on a large pocket table, and which has a … | N | C | 386 | N |
| shill • Plant (person) | A decoy who acts as an enthusiastic customer in order to stimulate the participation of others | N | C | 367 | C |
| Glossary of professional wrestling terms • Plant (professional wrestling) | Professional wrestling has accrued a considerable nomenclature throughout its existence. | N | N | 332 | N |
| Creature type (Dungeons & Dragons) • Plant (Dungeons & Dragons) | … | N | N | 320 | N |
| PLANT | Corporation | N | N | 133 | N |
| Control System • Plant (control theory) | A plant in control theory is the combination of process and actuator. | N | C |  | N |

Note: Verbs and names can be recognized by the end of their ID:

Example: bn:00046568**n** versus bn:00091692**v**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Synset for: Plant (verb) | Definition | Key | In WordNet | Connections | Relevant? |
| set • plant | Put or set (seeds, seedlings, or plants) into the ground | N | Y | 18 | Y |
| implant • embed • engraft • plant | Fix or set securely or deeply | N | Y | 9 | Y |
| found • constitute • establish • plant | Set up or lay the groundwork for | N | Y | 8 | Y |
| plant | Place into a river | N | Y | 5 | C |
| plant | Place something or someone in a certain position in order to secretly observe or deceive | N | Y | 8 | Y |
| implant • plant | Put firmly in the mind | N | Y | 4 | Y |
| set up • establish • found • plant | Set up or found | N | N | 6 | C |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Synset for: high (noun) | Definition | Key | In WordNet | Connections | Relevant? |
| high | A lofty level or position or degree | N | Y | 5 | C |
| high • high-pressure area | An air mass of higher than normal pressure | N | Y | 174 | Y |
| high | A state of sustained elation | N | Y | 6 | Y |
| high | A state of altered consciousness induced by alcohol or narcotics | N | Y | 8 | Y |
| heights • high | A high place | N | Y | 3 | Y |
| high school • senior high school • high | A public secondary school usually including grades 9 through 12 | Y | Y | 23.1K + | Y |
| high • high gear | A forward gear with a gear ratio that gives the greatest vehicle velocity for a given engine speed­­ | N | Y | 10 | C |
| Top (technical analysis) • High (technical analysis) | In technical analysis, a top is an event in which a security's market price reaches a high, then a higher high, and then a lower high | N | N | 83 | C |
| Ledisi discography • High (Ledisi song) | This article contains the discography of American soul and R&B singer-songwriter Ledisi. | N | N | 54 | N |
| High (Young Rising Sons song) | "High" is a song recorded by New Jersey band Young Rising Sons, released as the band's debut single on July 22, 2014. | N | N | 21 | N |
| High (Royal Headache album) | High is the second studio album by Australian punk rock band Royal Headache, released on 21 August 2015 by What's Your Rupture?. | N | N | 17 | N |
| High (tectonics) | A high in structural geology and tectonics an area where tectonic uplift has taken place relative to its surroundings. | N | N | 9 | Y |
| High (computability) | In computability theory, a Turing degree is high if it is computable in 0′, and the Turing jump is 0′′, which is the greatest possible degree in terms of Turing reducibility for the jump of a set which is computable in 0′. | N | N | 7 | C |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Synset for: high (verb)  *(Not in WordNet)* | Definition | Key | In WordNet | Connections | Relevant? |
| high | To hie; to hasten. | N | N | 0 | N |
| high | To rise. | N | N | 0 | N |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Synset for: high (adverb) | Definition | Key | In WordNet | Connections | Relevant? |
| high • high up | At a great altitude | N | Y | 1 | Y |
| high | In or to a high position, amount, or degree | N | Y | 3 | Y |
| luxuriously • high • richly | In a rich manner | N | Y | 3 | Y |
| high | Far up toward the source | N | Y | 0 | Y |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Synset for: high (adjective) | Definition | Key | In WordNet | Connections | Relevant? |
| high | Greater than normal in degree or intensity or amount | N | Y | 15 | Y |
| high | (literal meaning) being at or having a relatively great or specific elevation or upward extension (sometimes used in combinations like `knee-high') | N | Y | 20 | Y |
| eminent • high | Standing above others in quality or position | N | Y | 4 | Y |
| high-pitched • high | Used of sounds and voices; high in pitch or frequency | N | Y | 18 | Y |
| high • in high spirits | Happy and excited and energetic | N | Y | 4 | Y |
| gamey • gamy • high | (used of the smell of meat) smelling spoiled or tainted | N | Y | 3 | Y |
| high • mellow | Slightly and pleasantly intoxicated from alcohol or a drug (especially marijuana) | N | Y | 10 | Y |

