[Re-structuring 2](#_Toc52977139)

[Step by step 2](#_Toc52977140)

[CreateGraphInput.py 2](#_Toc52977141)

[senselabeled\_or\_text = True 2](#_Toc52977142)

[Vocabulary 2](#_Toc52977143)

[Compute Embeddings 4](#_Toc52977144)

[WordNetData and Post-processing 4](#_Toc52977145)

[Synonyms 6](#_Toc52977146)

[Graph matrix 7](#_Toc52977147)

[Graph edges 8](#_Toc52977148)

[T.setup\_train & co 9](#_Toc52977149)

[Graph: reflections & checks 10](#_Toc52977150)

# Re-structuring

The part that retrieves data from WordNet and stores them is OK, and does not need to be changed.

Regarding the vocabulary, reading data from a SLC corpus, et cetera there is a list of modifications that should have been implemented long ago:

* Make 2 separate folders, for “InputData\_sense-labeled” and “InputData\_standardText”
* The sense-labeled corpus must be *rewritten, lowercased.* There should be no difference in processing.
* I must write a protocol for experiments, that includes: checking what the Neural Network reads, checking that we can overfit on a mini-fragment
* New guideline: *no more manual switches*. We should have a folder with mini-corpuses for mini-experiments (both normal text and sense-labeled).

## Step by step

### CreateGraphInput.py

Functions:

1. reset()
2. reset\_embeddings()
3. exe\_from\_input\_to\_vectors(do\_reset, compute\_single\_prototype, sp\_method, vocabulary\_from\_senselabeled)

We must change the vocabulary\_from\_senselabeled parameter into a senselabeled\_or\_text parameter, that decides which folder we use, etc.

Moreover, we must have reset(senselabeled\_or\_text) and reset\_embeddings(senselabeled\_or\_text)

We also change the location of the graph and grapharea\_matrix: into Graph/DatasetTypeSubfolder

### senselabeled\_or\_text = True

**SLC.organize\_splits()**

Changed into:

SLC.organize\_splits(xml\_fnames)

with xml\_fnames = [**'semcor.xml'**]

The existing code works as intended, no need to modify. We recall that:

train\_root, valid\_root, test\_root = organize\_subcorpus(xml\_fpath, 0.8)  
write\_splits\_subcorpus(xml\_fpath, train\_root, valid\_root, test\_root)

### Vocabulary

It is now opportune for us to change:

vocabulary = **V.get\_vocabulary\_df**(senselabeled\_or\_text=vocabulary\_from\_senselabeled,  
 corpus\_txt\_fpaths=[vocab\_text\_fpath],  
 out\_vocabulary\_h5\_filepath=outvocab\_filepath, min\_count=2, lowercase=**True**)

**if** os.path.exists(out\_vocabulary\_h5\_filepath):  
 vocab\_df = pd.read\_hdf(out\_vocabulary\_h5\_filepath, mode=**'r'**)  
 logging.info(**"\*\*\* The vocabulary was loaded from the file "** + out\_vocabulary\_h5\_filepath)

We should not get the vocabulary\_of\_globals.h5 file only in 1 place, but in Vocabulary/SenseLabeled or Vocabulary/StandardText

**if** senselabeled\_or\_text:  
 vocabulary\_wordfreq\_dict = build\_vocabulary\_dict\_from\_senselabeled()  
**else**:  
 vocabulary\_wordfreq\_dict = build\_vocabulary\_dict\_fromtext(corpus\_txt\_fpaths)

**build\_vocabulary\_dict\_from\_senselabeled(lowercase)**:

vocab\_dict = {}  
  
tokens\_toexclude = [Utils.EOS\_TOKEN] *# + list(string.punctuation)  
# Commas and punctuation signs are present in the Sense-Labeled Corpus as separate tokens.  
# Therefore, it makes sense to keep them in the vocabulary, and thus in the graph, as globals*slc\_split\_names = [Utils.TRAINING, Utils.VALIDATION] *# , , Utils.TEST*

Consideration: since SemCor is smaller, it may make sense to create the vocabulary not only from the training set, but also from the validation set.

Not an extremely heavy choice anyway, given the 80-10-10 split. Let’s just keep T+V

**for** token\_dict **in** SLC.read\_split(slc\_split\_name):  
 token = VocabUtils.process\_word\_token(token\_dict, lowercase)  
 **if** token **not in** tokens\_toexclude:  
 **try**:  
 prev\_freq = vocab\_dict[token]  
 vocab\_dict[token] = prev\_freq + 1

Creation of the vocabulary with process\_word\_token(td, lowercase) has worked correctly so far.

