[Current Next Steps 3](#_Toc48562412)

[Preliminary information (Datasets, graphs) 5](#_Toc48562413)

[SemCor 5](#_Toc48562414)

[SemCor stats 5](#_Toc48562415)

[Graph (min\_freq=2, with dummySense) 5](#_Toc48562416)

[Graph (vocab. min. frequency=2, no dummy Sense) 6](#_Toc48562417)

[Graph (vocab. min. frequency=2, with dummy Sense) 7](#_Toc48562418)

[SemCor + MASC + OMSTI(300MB) 8](#_Toc48562419)

[Number of documents / sentences 8](#_Toc48562420)

[Graph 8](#_Toc48562421)

[Recap of experiments made on SemCor, and new directions 10](#_Toc48562422)

[Verifying the RNN setup on the standard LM task 13](#_Toc48562423)

[Table & Graph – WT-2 13](#_Toc48562424)

[Mini-experiment on fragment of WT-2 14](#_Toc48562425)

[Model 14](#_Toc48562426)

[Mini-exp A 15](#_Toc48562427)

[2LSTMs: Standard LM on WT-2 17](#_Toc48562428)

[Experiment A 17](#_Toc48562429)

[Model B 19](#_Toc48562430)

[Mini-exp B 19](#_Toc48562431)

[Experiment B 20](#_Toc48562432)

[Standard LM task on SemCor 22](#_Toc48562433)

[Table & Graph 22](#_Toc48562434)

[Mini-experiment on fragment of SemCor 23](#_Toc48562435)

[Model A 23](#_Toc48562436)

[Mini-exp A 24](#_Toc48562437)

[Experiment A 26](#_Toc48562438)

[Experiment B 27](#_Toc48562439)

[Model 27](#_Toc48562440)

[Mini-exp B 27](#_Toc48562441)

[Experiment B 28](#_Toc48562442)

[Experiment C 29](#_Toc48562443)

[Model 29](#_Toc48562444)

[Mini-exp C 29](#_Toc48562445)

[Experiment C 30](#_Toc48562446)

[Observations 31](#_Toc48562447)

[On the results 31](#_Toc48562448)

[Reviewing the input loading and vocabulary 31](#_Toc48562449)

[Next steps 34](#_Toc48562450)

[Replicating the old result (lowercased, <num>, Vocab from Training set) 35](#_Toc48562451)

[Graph 35](#_Toc48562452)

[Model 35](#_Toc48562453)

[Experiment 36](#_Toc48562454)

[Gathering more Sense-Labeled Data 37](#_Toc48562455)

[Review of Sense-Labeled Corpuses 37](#_Toc48562456)

[Running the pipeline, graph 38](#_Toc48562457)

[Number of documents / sentences in the corpuses 38](#_Toc48562458)

[Graph 39](#_Toc48562459)

[Graph retrieval 40](#_Toc48562460)

[SLC: Experiment A 41](#_Toc48562461)

[Model 41](#_Toc48562462)

[Exp A 42](#_Toc48562463)

[Observations: 42](#_Toc48562464)

[Freezing standard LM and then proceeding to Senses 43](#_Toc48562465)

[Notes & ideas 43](#_Toc48562466)

[Implementation 45](#_Toc48562467)

[Mini-experiment with freezing on SemCor, v. 1.0 45](#_Toc48562468)

[Mini-experiment with freezing on SemCor, v. 1.1 46](#_Toc48562469)

[Mini-experiment with freezing on SemCor, v. 1.2 48](#_Toc48562470)

[Implementation, II 48](#_Toc48562471)

[Mini-experiment with freezing on SLC, v. 1.2 48](#_Toc48562472)

[Gradually building a model 49](#_Toc48562473)

# Current Next Steps

Step 1: re-run the pipeline for SemCor, and try to replicate exactly the old result with 185 Valid-PPL. This would demonstrate that the architecture still works as intended.

Status: done,

the old result was so good because the SemCor corpus had been **lowercased**, and 2/3/n digit numbers had been turned into <num>.

we got 219.4 because we used an LSTM where a GRU works better. See [Experiment](#_Experiment).

This validates the architecture (i.e. it’s not bugged, as we also see from WikiText-2). However, when we try experiments on SemCor’s 650K training tokens, we get very high PPL values, like 445 Valid-PPL at [Experiment C](#_Experiment_C).

We recall that Penn-Treebank has 929K training tokens with heavy pre-processing, namely “…lower-cased, numbers replaced with N, punctuation removed. The vocabulary is the most frequent 10k words”. WikiText-2 has ~2 million training tokens.

Since the RNN architecture works as intended and is not bugged, we try to reach better performance by including *more* sense-labeled data, see Step 3.

Step 2: try a 1024>1024>512 architecture on WT-2, with no dropout, to try to bring the Valid-PPL well under 200.

status:

done. The first experiment on WT-2 is the one with a 1150>1150>400 LSTM that (mistakenly) applies a standard dropout of p=0.1. See [Experiment A](#_Experiment_A), it reaches 200.5 Validation PPL.

Moreover, [Experiment B](#_Experiment_B), using a 1024>1024>512 GRU with no dropout, reaches Valid-PPL

Step 3: gather enough sense-labeled data to obtain a corpus of size comparable to WT-2, to try to obtain a similar Valid-PPL.

I need to review the other Sense-Labeled Corpuses. The ones eligible for inclusion are those labelled with **WordNet 3.0** senses, i.e.:

SemCor (228k annotated / 778k words), OMSTI (1 million / 35 millions), MASC (115K / 596K). Ontonotes would be too, but they are saved in a complex acess-DB format. + SemEval 2013 (8.3K words), SemEval 2015(2.6K)

The current expanded SLC contains SemCor + MASC + the first 300 MB of text of OMSTI (for the sake of speed in the current experiments, we did not include it in its entirety).

training tokens: 4,875,767. Validation tokens: 605,366

The first tentative [Exp A](#_Exp_A) reaches 158 Valid-PPL, but there is a chance I am reading one of the subcorpuses twice so it may be bugged. It must be checked and re-done.