# 29/07

Observations:

* Not all synsets have a DBpedia entry. For instance, there are 4 synsets corresponding to WordNet senses for ‘plant’, and only the 2 main senses have External Links > DBpedia.
* Choice: due to the presence of other independent sources (Wiktionary, OmegaWiki), and due to the later insertion of synonims and antonyms, we decide to use a **Restrict** policy: we include the synsets of nouns (and also verbs, adjectives and adverbs) that have a direct correspondence with a sense of the target word in WordNet.
* About the subsequent Synonyms&Antonyms step: considering the WordNet synsets *and* the key concepts, all the other words in the sets will be counted as synonyms.

Operating with the nltk interface for WordNet:

We start from a target word, e.g. ‘plant’

We retrieve all the synsets for it.

[Synset('plant.n.01'), Synset('plant.n.02'), Synset('plant.n.03'), Synset('plant.n.04'), Synset('plant.v.01'), Synset('implant.v.01'), Synset('establish.v.02'), Synset('plant.v.04'), Synset('plant.v.05'), Synset('plant.v.06')]

Remember that POS-tagging and the different roles and meanings of a word are not addressed in this task. The purpose is to obtain graph-based, dictionary-enhanced word embeddings, not multi-sense.

Therefore, we simply collect all the definitions, without having POS-based differences.

(Just use synset.definition() on all)

# 31/07

Working with the HTTP API for BabelNet.

Given a target word, we can:

- select its synsets

- exclude those where the synsetType is NAMED\_ENTITIES instead of CONCEPTS

[problem: How do we deal with words like: “New York”, “Copenhagen”, “London”?

We have to rely on the other sources: WordNet, DBpedia and Wiktionary]

- *Restrict to WordNet* policy: in the list of senses:

- if there isn’t any WordNetSense, drop

- go into properties > simpleLemma. If the original target word is not contained in any of the lemmas, then drop. If it is found, keep.

- collect the definitions, through: glosses > [one of the dicts in the list] gloss , where the source must be either WN or WIKI

Observations: As it is, the definitions pulled from the BabelNet synsets are a superset of the Wordnet ones, because they also include related synsets.

While it is true that all accepted entries must have a correspondence in WordNet (although not necessarily directly with the target word), maybe it would be appropriate to move some of the related entries to the synonyms?

DBpedia access: we pull the encyclopedia definitions of the target words (written in CamelCase), proceeding as follows:

**PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>  
PREFIX dbres: <http://dbpedia.org/resource/>  
PREFIX dbo: <http://dbpedia.org/ontology/>  
  
SELECT DISTINCT ?encyclopedia\_def  
WHERE { dbres:"""** + target\_word + \  
**""" rdfs:comment ?encyclopedia\_def   
 FILTER (LANG(?encyclopedia\_def)='en')}**

*rdfs:comment* provides 3-4/7-8 rows. *dbo:abstract* has a more extensive definition, but I consider it too extensive, especially if compared with the other sources

# 01/08

To extract information from Wiktionary, we can use the wiktionaryparser Python package. Here the target word should be in lowercase, not CamelCase.

The structure returned is:

[ #list containing 1 dictionary

‘etimology’: {str} “From Middle English plante, from Old English plante (“young tree or shrub, herb newly planted”), …”

**‘definitions’**: {list} [

{dict} {‘partOfSpeech’:”noun”,

**‘text’**: [*'plant (plural plants)'*, '

(botany) An organism that is not an animal, especially an organism capable of photosynthesis. Typically a small or herbaceous organism of this kind, rather than a tree.',

'(botany) An organism of the kingdom Plantae; …’,…] }

'relatedWords': …

**'examples'**: [ …

{dict} {

'partOfSpeech' = {str} 'verb'

**'text'** = {list} [

*'plant (third-person singular simple present plants, present participle planting, simple past and past participle planted)'*,

…

'relatedWords' = {list} [{'relationshipType': 'related terms', 'words': ['plantation']},…]

**'examples'** = {list} ["…

}

OmegaWiki already provides an API. We navigate the structure returned by the request:

'http://www.omegawiki.org/api.php?action=ow\_express&search='

+ target\_word +'&format=json'

# 02/08

I decided to use the HDF5 data format to store the definitions, since it is a binary format known for its scalability and speed, and it can be used to store tables and organize them in groups/subfolders.