However, this is the insertion point for the *lemmatized* forms of words in the vocabulary. This was brought about by, for instance, having ‘eyelids’ but not ‘eyelid’ in SemCor, where the second form actually has WordNet senses & data.

How to handle this?

Since the frequency of a word is not relevant in the vocabulary apart from whether it is < min\_count or not, I can just add +1 to the frequency of the lemmatized word as well.

We have to take the lemmatizer for this, from:

lemmatizer = nltk.stem.WordNetLemmatizer()  
lemmatized\_forms = [LN.lemmatize\_term(word, lemmatizer) **for** word **in** vocab\_wordls]

**build\_vocabulary\_dict\_fromtext(corpus\_txt\_fpaths)**:

It doesn’t need to be changed, since it gives us 33,278 – correctly – on WikiText-2.

### Compute Embeddings

reset\_embeddings(senselabeled\_or\_text). #ok

single\_prototypes\_file #should go into the appropriate subfolder

**compute\_single\_prototype\_embeddings**(vocabulary\_df, spvs\_out\_fpath, method):

**if** method == Method.DISTILBERT:

… # currently not implemented

**else**: *# i.e. elif method == Method\_for\_SPV.FASTTEXT:*

fasttext\_vectors = EFT.load\_fasttext\_vectors()

**for** idx\_word\_freq\_tpl **in** vocabulary\_df.itertuples():

word = idx\_word\_freq\_tpl[1]

word\_vector = fasttext\_vectors[word]

word\_vectors\_lls.append(word\_vector)

embds\_nparray = np.array(word\_vectors\_lls)  
 np.save(spvs\_out\_fpath, embds\_nparray)

### WordNetData and Post-processing

vocabulary\_ls = RID.retrieve\_data\_WordNet(vocabulary)

we must adapt to the subfolders in InputData and Vocabulary

**PI.prepare**(vocabulary\_ls, sp\_method)

*# Phase 1 - Preprocessing: eliminating quasi-duplicate definitions and examples, and lemmatizing synonyms & antonyms*preprocess(vocabulary\_ls, inputdata\_folder)

*# Phase 2 - Create the Vocabulary table with the correspondences (wordSense, integer index).*create\_senses\_indices\_table(inputdata\_folder, vocabulary\_folder):

* Possible insertion point: until now, we add the dummySense if the word is one of those without senses. However, it is opportune to add it only if the *lemmatized form* has no senses. (remember to exclude words with <3chars, like ‘its’)
* It is also necessary to send the senselabeled\_or\_text specification, so we know which filepaths to use (where to find the vocabulary, etc.)

vocab\_fpath, vocabulary\_df, vocabulary\_words\_ls

and also input\_folder\_fpath.

defs\_input\_filepath, examples\_input\_filepath, defs\_input\_db, examples\_input\_db.

output\_filepath.

out\_indicesTable\_db = sqlite3.connect(output\_filepath)

out\_indicesTable\_db\_c = out\_indicesTable\_db.cursor()

out\_indicesTable\_db\_c.execute(**'''CREATE TABLE IF NOT EXISTS …**

**word\_senses\_series\_from\_defs** =   
 defs\_input\_db[Utils.DEFINITIONS][Utils.SENSE\_WN\_ID]

**word\_senses\_ls** = [sense\_str **for** sense\_str **in** word\_senses\_series\_from\_defs **if** Utils.get\_word\_from\_sense(sense\_str) **in** vocabulary\_words\_ls]

**for** wn\_id **in** word\_senses\_ls:

…out\_indicesTable\_db\_c.execute(

**"INSERT INTO indices\_table VALUES (?,?,?,?,?,?)"**,

(wn\_id, my\_vocabulary\_index,start\_defs\_count,   
 end\_defs\_count, start\_examples\_count, end\_examples\_count))

*# add the word to the set of words that do have a sense*words\_with\_senses\_set.add(Utils.get\_word\_from\_sense(wn\_id))

words\_without\_senses\_set = set(vocabulary\_words\_ls).difference(words\_with\_senses\_set)

\*\*\* XYZ

Insertion point:

for all words in words\_without\_senses\_set:

if their lemmatized form (that we use for senses-processing) has a sense   
 and they are >3 characters long,   
they do not need the addition of a dummySense.