Step 4: continue adjusting the code and running mini-experiments, to prepare the freezing mechanism.

Using the flags predict\_senses=**True** in training\_setup() and with\_freezing=**True** in training\_loop(), we apply the current version of the freezing mechanism:

* we set p.requires\_grad=**False** on all the parameters of the globals’ GRU and softmax
* We recreate the optimizer, to eliminate any remaining momentum
* We activate the senses’ GRU with model\_forParameters.predict\_senses = **True**.
* We set after\_freezing\_flag = **True**, so the loss will be computed as   
  loss = loss\_sense instead of loss = loss\_global + loss\_sense

Despite all of this, the globals’ loss still changes in the epochs after the freezing point.

ToDo:

build a small toy-model, that takes in the same input and labels, and examine the freezing mechanism.

Step 5: Finish the freezing mechanism – a very minor oscillation due to the embeddings is ok

Step 6: Use lowercased SemCor, for the sake of brevity and development. Execute RNN4Senses experiments, with and without the freezing mechanism

# Preliminary information (Datasets, graphs)

## SemCor

### SemCor stats

Note: currently I am also including punctuation in the vocabulary from SLC.

Training tokens: 646,038

Validation tokens: 80,760

Mini-dataset 1 : 180 tokens.

Mini-dataset 2: 63 tokens.

### Graph (min\_freq=2, with dummySense)

Data(edge\_index=[2, 216891], edge\_type=[216891], node\_types=[197686], num\_relations=[1], x=[197686, 300])

INFO : X\_senses.shape=torch.Size([73706, 300])

INFO : X\_globals.shape=torch.Size([53139, 300])

INFO : X\_definitions.shape=torch.Size([37859, 300])

INFO : X\_examples.shape=torch.Size([32982, 300])

Graph Intervals: Senses=[0, 73706) ; Globals=[73706 , 126845);   
Definitions=[126845, 164704); Examples=[164704,197686)

edges-definitions: [126845, 164703] -> [0, 37858] # from the definitions to the senses-with-data

edges-examples: [164704, 197685] -> [1, 37849] # not all senses have examples

sc (sense-children): [73706, 126844] -> [0,73705] # from the globals to all the senses, both with data and dummy.

INFO : Reading the sense-labeled corpus, to create the connections between globals and the senses that belong to other words.

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/semcor.xml

INFO : Error while attempting to execute query: SELECT vocab\_index FROM indices\_table WHERE word\_sense='governor's\_race.n.01' . Skipping sense …

get\_additional\_edges\_sensechildren\_from\_slc: [73707, 126838] -> [1, 37857] # from globals to the senses-with-data

INFO : sc\_edges\_with\_external.\_\_len\_\_()=16504

get\_edges\_selfloops:

INFO : []

INFO : len(edges\_ls)==0 # now we do not need to add self-loops, because of the connections to the dummySenses, as expected.

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=90210 # (i.e. 73707 + 16504)

INFO : syn\_edges.\_\_len\_\_()=50084

INFO : ant\_edges.\_\_len\_\_()=5756

### Graph (vocab. min. frequency=2, no dummy Sense)

We gather the WordNet data and build the graph using SemCor’s vocabulary.

Constructing X, matrix of node features

**X\_senses**.shape=torch.Size([**25986**, 300])

**X\_globals**.shape=torch.Size([**21988**, 300])

**X\_definitions**.shape=torch.Size([**25986**, 300])

**X\_examples**.shape=torch.Size([**26003**, 300])

Defining the edges: def, exs

def\_edges\_se.\_\_len\_\_()=25986

exs\_edges\_se.\_\_len\_\_()=26003

Defining the edges: sc

sc\_edges.\_\_len\_\_()=25986

sc\_edges\_with\_selfloops.\_\_len\_\_()=38611

This way, we also determine that there were 38611– 21988**= 16623** globals with no dictionary information, over a total of **21988 (75.6%).**

**syn\_edges**.\_\_len\_\_()=**26222**

**ant\_edges**.\_\_len\_\_()=**3780**

Data(edge\_index=[2, 120602], edge\_type=[**120602**], node\_types=[**99963**], num\_relations=[1], **x=[99963, 300]**)

### Graph (vocab. min. frequency=2, with dummy Sense)

INFO : X\_senses.shape=torch.Size([**43559**, 300])

INFO : X\_globals.shape=torch.Size([**25693**, 300])

INFO : X\_definitions.shape=torch.Size([29133, 300])

INFO : X\_examples.shape=torch.Size([28077, 300])

Graph ranges:

senses: [0,43559)

globals: [43559,69252)

definitions: [69252,98385)

examples: [98385,126462)

edges-definitions : [69252,98384](defs) -> [0, 29132](senses with data)

INFO : def\_edges\_se.\_\_len\_\_()=29133

edges-examples : [98385, 126461](examples) -> [0, 29123](senses with data)

INFO : exs\_edges\_se.\_\_len\_\_()=28077

INFO : Defining the edges: sc

INFO : Reading the sense-labeled corpus, to create the connections between globals and the senses that belong to other words.