The pandas library provides an interface.

Note: given the current retrieval mechanisms, some of the definitions retrieved from BabelNet-WN are going to coincide with the ones from WordNet.

Handling this aspect also depends on the organization of the definition data.

Organization, prelude:

“The ordinary HDF5 library apparently does not even support concurrent reading of different files by multiple threads…”

On the matter,

“Starting with version 2.5.0, h5py includes support for the HDF5 SWMR features.

The SWMR features allow simple concurrent reading of a HDF5 file while it is being written from another **process**.”

Taking into account that every read/write must be sequential anyway, we have 3 alternatives:

1. Put all the definitions, from all the sources, into one file.  
   n: We can still use different threads to send the web requests, gather the data from them, dump.
2. Use several files, one per source (Wordnet (Wikipedia+WN), BabelNet, Wiktionary, DBpedia, ΩWiki)
3. Use several files, one per each chunk of the vocabulary (e.g. 5000 words)

Moreover: how to deal with the partial overlap BabelNet-WN & WordNet?

The frequency of a word in a set of definitions is an important piece of information: it will add a weight to the edge in the graph.

It is opportune to include only one copy of a definition from WordNet. Policy: *exclude* all WordNet definitions from the BabelNet input, keep only those from Wikipedia. This has the added advantage of not focusing on the synonyms’ definitions.

The workflow becomes the following:

* Use different threads to send the web requests
* Gather the data from them,
* Write all the definitions for that word into a file. We may use either one HDF5 file, or several (one for each vocabulary chunk) depending on the size.

# 03/08, 04/08

Immediately before setting up the definition retrieval, it is necessary to examine:

Which elements are we going to extract from each source?

* WordNet:
  + Definitions
  + Examples
  + Synonyms: from the lemmas in the same synset
  + Antonyms: in adjectives, that have the Antonym semantic relation.
* BabelNet:
  + Definitions: only those from Wikipedia  
    among the other sources: WordNet, Wiktionary and OmegaWiki are already collected separately. Wikidata, Wikiquote, and FrameNet are not as relevant and/or reliable
  + Examples
  + Synonyms: from the lemmas in the selected synsets
* Wiktionary:
  + Definitions
  + Examples
  + Synonyms
  + Antonyms
* OmegaWiki:
  + Definitions
  + Synonyms (provided for each meaning)
* DBpedia
  + Encyclopedia definition

Emerging questions:

1. Multi-word expressions.

For instance, the list of synonyms for the term ‘Sunlight’ from BabelNet is:

['sunlight', 'sunshine', 'sun', 'solar\_radiation', 'Natural\_lighting', 'Solar\_Irradiation', 'Solar\_irradiation', 'Solar\_output', 'Solar\_Radiation', 'Solar\_spectrum', …]

And the list of synonyms for ‘sunlight’ in OmegaWiki is:

['solar radiation', 'sunlight', 'sunshine']

In the current version, we take the simplifying choice and eliminate all multi-word synonyms (i.e. with an underscore or with a space). We can do this after we have retrieved them, when it is the moment to store them.

1. Antonyms

What could be a good source for antonyms? Can get them from Wiktionary, it does have more than just definitions and examples.

# 05/08

I deem it useful to employ multiple threads, since we send the HTTP requests to different sources.  
(For the computation part, it is equivalent to operating sequentially, as Python threads keep a global lock. Multiple processes do not share the same memory, and dealing with IPM is not worth it).

Note: we need each thread to return values: the definitions/examples/synonyms/antonyms that have been extracted. We can use concurrent.futures.ThreadPoolExecutor …

For now: set up **sequential** processing and storage.

Using 1 HDF5 file for category (definitions, examples, synonyms, antonyms, encyclopedia\_defs)

Note: when storing examples, it is opportune to eliminate duplicates (since part of those in BabelNet are also present in WordNet and Wiktionary)

# 21/08

Let us check the results of the GetInputData phase, and examine the purposes of the different components, to determine what post-processing is necessary, and then which methods should be applied.

[Adjusting the language detection…

langid is better than py-cld2, given that we are operating on short text fragments…

Must eliminate starting and ending space from synonyms and antonyms – done

It is opportune to verify whether all the tokens of a definition/example/a synonym/whatever can be found in the vocabulary of the target language.

Nltk.corpus.words.words() provides one for English. What about other languages?