**for** word **in** words\_without\_senses\_set:  
 *# no definitions nor examples to add here. We will add the global vector as the vector of the dummy-sense.* dummy\_wn\_id = word + **'.'** + **'dummySense'** + **'.01'**  out\_indicesTable\_db\_c.execute(**"INSERT INTO indices\_table VALUES (?,?,?,?,?,?)"**, (dummy\_wn\_id, my\_vocabulary\_index,…

Then,

*# Phase 3 - get the sentence embeddings for definitions and examples, using BERT or FasText, and store them*CE.compute\_elements\_embeddings(Utils.DEFINITIONS, embeddings\_method, inputdata\_folder)  
CE.compute\_elements\_embeddings(Utils.EXAMPLES, embeddings\_method, inputdata\_folder)  
  
*# Phase 4 - use PCA dimensionality reduction for definitions and examples, for future graph processing*apply\_PCA\_to\_defs\_examples(embeddings\_method, inputdata\_folder)

Checking the pipeline up to this point:

SemCor: |V|=24688 from the training + validation sets. ‘eyelid’ and ‘eyelids’ can both be found.   
|Definitions|=33574: say.n.01, …, say.v.11, friday.n.01, investigation.n.02, atlanta.n.02, etc.

|Examples|=29594.

note: I observe that the synonyms are… not lowercased. This must be rectified.

### Synonyms

Either the following is wrong, or I extract the word from the sense at graph-creation time:

|  |  |  |
| --- | --- | --- |
| index | sense\_wn\_id | synonyms |
| 0 | friday.n.01 | Friday |
| 1 | friday.n.01 | Fri |
| 0 | investigation.n.02 | investigating |
| 0 | atlanta.n.01 | Atlanta |
| 1 | atlanta.n.01 | capital\_of\_Georgia |
| 2 | atlanta.n.02 | Atlanta |
| 3 | atlanta.n.02 | battle\_of\_Atlanta |
| 0 | produce.n.01 | green\_goods |
| 1 | produce.n.01 | green\_groceries |
| 0 | friday.n.01 | Friday |
| 1 | friday.n.01 | Fri |

How do we process synonyms in the graph?

syn\_edges = get\_edges\_nyms(Utils.SYNONYMS, globals\_vocabulary\_df, num\_senses, inputdata\_folder), i.e.:

**for** tpl **in** nyms\_df.itertuples():  
 word\_sense = tpl.sense\_wn\_id  
 word1 = Utils.get\_word\_from\_sense(word\_sense)

word1 = VocabUtils.process\_word\_token({**'surface\_form'**: word1}, lowercasing=**True**) # we must decide how to handle this

word2 = getattr(tpl, nyms\_name)  
 word2 = VocabUtils.process\_word\_token({**'surface\_form'**: word2}, lowercasing=**True**)

and then we get their indices, and establish a bidirectional edge

edges\_ls.append((global\_idx\_1, global\_idx\_2))  
edges\_ls.append((global\_idx\_2, global\_idx\_1))

Recap: the association from WordNet, sense\_wn\_id🡪synonym, is actually viable. We get the word from the sense, and connect the 2 globals.

The problem connected to this is: how to handle the fact that SemCor is not lowercased, but only processed/received as lowercased?

We can still only process/receive it as lowercased, but this will not be determined by a (fragile) manual switch anymore, but from whether we are on SLC or not, for instance whether “SenseLabeled” is in the path of the inputdata\_folder.

WikiText-2: |V|=33,278, as expected.

|Definitions|=44756. no.1.01, …, japanese.n.01, …, flea.n.01

|Examples|=40188

### Graph matrix

**create\_graph(method, slc\_corpus):**

single\_prototypes\_file, globals\_vocabulary\_fpath, globals\_vocabulary\_df, globals\_vocabulary\_ls.

E\_embeddings = torch.tensor(np.load(os.path.join(inputdata\_folder, single\_prototypes\_file))).to(torch.float32)

use\_PCA = Utils.GRAPH\_EMBEDDINGS\_DIM < embeddings\_size

X\_definitions = load\_senses\_elements(Utils.DEFINITIONS, method, use\_PCA, inputdata\_folder)  
X\_examples = load\_senses\_elements(Utils.EXAMPLES, method, use\_PCA, inputdata\_folder)

X\_globals = torch.tensor(initialize\_globals(X\_definitions, E\_embeddings, globals\_vocabulary\_ls)).to(torch.float32)

Currently, a global is initialized as the average of all the definitions of the senses of the word.