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/semcor.xml

edges-get\_additional\_edges\_sensechildren\_from\_slc :

[43563, 69072](globals) -> [0, 29132](senses with data)

INFO : sc\_edges\_with\_external.\_\_len\_\_()=10165

edges-get\_edges\_sensechildren : [43559, 69251] -> [0, 43558]

INFO : sc\_edges.\_\_len\_\_()=53724

INFO : get\_edges\_selfloops>max\_sense=43558

INFO : []

INFO : len(edges\_ls)==0

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=53724

INFO : Defining the edges: syn, ant

INFO : syn\_edges.\_\_len\_\_()=31330

INFO : ant\_edges.\_\_len\_\_()=4158

INFO : Data(edge\_index=[2, 146422], edge\_type=[146422], node\_types=[126462], num\_relations=[1], x=[126462, 300])

## SemCor + MASC + OMSTI(300MB)

### Number of documents / sentences

INFO : Organizing subcorpus at: TextCorpuses/My Sense-Labeled Corpus/semcor.xml

INFO : The corpus at TextCorpuses/My Sense-Labeled Corpus/**semcor**.xml has **352** documents

INFO : Training dataset will contain: **281** documents , Validation dataset will contain: **35** documents , Test dataset will contain: **36** documents

INFO : Organizing subcorpus at: TextCorpuses/My Sense-Labeled Corpus/masc.xml

INFO : The corpus at TextCorpuses/My Sense-Labeled Corpus/**masc**.xml has 1 documents

INFO : Training dataset will contain: **25407** sentences , Validation dataset will contain: **3176** sentences , Test dataset will contain: **3176** sentences

INFO : Organizing subcorpus at: TextCorpuses/My Sense-Labeled Corpus/subset\_omsti\_aa.xml

INFO : The corpus at TextCorpuses/My Sense-Labeled Corpus/subset\_omsti\_aa.xml has 1 documents

INFO : Training dataset will contain: **83703** sentences , Validation dataset will contain: **10463** sentences , Test dataset will contain: **10463** sentences

INFO : \*\*\* Creating vocabulary at Vocabulary/vocabulary\_of\_globals.h5

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/semcor.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/masc.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/subset\_omsti\_aa.xml

INFO : Removing from the vocabulary words with frequency **< 2**

INFO : \*\*\* The vocabulary was created. Number of words= **52781**\*\*\*

### Graph

INFO : Constructing X, matrix of node features

INFO : X\_senses.shape=torch.Size([**73177**, 300])

INFO : X\_globals.shape=torch.Size([**52781**, 300])

INFO : X\_definitions.shape=torch.Size([35850, 300])

INFO : X\_examples.shape=torch.Size([31976, 300])

Graph ranges:

senses=[0, 73177) (senses with data: 35850)

globals=[73177, 125958)

definitions=[125958, 161808)

examples=[161808, 193784)

INFO : Defining the edges: def, exs

INFO : def\_edges\_se.\_\_len\_\_()=35850

INFO : exs\_edges\_se.\_\_len\_\_()=31976

INFO : Defining the edges: sc

INFO : Reading the sense-labeled corpus, to create the connections between globals and the senses that belong to other words.

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/semcor.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/masc.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/subset\_omsti\_aa.xml

INFO : sc\_edges\_with\_external.\_\_len\_\_()=13642

INFO : sc\_edges.\_\_len\_\_()=86819

INFO : get\_edges\_selfloops>max\_sense=73176

INFO : []

INFO : len(edges\_ls)==0

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=86819

INFO : Defining the edges: syn, ant

INFO : syn\_edges.\_\_len\_\_()=57814

INFO : ant\_edges.\_\_len\_\_()=5940

Data(edge\_index=[2, 218399], edge\_type=[218399], node\_types=[193784], num\_relations=[1], x=[193784, 300])

# Recap of experiments made on SemCor, and new directions

|  |  |  |  |
| --- | --- | --- | --- |
| Text | Architecture | Senses’ method, other hyperparameters | Results on Validation set |
| Preprocessing: min\_frequency=2 + <num>  |globals| =21988  |senses| = 25986  no dummySense, the senses’ RNN proceeds by “snapshots” | Dropout(p=0)  main\_rnn:  (0): GRU(300, 1024)  (1): GRU(1024, 1024)  (2): GRU(1024, **512**)  (rnn\_senses):  GRU(1024 x 2Layers)  + linear2Global, linear2Senses as usual | none (GRU only)  Input signals: 1) The word embedding of the current global (d=300)  batch\_size=40  TBPTT length=35 learning srate=10^(-4) | Globals:  **185.95**@ep14  Senses:  **793.37**@ep1 |
| As above,  Preprocessing: min\_frequency=2 + <num>  |globals| =21988  |senses| = 25986  no dummySense | Globals:  Main GRU with 3 layers  (1024 > 1024 > **1024**)  Senses: GRU(1024 x 2Layers) | none (GRU only)  Input signals: 1) The word embedding of the current global (d=300)  batch\_size=40  TBPTT length=35 learning rate=5\*e-5 | globals:  **188.67** @ep23  senses:  **811.1** @ep2 |
| As above,  Preprocessing: min\_frequency=2 + <num>  |globals| =21988  |senses| = 25986  no dummySense | Globals:  Main GRU with 3 layers  (1024 > 1024 > 1024)  Senses: GRU(1024 x 2Layers) | Senses’ method: SelectK5,  apply softmax on the logits of the senses of the most likely ***k=5*** globals  bs=40 x seq\_len=35  learning rate=5\*e-5 | Globals:  **192.49**@ ep16,  senses:  **25986.01**@ ep1  We remember that when operating with SelectK, the PPL value is altered. This consideration was the reason for the introduction of the number of correct predictions. |
| As above,  Preprocessing: min\_frequency=2 + <num>  |globals| =21988  |senses| = 25986  no dummySenses | Globals:  Main GRU with 3 layers  (1024 > 1024 > 1024)  Senses: GRU(1024 x 2Layers)s | SelectK1, apply softmax on the logits of the senses of the most likely ***k=1*** global | Globals:  **191.17** @ ep20  Correct/total:  18315/81760  Top10/total:  39999/81760  Senses:  Correct/total:  448/7474  Top10/total:  757/7474  n: we have few senses, since it was after freq=2 and <num> pre-processing, and without dummySenses |
| After this point, I switched to the vocabulary with min\_freq=1, |V|=53,138.  Then: the dummySense got implemented, e.g. adding  ‘for.*dummySense.01*’ nodes, connected to the globals like ‘for’. The sense label is now always present.  Then: the double-PPL got implemented (even if the results on the multi-sense in large-scale experiments seem dubious, I should double-check it)  Then: I decided to try out existing architectures to try to obtain a better RNN. | | | |
| No preprocessing at all, min\_freq=1.  |globals|=53139  |senses|=73706  with dummySenses | From “Improving Neural Language Models with a continuous cache”, but using Adam instead of Adagrad + gradient clipping  Dropout(p=0.65)  main\_rnn\_ls:  (0): LSTM(300, 1024)  + linear2global  senses\_rnn\_ls:  (0): LSTM(300, 1024)  + linear2senses | none (LSTM only)  bs=20 x seq\_len=30  learning rate=10^(-4) | Globals:  **655.76**@ep6  Senses:  **1775.81** @ ep1  Correct/total:  4259/81000  Top10/total:  24921/81000  Senses (of globals with multiple senses):  Training PPL @ep1 is 35.16(?),  but  correct/total= 5/24348  and top10/total=  10130/243482 |
| No preprocessing at all, min\_freq=1.  |globals|=53139  |senses|=73706  with dummySenses | Copying the structure of the AWD-LSTM, although I am not tying the weights:  Main LSTM with 3 layers (1150>1150>400)  Senses LSTM identical  (1150>1150>400) | none (LSTM only)  bs=40 x seq\_len=35  learning rate=5e(-5) | Globals:  **1067.93** @ ep9  Senses:  **1856.19** @ ep2 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