Other languages can wait. For now, the language detection system is:

* Ranked as one of the 5 most likely languages by the langid module
* If not, check if all the tokens can be found in the vocabulary of the language

Side benefit: emoticons and images, that may be supplied by BabelNet, are removed.]

[Must review the Wiktionary source…

Removed wrong examples-with-synonyms of the Form: (for ‘move’) “Synonym: to stir”

Check antonyms: adjusted error of always skipping the first 2 chars even when “ :” does not remain]

{Soon to add:

Could save the BabelNet synsets, in order to avoid having to send too many requests, and speed up the process of data collection}

The Plan:

* Dictionary definitions: use pre-trained word embeddings (e.g. BERT, or others) and then obtain sentence embeddings for the defs.
  + BERT (or another instrument, or a Word2Vec/GloVe bootstrap on definitions + encyclopedia defs. + examples) provides word embeddings.
  + Sentence embeddings could be obtained: - by averaging / by Doc2Vec
  + We obtain N definitions in the multidimensional space. Each one of them is connected to the core/target word *w* by a ‘definition’ edge.  
    (Alternative approach: get an average of the definition embeddings. The problem here is: excessive averaging versus fragmentation)
* Encyclopedia definitions: same as above
* Examples: they always contain the target word. We can obtain an embedding location for it by using the Skip-gram loss.
  + Examples consist of a great number of short text fragments.
  + We initialize randomly the example ‘entity’ for *w*
  + Then we apply Word2Vec-SkipGram over all the examples, with the aim of learning the target word’s embedding.  
    (this implies ignoring all examples that do not include the target word in the string. Note that inflections and verb forms (e.g. “moves”, “moved” should still be included)
* Synonyms and antonyms: are part of the graph, using the 'syn' and 'ant' edges.
  + Synonyms and antonyms estasblish a connection between the entities of target words.
  + It is necessary to adjust the loss function, to pull/push for synonyms/antonyms

# 22/08

Regarding baselines:

If we operate with a varying number of definitions, then graphSAGE is not an optimal choice: for the aggregator functions, it samples a fixed-size set of neighbors, to manage the amount of computation.

If we are more interested in considering all the connections, and evaluating which are the most important, then an Attention mechanism may be opportune.

Moreover – as claimed in the GraphAttentionNetworks paper by Velickovic 2018, graphSAGE achieved some of its better results when using a LSTM aggregator, which is sequential and thus needs to receive a random permutation of the neighbours (->impractical).

The remaining valid variant of graphSAGE would be the one using MaxPooling.

**DCGCNs** (Densely Connected Graph Convolutional Networks) must surely be included.

They have an advanced architecture, that combines attention (see GANs), spatial convolution (see graphSAGE), and layer aggregation (see Jump Knowledge Networks).

The code can be found at <https://github.com/Cartus/DCGCN>

I need to extract the Encoder half of the architecture, since the Decoder half employed for AMR and syntacticNMT is useless here.

In addition to DCGCNs, it may be profitable to examine the simpler Graph Attention Networks; moreover, I could add a Gated Recurrent Unit (see ‘Graph-to-Sequence Learning using Gated GNNs’ by Beck et al – 2018) on the top of Graph Attention Networks

However, before constructing GNN architectures, it is necessary to create the embeddings of the different entities.

Starting with:

Examples:

* Given a target word *w*
* Initialize randomly a vector of *d* features (e.g. *d*=300, as a starting point)
* Make sure that the examples have been preprocessed, i.e. that all *stopwords* have been removed.
  + nltk.corpus.stopwords.words(extended\_lang\_id) can be the first source. Others may be collected later
  + Note: are we sure that eliminating stopwords is the way to go?  
    It certainly allows to focus on meaningful words, but we lose the syntax  
    Only an ablation study can give the answer – but eliminating stopwords is the most likely choice when using Word2Vec  
    e.g.. The cabin is not wide from the lake 🡪 ['cabin', 'wide', 'lake']
  + {note: Wiktionary examples provide duplicates. Must go fix them…}
  + HDF5 can not store mixed object types: must rejoin the tokens into a string, separated by whitespace

# 23/08

On examples:

We must obtain the Skip-gram embedding in the multidimensional space for the target word.  
We can operate either on bootstrapped word embeddings, or instead on pre-trained word embeddings.

How to load the BERT pre-trained embeddings?

It seems opportune to refer to the Base version of the model, not the Large one. Hidden dimension *d* = 768.