Alternative choices are possible, such as the (possibly PCA-reduced) FastText embedding. However, when we are concatenating FastText embedding ++ global node-state as input signals of the RNN, I would prefer if they were not identical at the start.

**if** sense\_start\_defs != sense\_end\_defs:  
 sense\_def\_vectors =   
 X\_definitions[sense\_start\_defs:sense\_end\_defs]  
 **if** words\_definitionvectors\_dict[word] **is None**:  
 words\_definitionvectors\_dict[word] = sense\_def\_vectors  
 **else**:  
 words\_definitionvectors\_dict[word] =   
 np.concatenate([words\_definitionvectors\_dict[word],   
 sense\_def\_vectors])

**else**: *# if there are no definitions, use the (PCA-reduced) version of the word embedding*

**if** words\_definitionvectors\_dict[word] **is None**:  
 words\_definitionvectors\_dict[word] =   
 np.expand\_dims(E\_reduced\_embeddings[word\_vocabulary\_index,:], axis=0)

**for** word **in** words\_definitionvectors\_dict.keys():  
 average\_of\_definitions =   
 np.average(words\_definitionvectors\_dict[word], axis=0)  
 X\_globals\_ls.append(average\_of\_definitions)

X\_senses, num\_dummysenses = initialize\_senses(X\_definitions, X\_examples, X\_globals, globals\_vocabulary\_ls, average\_or\_random\_flag=**True**)

# senses are initialized with the average of the definitions, dummySenses with the globals from X\_globals

X = torch.cat([X\_senses, X\_globals, X\_definitions, X\_examples])

### Graph edges

def\_edges\_se = get\_edges\_elements(Utils.DEFINITIONS, num\_senses + num\_globals)

exs\_edges\_se = get\_edges\_elements(Utils.EXAMPLES, num\_senses + num\_globals+ X\_definitions.shape[0])

**if** slc\_corpus:  
 sc\_external\_edges =   
 get\_additional\_edges\_sensechildren\_from\_slc(globals\_vocabulary\_df,  
 globals\_start\_index\_toadd=num\_senses,   
 inputdata\_folder=inputdata\_folder)  
 sc\_edges.extend(sc\_external\_edges)

We read the training set alone, to create the connections between globals and the senses of other words (e.g. ‘atlanta’ -> location.n.01)

i.e.:

**while True**:  
 token\_dict = slc\_train\_corpus\_gen.\_\_next\_\_()

**if 'wn30\_key' in** keys:  
 wordnet\_sense = try\_to\_get\_wordnet\_sense(wn30\_key)

**else**:  
 **continue** *# there was no sense-key specified for this token*

*# 2) Get the global word of this token*word = VocabUtils.process\_word\_token(token\_dict, lowercasing=on\_senselabeled)

lemmatized\_word = lemmatize\_term(word, lemmatizer)*# since currently we always lemmatize in SelectK and other sense architectures***if** lemmatized\_word **not in** wordnet\_sense: *# we are connecting all the "external" senses, e.g. say->state.v.01*

*# We add self-loops to all globals without a sense.*edges\_selfloops = get\_edges\_selfloops(sc\_edges, num\_globals=num\_globals, num\_dummysenses=num\_dummysenses)

*# The dummy senses should already be connected to their globals*

*# And therefore, this should yield no new edges*

syn\_edges = get\_edges\_nyms(Utils.SYNONYMS, globals\_vocabulary\_df, num\_senses, inputdata\_folder)

ant\_edges = get\_edges\_nyms(Utils.ANTONYMS, globals\_vocabulary\_df, num\_senses, inputdata\_folder)

etc.etc.

**Issue**: debugging the graph

INFO : word=the

INFO : words\_definitionvectors\_dict[word]=(1, 300)

INFO : word=said

For some strange reason, I don’t find “said” in the vocabulary, only “sage, sail”

but ‘said’ is in the h5 vocabulary of senselabeled… how are we iterating?

**while** (**True**):  
 db\_row = indicesTable\_db\_c.fetchone()  
 **if** db\_row **is None**:  
 **break** word\_sense = db\_row[0]

we are iterating over the senses of the indicesTable. However, this time ‘said.dummySense.01’ is not there anymore, because ‘say’ has senses and is the lemmatized form of said’. Consequence of the latest modification.