2 directions:

Try the current RNN setup on WT-2, see if I can replicate acceptable results.

Then: move on to SemCor (globals only)

Then: extend to Senses on SemCor

Hypothesis : I could. train a standard LM, freeze, and then train another part on the senses afterwards

Hypothesis : instead of using an architecture, I could use an heuristic: choose the sense where the definition has the greatest overlap with the sentence, and the methods I add should be better than the baseline.

# Verifying the RNN setup on the standard LM task

First of all, we re-create the indices\_table.sql with the dummy senses, and then the graph for WT-2.

## Table & Graph – WT-2

The **indices\_table.sql** now contains:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| zoologist.n.01 | 28140 | 28140 | 28141 | 26593 | 26593 |
| Kent.dummySense.01 | 28141 | 28141 | 28141 | 26593 | 26593 |

**The graph**:

INFO : Constructing X, matrix of node features

INFO : X\_senses.shape=torch.Size([**50938**, 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])

Graph ranges:

senses=[0, **50938**)

globals=[**50938**,84216)

definitions=[84216, 112357)

examples=[112357, 138950)

INFO : def\_edges\_se.\_\_len\_\_()=28141

INFO : exs\_edges\_se.\_\_len\_\_()=26593

Defining the edges: sc

sc\_edges.\_\_len\_\_()=50938

get\_edges\_selfloops>max\_sense=50937

[]

len(edges\_ls)==0

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=50938

INFO : Defining the edges: syn, ant

INFO : syn\_edges.\_\_len\_\_()=42024

INFO : ant\_edges.\_\_len\_\_()=3942

Data(edge\_index=[2, 151638], edge\_type=[151638], node\_types=[138950], num\_relations=[1], x=[138950, 300])

**Graph retrieval**:

AD.get\_node\_data(grapharea\_matrix, **174**, grapharea\_size=32, features\_mask= (True,False,False))

The adjacent nodes are:

(tensor([ 174, 84390, 53468], device='cuda:0'), None, None)

|  |  |
| --- | --- |
| 174 (sense) | access.v.01 |
| 84390 (definition n. 84390-84216=174) | obtain or retrieve from a storage device; as of information on a computer |
| 53468 (global n. 53468-50938=2530) | access |

AD.get\_node\_data(grapharea\_matrix, **59000**, grapharea\_size=32, features\_mask=(True,False,False))

Adjacent nodes:

(tensor([59000, 17105, 17106, 70867], device='cuda:0')

|  |  |
| --- | --- |
| 59000 (59000-50938 = 8062nd globals) | oath |
| 17105 (sense) | oath.n.02 |
| 17106 (sense) | oath.n.03 |
| 70867 (70867-50938 = 19929th global) | curse |

## Mini-experiment on fragment of WT-2

### Model

DataParallel(

(module): RNN(

(dropout): Dropout(p=0.1, inplace=False)

(main\_rnn\_ls): ModuleList(

(0): LSTM(300, 1150)

(1): LSTM(1150, 1150)

(2): LSTM(1150, 400)

)

(linear2global): Linear(in\_features=400, out\_features=33278, bias=True)

)

)

Parameters:

('module.X', torch.Size([138950, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([1150]), torch.float32, False)

('module.embedding\_zeros', torch.Size([1, 300]), torch.float32, False)

('module.memory\_hn', torch.Size([3, 2, 1150]), torch.float32, False)

('module.memory\_cn', torch.Size([3, 2, 1150]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([3, 2, 1150]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([3, 2, 1150]), torch.float32, False)

('module.main\_rnn\_ls.0.weight\_ih\_l0', torch.Size([4600, 300]), torch.float32, True)