(From <https://github.com/google-research/bert> ):

**“Fine-tuning with BERT**

Important: All results on the paper were fine-tuned on a single Cloud TPU, which has 64GB of RAM. It is currently not possible to re-produce most of the BERT-Large results on the paper using a GPU with 12GB - 16GB of RAM, because the maximum batch size that can fit in memory is too small …”

“**Using BERT to extract fixed feature vectors (like ELMo)**

In certain cases, rather than fine-tuning the entire pre-trained model end-to-end, it can be beneficial to obtained pre-trained contextual embeddings, which are fixed contextual representations of each input token generated from the hidden layers of the pre-trained model. This should also mitigate most of the out-of-memory issues.

As an example, we include the script extract\_features.py …”

Note:

Embeddings from BERT are created looking at the entire input sentence. Thus, we do not have a 1-to-1 correspondence word-to-embedding, but a word can be represented with different embeddings depending on the context.

{From <https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/> :

“It is worth noting that word-level similarity comparisons are not appropriate with BERT embeddings because these embeddings are contextually dependent, meaning that the word vector changes depending on the sentence it appears in.

This allows wonderful things like polysemy so that e.g. your representation encodes river “bank” and not a financial institution “bank”, but makes direct word-to-word similarity comparisons less valuable.

However, for *sentence embeddings* similarity comparison is still valid…

And … many similarity metrics make assumptions about the vector space (equally-weighted dimensions, for example) that do not hold for our 768-dimensional vector space.”}

Which brings forward another question:

Given that our instrument, based on dictionary resources, has the purpose of being

(a) a retrofitting method

(b) with the bootstrap variant, a way of creating word embeddings outright

Which evaluation measure to use?

{Examining the evaluation measures of several works, including those that employ dictionary resources…}

# 24/08

It would appear that:

Both BERT and ELMo embeddings are contextual. A word is represented by different vectors depending on the surrounding context. Their performance is evaluated on downstream tasks, not on Word Similarity benchmarks.

GPT uses a standard Language Model with a stacked Transformer-decoder. However, its embeddings are based on the BPE – Byte Pair Encoding, that merges subword units.

I need more information on whether/how to include pre-trained embeddings from one of the major Transfer Learning models.

In the meantime, it is opportune to get “classic” word embeddings, from either Google-Word2Vec or GloVe (or both)

Regarding the evaluation measures:

It is opportune to have **2 roads** for the evaluation of our word embeddings:

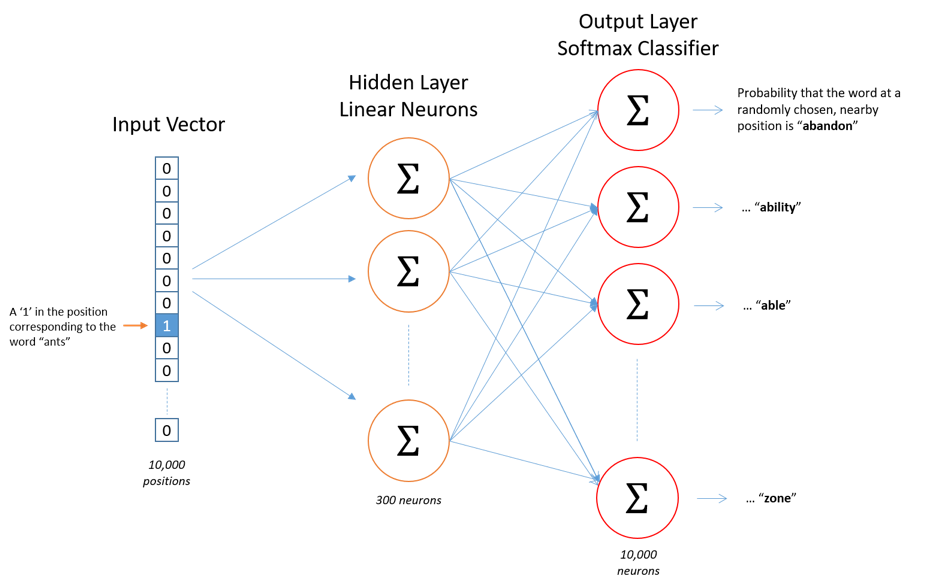
1. Word Similarity measures. They will be compared with the baselines such as Word2Vec, and also with the other works that attempted to use dictionary resources as a means of refinement/retrofitting.
   1. Measures: mainly Spearman’s correlation
   2. Benchmarks: SimVerb-3500, MEN, WordSim-353, etc.
   3. Baselines:   
      “Auto-Encoding Dictionary Definitions into Consistent Word Embeddings”,  
      “Dict2vec”, and possibly others.  
      Moreover, keep an eye out for the official version of <https://github.com/kudkudak/word-embeddings-benchmarks>, incoming in a few months.
2. Downstream tasks.  
   Currently, there is one downstream task of interest: Language Modeling.  
   So, perplexity on Wikitext-103, etc.  
   Emerging problem: we need to choose an architecture in which we can “plug in” the pre-made word embeddings.