At this point, I prefer to have all the globals initialized with the (possibly PCA-reduced) FastText embedding of the corresponding word.

### T.setup\_train & co

graph\_dataobj = DG.get\_graph\_dataobject(new=**False**, method=method, slc\_corpus=slc\_or\_text\_corpus).to(DEVICE)

grapharea\_matrix = AD.get\_grapharea\_matrix(graph\_dataobj, grapharea\_size, hops\_in\_area=1, graph\_folder=graph\_folder)

embeddings\_matrix = torch.tensor(np.load(os.path.join(inputdata\_folder, single\_prototypes\_file))).to(torch.float32)

vocabulary\_df = pd.read\_hdf(globals\_vocabulary\_fpath, mode=**'r'**)

**if** slc\_or\_text\_corpus:  
 vocabulary\_numSensesList = vocabulary\_df[**'num\_senses'**].to\_list().copy()  
 **if** all([num\_senses == -1 **for** num\_senses **in** vocabulary\_numSensesList]):  
 vocabulary\_df = AD.compute\_globals\_numsenses(graph\_dataobj, grapharea\_matrix, grapharea\_size)

This is necessary to distinguish the words that have multiple senses, so we can record separate statistics.

model = RNNs.RNN(graph\_dataobj, grapharea\_size, grapharea\_matrix, vocabulary\_df,  
 embeddings\_matrix, include\_globalnode\_input,  
 batch\_size=batch\_size, n\_layers=3, n\_hid\_units=1024)

**if** n\_gpu > 1 **and** allow\_dataparallel:  
 logging.info(**"Using "** + str(n\_gpu) + **" GPUs"**)  
 model = torch.nn.DataParallel(model, dim=0)  
 model\_forDataLoading = model.module

model.to(DEVICE)

…train\_dataloader, valid\_dataloader

T.run\_train(model, learning\_rate, train\_dataloader, valid\_dataloader, num\_epochs, predict\_senses, with\_freezing)

does not expect modifications.

## Graph: reflections & checks

In the current version all the globals are initialized as the (possibly PCA-reduced) FastText embedding of the word.

### Statistics of version 1.0

On **SemCor**:

INFO : X\_senses.shape=torch.Size([**40095**, 300])

INFO : X\_globals.shape=torch.Size([**24689**, 300])

INFO : X\_definitions.shape=torch.Size([33568, 300])

INFO : X\_examples.shape=torch.Size([29585, 300])

graph ranges = [0, senses, 40095, globals, 64784, definitions, 98352, examples, 127937]

previous graph ranges = [0, senses, 41206, globals, 63988, defs, 94434, examples, 122355]

previously having

INFO : X\_senses.shape=torch.Size([**41206**, 200]) # senses with data: 30445

INFO : X\_globals.shape=torch.Size([**22782**, 200]) (they were only from the training set)

INFO : X\_definitions.shape=torch.Size([30446, 200])

INFO : X\_examples.shape=torch.Size([27921, 200])

globals\_needing\_selfloop=[40096, 40106, 40113, 40125, …, 64757, 64766, 64773, 64777]

On **WikiText-2**:

INFO : X\_senses.shape=torch.Size([**46710**, 300])

INFO : X\_globals.shape=torch.Size([**33278**, 300])

INFO : X\_definitions.shape=torch.Size([28141, 300])

INFO : X\_examples.shape=torch.Size([26593, 300])

previously:

INFO : X\_senses.shape=torch.Size([28141, 300]) (it was without dummySenses)

INFO : X\_globals.shape=torch.Size([**33278**, 300])

INFO : X\_definitions.shape=torch.Size([28141, 300])

INFO : X\_examples.shape=torch.Size([26593, 300])

INFO : globals\_needing\_selfloop=[46731, 46736, 46765, 46767, … s, 79940, 79941, 79942]

### Lemmatized forms, self-loops vs edges

Why do many globals need selfloop?

Because now we are not adding a dummySense anymore for all the globals where the lemmatized form has a sense.

Those globals are left in the graph, they have their own embedding but they are isolated form the other nodes. Why?

For prediction reasons: when we predict the globals, we want to choose among all of them.

However, they should not be disconnected from their lemmatized form.

Bonus: the lemmatized form has senses, that are connected to definitions et cetera.

Wouldn’t it be better if we connected the lemmatized forms through a new edge type, “lemmatized”, that leads from said --to--> say ?