('module.main\_rnn\_ls.0.weight\_hh\_l0', torch.Size([4600, 1150]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_ih\_l0', torch.Size([4600]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_hh\_l0', torch.Size([4600]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_ih\_l0', torch.Size([4600, 1150]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_hh\_l0', torch.Size([4600, 1150]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_ih\_l0', torch.Size([4600]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_hh\_l0', torch.Size([4600]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_ih\_l0', torch.Size([1600, 1150]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_hh\_l0', torch.Size([1600, 400]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_ih\_l0', torch.Size([1600]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_hh\_l0', torch.Size([1600]), torch.float32, True)

('module.linear2global.weight', torch.Size([33278, 400]), torch.float32, True)

('module.linear2global.bias', torch.Size([33278]), torch.float32, True)

Number of trainable parameters=74.781M

where embeddings =41.685M

softmax=13.344M

core=19.752M

### Mini-exp A

sum([len(line.split()) for line in train\_file.readlines()]) = 598

and in fact len(train\_dataloader)= 100, with bsize=2 and seq\_len=3

I may as well raise to bsize=4 and seq\_len=10 to be faster…

Actually, it makes sense to restrict the fragment further, otherwise I have to wait 20+ minutes for 1 mini-experiment.

Next version: 318 tokens

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with 3 layers (1150>1150>400) | 1) The word embedding of the current global (d=300) | batch\_size=4  TBPTT length=8 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
| **Senses LSTM** (L=3, d=1150>1150>400) into linear2Senses FF-NN | learning rate=0.0001 |
| dropout p=0.1 |

INFO : Hyperparameters: \_batchPerSeqlen40\_area32\_lr0.0001\_epochs400

|  |  |
| --- | --- |
| *Epoch* | *Training PPL, globals* |
| 1 | 32847.53 |
| 2 | 21196.15 |
| 5 | 448.27 |
| 10 | 121.72 |
| 50 | 93.52 |
| 100 | 98.79 |
| 150 | 97.92 |
| 175 | 55.55 |
| 200 | 25.45 |
| 225 | 14.74 |
| 250 | 8.53 |
| 275 | 5.05 |
| 300 | 3.85 |
| 350 | 1.95 |
| 400 | 1.49 |

Question: What is the 1.49 instead of 1.0 due to? Is there a global we can not get, or is it just due to the probabilities?

|  |  |
| --- | --- |
| INFO : The top- 5 predicted globals are:  INFO : Word: before ; p=11.59%  INFO : Word: the ; p=10.9%  INFO : Word: steamed ; p=6.64%  INFO : Word: two ; p=4.69%  INFO : Word: followed ; p=3.83% | Label: the next global is: command(from 594)  fail |
| INFO : Word: of ; p=46.71%  INFO : Word: ships ; p=13.25%  INFO : Word: @-@ ; p=11.41%  INFO : Word: fleet ; p=3.23%  INFO : Word: to ; p=2.8% | Label: the next global is: of(from 16)  ok |
| INFO : Word: the ; p=40.96%  INFO : Word: Vice ; p=6.39%  INFO : Word: steaming ; p=6.21%  INFO : Word: Kronprinz ; p=5.58%  INFO : Word: I ; p=4.99% | Label: the next global is: Vice(from 3184)  low, 2nd alternative |
| INFO : The top- 5 predicted globals are:  INFO : Word: Admiral ; p=20.52%  INFO : Word: which ; p=18.75%  INFO : Word: by ; p=9.55%  INFO : Word: south ; p=8.57%  INFO : Word: spotted ; p=4.1% | Label: the next global is: Admiral(from 4118)  ok |
| INFO : Word: David ; p=68.38%  INFO : Word: behind ; p=3.12%  INFO : Word: both ; p=2.81%  INFO : Word: III ; p=2.26%  INFO : Word: while ; p=1.8% | Label: the next global is: David(from 3648)  ok |
| INFO : Word: a ; p=17.46%  INFO : Word: . ; p=13.97%  INFO : Word: and ; p=12.33%  INFO : Word: ships ; p=6.09%  INFO : Word: were ; p=5.64% | Label: the next global is: Beatty(from 11355)  fail |
| INFO : Word: . ; p=20.17%  INFO : Word: König ; p=17.34%  INFO : Word: were ; p=13.67%  INFO : Word: that ; p=6.12%  INFO : Word: Division ; p=3.98% | Label: the next global is: .(from 15)  ok |
| INFO : Word: The ; p=44.68%  INFO : Word: At ; p=8.75%  INFO : Word: Squadron ; p=8.18%  INFO : Word: Battlecruiser ; p=6.87%  INFO : Word: Markgraf ; p=3.6% | Label: the next global is: The(from 83)  ok |
| INFO : Word: opposing ; p=20.8%  INFO : Word: 17 ; p=15.55%  INFO : Word: before ; p=6.2%  INFO : Word: bring ; p=6.02%  INFO : Word: this ; p=5.6% | Label: the next global is: opposing(from 10184)  ok, but low |
| INFO : Word: ships ; p=78.1%  INFO : Word: units ; p=3.65%  INFO : Word: an ; p=2.37%  INFO : Word: encountered ; p=1.87%  INFO : Word: and ; p=1.85% | Label: the next global is: ships(from 3951)  ok |

It’s still reasonably close enough to full overfit, we can assume it would reach it if we had more epochs. We proceed with the main experiment.