Pre-trained Word2Vec embeddings trained on Google News, with #Vocabulary = 3M words, loaded.

We continue were we left off:

Examples:

* Given a target word *w*
* Initialize randomly a vector of *d* features (e.g. *d*=300, as a starting point)
* Make sure that the examples have been preprocessed, i.e. that all *stopwords* (and punctuation) have been removed.
* Implement the Skip-Gram loss, and train the vector *w* over the examples for the target word, by selecting a window in each example.



Situation:

If my intent is to randomly initialize the example\_entity\_vector of dimension *d*, then I should concatenate it to the Weights\_Embeddings matrix that goes from the input to the hidden layer.

Then, the one-hot encoding from the input will select that row alone as the hidden layer.

And the Skip-Gram will try to predict one of the words in the surrounding window, with the loss: softmax-with-negative-sampling (or noise-contrastive estimation).

Problem: From the pretrained Word2Vec, I can easily extract the first matrix of weights W\_E (the embeddings), but not the second matrix W\_prime that goes from the hidden layer to the output.

And only the latter is used in Skipgram.

Alternatively:

I could simply iterate, for each target word/example\_entity, using repeatedly the pre-trained version as a starting point:

Build a Skip-Gram graph, where the only element that is pre-set is the matrix W\_E.

Zero-out / random out the row of the target word *w* in the embeddings.

Train a SkipGram model over the corpus: examples for the word *w*. I can even proceed as I originally devised: train only on word-centered occurrences, and keep the other embeddings fixed.

# 25/08

Observation (from the genism site):

“It is impossible to continue training the vectors loaded from the C format   
because the hidden weights, vocabulary frequencies and the binary tree are missing.   
To continue training, you’ll need the full [**Word2Vec**](https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec) object state, as stored by [**save()**](https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec.save), not just the **[KeyedVectors](https://radimrehurek.com/gensim/models/keyedvectors.html" \l "gensim.models.keyedvectors.KeyedVectors" \o "gensim.models.keyedvectors.KeyedVectors)**.”

Therefore, we operate with the matrix W\_E and not W\_prime.

Note: the vocabulary of the GoogleNews-Word2Vec model has 3 million words.

Let us try to use it as-it-is. If the operations are too slow, then we may use a common subsample of it.

Note: possible instrument to examine the final location of an entity:

result = **word\_vectors.similar\_by\_word**("cat")

>>> print("{}: {:.4f}".format(\*result[0]))

dog: 0.8798

W\_E is not used in the computation graph.

I would lose 100% of the information from the pretrained embeddings.

Is there a way to retain this information?

Alternative:

Use the pretrained embeddings as a starting point for the W\_E matrix.

Update W\_E with a standard Skip-Gram model, sliding over all the examples (1 example == 1 sentence).

* Criticism towards the Vocabulary of the pretrained Word2Vec:  
  It is not case-insensitive, and it contains some stopwords but not others. (and mis-spellings, too)

model\_wv.index2word.index("The") Out[30]: 7

In[31]: model\_wv.index2word.index("the") Out[31]: 11

In[32]: model\_wv.index2word.index("of")

ValueError: 'of' is not in list

In[33]: model\_wv.index2word.index("and")

ValueError: 'and' is not in list

In[34]: model\_wv.index2word.index("above") Out[34]: 894

In[35]: model\_wv.index2word.index("Above") Out[35]: 14992

Hypothesis: maybe I do not need to build SkipGram myself.

I may be able to initialize a Word2Vec-model object with the pretrained model’s vocabulary and embeddings.

Negative: the vocab object inside the KeyedVectors contains only indices, there is no frequency information.

# 26/08

Building the NN graph for Skip-Gram…

ValueError: Cannot create a tensor proto whose content is larger than 2GB.

W\_E = tf.get\_variable(name=**"embeddings"**, initializer=embeddings\_atstart)

-> Must crop the vocabulary from GoogleNews-Word2Vec

From “Wiktionary-based Word Embeddings” (De Melo, 2015):

“trained on a Google News dataset consisting of about 100B word tokens using word2vec. The vocabulary size is 3,000,000.

However, out of these 3,000,000, actually 2,070,978 terms contain a space [i.e. underscore], most of which are multi-word expressions or named entities. Thus, the number of genuine words is much smaller”

pretrained\_model\_wv["Abraham\_Lincoln"][0:2]

Out[14]: array([0.55859375, 0.27929688], dtype=float32)’

* First possible modification to the vocabulary:  
  Lowercase everything. Join any identical words, such as ‘The’ and ‘the’, replacing them with the vector of the lowercase version.  
  pretrained\_model\_wv.similarity("the", "The")  
  Out[16]: 0.4476719
* Alternative (or also second possible modification):  
  Check the word frequencies on a reasonably-sized dataset (e.g. WikiText-103, 1BillionWords. Or Google-News itself? No, 100 billion words).   
  Keep in anything with count > 5 (minus stopwords).
* Alternative: create an entirely new vocabulary, possibly from definitions + examples.

# 27/08

Reconsidering the graph:

It may also be necessary to aggregate (or filter) the synonyms

(A filter may be based on the frequency in defs + examples)

The aggregation of examples has 2 directions:

* Bootstrap : Skip-Gram over the corpus of examples, then select *w* ‘s vector
* WRetrofitting on Pre-trained” version : Continuous-BagofWords, using pre-trained embeddings to build *w* ‘s vector, by applying CBOW over the corpus of examples.

Adjusting Bootstrap version. Need a better vocabulary…

# 29/08

Problem: using vocabulary = nltk.corpus.words.words() we do not include plurals & co.

ValueError: 'temperatures' is not in list

There are 3 approaches:

1. Using a vocabulary large enough that it contains all possible forms.
   1. Problem: we have seen that the Word2Vec-GoogleNews vocabulary, with its 3 million words and a tensor of > 2GB, is too large to handle
   2. Possible solution: create a vocabulary from WikiText-103 / 1BillionWords
2. Stemming all the words in the examples.
3. Ignoring the forms that we do not have in the vocabulary, letting them be <unk>

Regarding the nltk-corpus-words vocabulary: a lot of obscure words are in the corpus (abampere, academist, acceptant…) , while for instance ‘okay’ is not included

# 30/08

Currently trying to extract a vocabulary from WikiText-103

For the validation set:

Utils.count\_tokens\_in\_corpus(corpus\_txt\_filepath\_V)

Reading in line n. : 0 ; number of tokens encountered: 0

Reading in line n. : 1000 ; number of tokens encountered: 40627

Reading in line n. : 2000 ; number of tokens encountered: 95037

Reading in line n. : 3000 ; number of tokens encountered: 149689

Out[23]: 186302

From the statistics, we expected: 217,646

(We replace the punctuation)

If we don’t replace it, using nltk’s word\_tokenizer, we obtain: 222,377

For the test set:

Utils.count\_tokens\_in\_corpus(corpus\_txt\_filepath\_T)

Reading in line n. : 0 ; number of tokens encountered: 0

Reading in line n. : 1000 ; number of tokens encountered: 51195

Reading in line n. : 2000 ; number of tokens encountered: 99471

Reading in line n. : 3000 ; number of tokens encountered: 153499

Reading in line n. : 4000 ; number of tokens encountered: 194351

Out[25]: 209282

From the statistics, we expected: 245,569

If we don’t replace it, using nltk’s word\_tokenizer, we obtain: 252,416

For the Training set:

Reading in line n. : 1801000 ; number of tokens encountered: 87753527

87,766,328

If we do not replace the punctuation:

Reading in line n. : 1,801,000 ; number of tokens encountered: 104,735,405

104750753

Using Word2Vec’s build\_vocab is faster than manually appending to dictionary…

{Problem with the CorpusTokenizerIterator:  
on the training set,  
Tokenizing corpus and creating the vocabulary. Processing batch 63. Line: 258048. Token: 398,007,850}

But when reading from Utils, we got:   
Reading in line n. : 258000 ; number of tokens encountered: 12,448,412

From the iterator (4096 lines per batch):

Tokenizing corpus and creating the vocabulary. Processing batch 1. Line: 4096. Token: 198,376

Batch : 1. Extraction from corpus=0.9459s. Appending to vocabulary=0.4013s

Tokenizing corpus and creating the vocabulary. Processing batch 2. Line: 8192. Token: 589,523