## 2LSTMs: Standard LM on WT-2

### Experiment A

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with 3 layers (1150>1150>400) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
| **Senses LSTM** (L=3, d=1150>1150>400) into linear2Senses FF-NN | learning rate=0.0001 |
| dropout p=0.1 (bad choice to use naïve dropout on RNNs) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* | *Validation PPL, globals* | *Validation PPL, senses* | *Validation PPL, multi-senses* |
| 1 | 1520.09 |  |  | 1002.65 |  |  |
| 2 | 879.67 |  |  | 503.68 |  |  |
| 3 | 529.35 |  |  | 383.3 |  |  |
| 4 | 414.48 |  |  | 330.07 |  |  |
| 5 | 350.77 |  |  | 298.65 |  |  |
| 6 | 307.01 |  |  | 276.33 |  |  |
| 7 | 274.15 |  |  | 260.05 |  |  |
| 8 | 247.39 |  |  | 245.82 |  |  |
| 9 | 225.15 |  |  | 234.77 |  |  |
| 10 | 206.06 |  |  | 226.87 |  |  |
| 11 | 189.63 |  |  | 219.64 |  |  |
| 12 | 175.45 |  |  | 214.32 |  |  |
| 13 | 162.97 |  |  | 209.7 |  |  |
| 14 | 152.15 |  |  | 205.95 |  |  |
| 15 | 142.2 |  |  | 204.19 |  |  |
| 16 | 133.49 |  |  | 203.09 |  |  |
| 17 | 125.44 |  |  | 201.99 |  |  |
| 18 | 118.2 |  |  | **200.52** |  |  |
| 19 | 111.58 |  |  | 200.59 |  |  |
| 20 | 105.5 |  |  | 201.39 |  |  |
| 21 | 99.76 |  |  | 202.42 |  |  |
| 22 |  |  |  |  |  |  |
| 23 |  |  |  |  |  |  |
| 24 |  |  |  |  |  |  |
| 25 |  |  |  |  |  |  |

**Considerations**:

A validation perplexity of **200.52** at epoch 18 is a reasonable result, while not ideal.

This means that the current implementation of the RNN still works as intended, it’s not fundamentally wrong/bugged.

However, previous results were better. We pull them from the globals’ document and show them here:

“Using as input signal only the FastText word embedding of the current global,

The LSTM with 3 layers (1024, 1024, 512), bsz=40 x seq\_len=70, lr=10^(-4) gets:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* |
| 34 | 80.65 | 551,203 | 1,109,417 | **188.48** | 48,929 | 106,197 |

The LSTM with 3 layers (1024, 1024, 1024), bsz=40 x seq\_len=35, lr=5e-5 gets:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* |
| 28 | 71.61 | 542,757 | 1,137,084 | **192.92** | 48,004 | 105,263 |

“

Decision:

Reserve 2 GPUs to try out other architectures on WT-2. In particular:

- a new idea, 2Layers, 1024>1024, dropout=0.1

- the old 1024>1024>512 with no dropout, that managed to reach 188 valid-ppl.

In the meantime, now that we know that the results of the RNN on WT-2 are in the reasonable range, use 2 GPUs to execute the next steps on SemCor:

SemCor, globals only, mini-experiment

SemCor, globals only, experiment.

**Reasoning on Dropout**

(<https://medium.com/@bingobee01/a-review-of-dropout-as-applied-to-rnns-72e79ecd5b7b>)

“RNN’s differ from feed-forward -only neural nets in that previous state is fed-back into the network, allowing the network to retain memory of previous states. As such, applying standard dropout to RNN’s tends limits the ability of the networks to retain their memory, hindering their performance.”

“As a way of overcoming performance issues with dropout applied to RNN’s, Zaremba et al. (2014) and Pham et al. (2013) applied dropout only to the non-recurrent connections (Dropout was not applied to the hidden states). “By not using dropout on the recurrent connections, the LSTM can benefit from dropout regularization without sacrificing its valuable memorization ability”

“‘variational dropout’ : repeating “the same dropout mask at each time step for both inputs, outputs, and recurrent layers (drop the same network units at each time step)

“Merity et al., (2017) use DropConnect (Wan et al., 2013) on the recurrent hidden to hidden weight matrices, and variational dropout for all other dropout operations, as well as several other regularization strategies”

I could bring back the DropConnect code I wrote for the AWD-LSTM, but I will try 1024>1024 without any dropout first.

### Model B

INFO : Hyperparameters: \_batchPerSeqlen32\_area32\_lr0.0001\_epochs400

INFO : Model:

INFO : DataParallel(

(module): RNN(

(dropout): Dropout(p=0, inplace=False)

(main\_rnn\_ls): ModuleList(

(0): GRU(300, 1024)

(1): GRU(1024, 1024)

(2): GRU(1024, 512)

)

(linear2global): Linear(in\_features=512, out\_features=33278, bias=True)

)

)

INFO : Parameters:

INFO : ('module.X', torch.Size([138950, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([1024]), torch.float32, False)

('module.embedding\_zeros', torch.Size([1, 300]), torch.float32, False)

('module.memory\_hn', torch.Size([3, 4, 1024]), torch.float32, False)

('module.memory\_cn', torch.Size([3, 4, 1024]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([3, 4, 1024]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([3, 4, 1024]), torch.float32, False)

('module.main\_rnn\_ls.0.weight\_ih\_l0', torch.Size([3072, 300]), torch.float32, True)

('module.main\_rnn\_ls.0.weight\_hh\_l0', torch.Size([3072, 1024]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_ih\_l0', torch.Size([3072]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_hh\_l0', torch.Size([3072]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_ih\_l0', torch.Size([3072, 1024]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_hh\_l0', torch.Size([3072, 1024]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_ih\_l0', torch.Size([3072]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_hh\_l0', torch.Size([3072]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_ih\_l0', torch.Size([1536, 1024]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_hh\_l0', torch.Size([1536, 512]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_ih\_l0', torch.Size([1536]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_hh\_l0', torch.Size([1536]), torch.float32, True)

('module.linear2global.weight', torch.Size([33278, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([33278]), torch.float32, True)

INFO : Number of trainable parameters=71.490M

where embeddings=41.685M, softmax=17.071M, core=12.734M

### Mini-exp B

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=4  TBPTT length=8 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.0001 |
| dropout=none |

|  |  |
| --- | --- |
| Epoch | Training PPL on globals |
| 1 | 32339.56 |
| 2 | 25094.24 |
| 10 | 132.96 |
| 50 | 97.45 |
| 100 | 90.58 |
| 150 | 60.94 |
| 175 | 4.59 |
| 200 | 1.42 |
| 250 | 1.09 |

### Experiment B

note: I hypothesize that I can run a second experiment in parallel on another dataset only **after** the first epoch – when the input and labels have been recorded, and there is no need to access the indices\_table.sql or other elements…

but the lemmatization would still access the DB.

It is necessary to split it in 2, indices\_table\_text and indices\_table\_slc .sql …

Given that the SemCor experiment has priority, I decide to stop the 2 GPUs on WikiText-2, and move on to SemCor with a reasonable 1024>1024>512 architecture. The better experiment on the globals of WT-2 will follow.

No, SemCor may benefit from checking the predictions to verify the DataLoader. Go on with the experiment on WT-2

Too bad the old, non-reimported version of Training.py skips the validation. Must redo.

n: bug on the GRU version, have to adjust

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.00005 |
| dropout=none |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL, globals* |
| 1 | 1417.27 | 678.27 |
| 2 | 664.67 | 442.06 |
| 3 | 480.22 | 360.89 |
| 4 | 392.93 | 316.53 |
| 5 | 339.59 | 289.01 |
| 6 | 301.43 | 269.75 |
| 7 | 271.8 | 255.07 |
| 8 | 247.98 | 242.13 |
| 9 | 228.41 | 232.04 |
| 10 | 211.99 | 225.18 |
| 11 | 197.93 | 218.1 |
| 12 | 185.68 | 212.98 |
| 13 | 174.9 | 208.21 |
| 14 | 165.27 | 203.92 |
| 15 | 156.61 | 201.28 |
| 16 | 148.74 | 198.76 |
| 17 | 141.55 | 196.41 |
| 18 | 134.93 | 193.8 |
| 19 | 128.81 | 192.01 |
| 20 | 123.13 | 190.65 |
| 21 | 117.84 | 189.54 |
| 22 | 112.88 | 188.25 |
| 23 | 108.24 | 188.09 |
| 24 | 103.86 | 187.69 |
| 25 | 99.73 | 187.18 |
| 26 | 95.83 | 187.02 |
| 27 |  |  |
| 28 |  |  |
| 29 |  |  |
| 30 |  |  |

# Standard LM task on SemCor

## Table & Graph

**indices\_table.sql**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| zoning.n.01 | 29132 | 29132 | 29133 | 28077 | 28077 |
| bunched.dummySense.01 | 29133 | 29133 | 29133 | 28077 | 28077 |

**Graph**

INFO : NumExpr defaulting to 8 threads.

INFO : Constructing X, matrix of node features

INFO : X\_senses.shape=torch.Size([43559, 300])

INFO : X\_globals.shape=torch.Size([25693, 300])

INFO : X\_definitions.shape=torch.Size([29133, 300])

INFO : X\_examples.shape=torch.Size([28077, 300])

Graph ranges:

senses: [0, 43559) (with data: 29132)

globals: [43559, 69252)

definitions: [69252, 98385)

examples: [98385, 126462)

INFO : Defining the edges: def, exs

INFO : def\_edges\_se.\_\_len\_\_()=29133 INFO : exs\_edges\_se.\_\_len\_\_()=28077

INFO : Defining the edges: sc

INFO : Reading the sense-labeled corpus, to create the connections between globals and the senses that belong to other words.

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/semcor.xml

INFO : sc\_edges\_with\_external.\_\_len\_\_()=10165

INFO : sc\_edges.\_\_len\_\_()=53724

INFO : get\_edges\_selfloops>max\_sense=43558

INFO : [] INFO : len(edges\_ls)==0

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=53724

INFO : Defining the edges: syn, ant

INFO : syn\_edges.\_\_len\_\_()=31330

INFO : ant\_edges.\_\_len\_\_()=4158

Data(edge\_index=[2, 146422], edge\_type=[146422], node\_types=[126462], num\_relations=[1], x=[126462, 300])

File size of kbGraph.dataobject=156.3MB (156281087)

File size of nodes\_32\_areahops\_1\_graphArea\_matrix.npz=1.4MB (1379932)

**Graph retrieval**

AD.get\_node\_data(grapharea\_matrix, **0**, grapharea\_size=32, features\_mask= (True,False,False))

The adjacent nodes are:

tensor([ 0, 69252, 98386, 98387, 98385, 45017, 53617, 44228]), None, None)

|  |  |
| --- | --- |
| 0 (sense) | a\_bit.r.01 |
| 69252 (definition n. 0) | to a small degree; somewhat |
| 98385, 98386, 98387 (examples n. 0,1,2) | it's a bit warm felt a little better  a trifle smaller |
| 44228, 45017, 53617 (-43559, globals n. 669, 1458, 10058) | a\_bit a\_little a\_trifle |

AD.get\_node\_data(grapharea\_matrix, **52000**, grapharea\_size=32, features\_mask=(True,False,False))

Adjacent nodes:

(tensor([52000, 8360, 8361, 8362, 61894, 54796]), None, None)

|  |  |
| --- | --- |
| 8360, 8361, 8362 (senses) | dominant.a.01  dominant.a.02  dominant.n.01 |
| 52000 (-43559, global n. 8441) | dominant |
| 54796 (global n. 11237) | subordinate |
| 61894 (global n. 18335) | prevailing |