Batch : 2. Extraction from corpus=0.9223s. Appending to vocabulary=0.4452s

…

Tokenizing corpus and creating the vocabulary. Processing batch 5. Line: 20480. Token: 2,956,691

Batch : 5. Extraction from corpus=0.8726s. Appending to vocabulary=0.663s

From the reader in Utils:

Reading in line n. : 0 ; number of tokens encountered: 0

Reading in line n. : 1000 ; number of tokens encountered: 46600

Reading in line n. : 2000 ; number of tokens encountered: 101406

Reading in line n. : 3000 ; number of tokens encountered: 146646

Reading in line n. : 4000 ; number of tokens encountered: 195,208

Reading in line n. : 5000 ; number of tokens encountered: 233327

…

Reading in line n. : 9000 ; number of tokens encountered: 430631

Reading in line n. : 10000 ; number of tokens encountered: 478,016

…

Reading in line n. : 20000 ; number of tokens encountered: 957731

Reading in line n. : 21000 ; number of tokens encountered: 1,009,332

Possibly because self.current\_tokens was never re-initialized to [] ?

Maybe this will make the manual dictionary creation viable?

Let us not use the CorpusTokenizerIterator, and see if we manage to eliminate this error…

Status: the Iterator was eliminated > and so was the error

}

Estimated time to create manually the dictionary of word frequencies from WikiText-103’s training set:

( (87,707,756 tokens after removing punctuation / 100,000 avg tokens in 2K lines)

\* 0.5 avg time to process 2K lines [it would be 0.39 on the Lambda machine]) / 60 seconds =

7.308 minutes

For the sake of speed, we will save the vocabulary in a HDF5 file.

We can now examine our vocabulary.

We notice that:

1. It is not lower-cased (New, Zoological, Sweden, Dutch)
2. It contains individual numbers (12, 85, 1914 are all different)
3. It contains words with frequency=1, like ‘televisual’
4. It contains non-Unicode characters, such as: 戦場のヴァルキュリア3
5. It still contains stopwords. For the purpose of example entities, they should be removed, using the same stopwords list.
6. We should keep the <unk> token, but it gets modified into ‘unk’ when removing the punctuation

What can be done to solve these problems?

1. When reading tokens in a sentence, lower-case them
2. Replace all numbers with at most 3 digits with a <num> token. If they have 4 digits they are likely to be dates, we keep them
3. Eliminate all words with frequency < 5,must be after creation.
4. When inserting a character into the word-frequency dictionary, check that it is Latin? (not a priority. How many cases like this are we going to have?)
5. Remove the tokens that are in nltk.corpus.stopwords.words(extended\_lang\_id)
6. A possible solution is: change the order of punctuation>tokenization to tokenization>punctuation. Trailing punctuation marks that pass the tokenizer should not be significant (i.e. only ‘s)… check.

# 31/08

With all the added checks, the creation of the vocabulary is slower (about 5s for 300K tokens)

Reading in line n. : 1800000 ; number of tokens encountered: 57504105 ; time elapsed = 5.6924 s

Vocabulary dictionary created, after processing 57,543,072 tokens

(if we count those obtained after the processing&filtering)

* Note: the isolated hyphen, ‘-‘, with frequency 894,577, is a problem.   
  Since the tokenizer does not separate it when handling joined words (e.g.: ['high-low', ':', 'experiment', 'with', 'hyphen-mechanism']), we can eliminate it in the punctuation step.
* Some characters are Greek / Japanese / Chinese / Arab.
* The numbers with >= 4 digits remain. This was partly intended. However, those with >4 should be turned into <num> as well (keep only years)

The order of operations: tokenization > punctuation is dictated by the necessity to keep the <unk> token.

However, since to build the vocabulary we do not care for the sentences, but only for the words and their frequencies, it would be opportune to remove the periods (‘.’) beforehand. That also allows us to use word\_tokenizer alone and not call the slower sentence\_tokenizer behind the scenes.

# 02/08

While we create the vocabulary from the training set of WikiText-103, we examine how to use it in the Bootstrap-Skipgram.

Vocab\_size = 3436 from the WT-validation corpus. Check… yes.

Almost everything is solved. There is only one problem remaining:

unk 449749

<unk> 123574

* Adjusted through manual rule, on meeting […, ‘<’, ‘unk’, ‘>’ ,…] tokens.

Examining whether the batch generator on input pairs (centerWord, wordToPredict) stops…

It eventually causes StopIteration

Major restructuring:

Adjusted objective for Task 1: Multi-sense Language Modeling based on dictionary resources