## Mini-experiment on fragment of SemCor

### Model A

INFO : Model:

INFO : DataParallel(

(module): RNN(

(dropout): Dropout(p=0, inplace=False)

(main\_rnn\_ls): ModuleList(

(0): LSTM(300, 1024)

(1): LSTM(1024, 1024)

)

(linear2global): Linear(in\_features=1024, out\_features=25693, bias=True)

)

)

INFO : Parameters:

INFO : ('module.X', torch.Size([126462, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([1024]), torch.float32, False)

('module.embedding\_zeros', torch.Size([1, 300]), torch.float32, False)

('module.memory\_hn', torch.Size([2, 2, 1024]), torch.float32, False)

('module.memory\_cn', torch.Size([2, 2, 1024]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([2, 2, 1024]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([2, 2, 1024]), torch.float32, False)

('module.main\_rnn\_ls.0.weight\_ih\_l0', torch.Size([4096, 300]), torch.float32, True)

('module.main\_rnn\_ls.0.weight\_hh\_l0', torch.Size([4096, 1024]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_ih\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_hh\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_ih\_l0', torch.Size([4096, 1024]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_hh\_l0', torch.Size([4096, 1024]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_ih\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_hh\_l0', torch.Size([4096]), torch.float32, True)

('module.linear2global.weight', torch.Size([25693, 1024]), torch.float32, True)

('module.linear2global.bias', torch.Size([25693]), torch.float32, True)

INFO : Number of trainable parameters=78.102M

where embeddings=37.939M, softmax= 26.335M, core=13.828M

### Mini-exp A

with bsz=2 and seq\_len=3,

len(train\_dataloader)=11, len(valid\_dataloader)=10

It may be necessary to check the predictions, the DataLoad on SemCor appeared to be non-perfect the last times.

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with 2 layers (1024>1024) | 1) The word embedding of the current global (d=300) | batch\_size=2  TBPTT length=3 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.001 |
| dropout=none |

|  |  |
| --- | --- |
| Epoch | Training PPL on globals |
| 1 | 8003.71 |
| 2 | 86.37 |
| 5 | 37.3 |
| 10 | 37.01 |
| 50 | 1.08 |
| 100 | 1.02 |

**Analysis of predictions (aim: check DataLoading)**

Epoch 300: the text, reconstructed from the labels, marking the batch start/end:

\* in which the election was conducted \* <unk> said Friday an investigation of \* Atlanta s recent primary\_election produced “ \* no evidence “ that any irregularities \* took\_place . <unk> The jury further \* said in term end <unk> that \* the <unk> , which had over-all \* charge of the election , “ \* deserves the praise and thanks of \* the <unk> “ for the manner \* in which the election was conducted

This would explain why sometimes the total number of senses changes from one epoch to another: one batch is repeated.

Let us examine what happens in consecutive epochs:

Epoch 1, reconstructed text and batches:

<unk> said Friday an investigation of

Atlanta s recent primary\_election produced “

no evidence “ that any irregularities

took\_place . <unk> The jury further

said in term end <unk> that

the <unk> , which had over-all

charge of the election , “

deserves the praise and thanks of

the <unk> “ for the manner

in which the election was conducted

<unk> said Friday an investigation of

Epoch 2, reconstructed text and batches:

Atlanta s recent primary\_election produced “

no evidence “ that any irregularities

took\_place . <unk> The jury further

said in term end <unk> that

the <unk> , which had over-all

charge of the election , “

deserves the praise and thanks of

the <unk> “ for the manner

in which the election was conducted

<unk> said Friday an investigation of

Atlanta s recent primary\_election produced “

Processing the text starting at a different point is not a problem. It’s the same text, and the same long-range dependencies will be learnt. Actually, if the start point is slightly moved, this introduces an element of variation that can help generalization.

However, it’s non-standard, and I need to skip the batch n.1 or to modify&adjust it.

I decide to adjust **for** b\_idx **in** range(len(train\_dataloader)-1)

Training epoch n.1:

<unk> said Friday an investigation of \* …

\* in which the election was conducted\*

Training epoch n.2:

<unk> said Friday an investigation of \* …

\* in which the election was conducted\*

Now it is fixed.

## Experiment A

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with 2 layers (1024>1024) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=10^(-4) |
| dropout=none |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL, globals* |
| 1 | 1004.74 'correct\_g': 54,041, 'top\_10\_g': 220,196, 'tot\_g': 645,400 | 819.76 'correct\_g': 9213, 'top\_k\_g': 28156, 'tot\_g': 81200 |
| 2 | 594.33 | 650.57 |
| 3 | 467.19 | 596.42 |
| 4 | 402.35 | 554.93 |
| 5 | 357.36 | 524.79 |
| 6 | 324.48 | 499.02 |
| 7 | 297.04 | 483.42 |
| 8 | 275.01 'correct\_g': 121121, 'top\_k\_g': 274860, 'tot\_g': 645400 | 470.16 'correct\_g': 14608, 'top\_k\_g': 33371, 'tot\_g': 81200 |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |
| 15 |  |  |
| 16 |  |  |
| 17 |  |  |
| 18 |  |  |
| 19 |  |  |
| 20 |  |  |
| 21 |  |  |
| 22 |  |  |
| 23 |  |  |
| 24 |  |  |
| 25 |  |  |

While the experiment runs, I can prepare a freeze\_flag for the RNN when I predict senses…

Observation: stopping experiment. Reason: #BadArchitecture.

The Baseline 0: 2GRUs had 284trainPPL, 233ValidPPL @ epoch 5, not 357&524.

3 layers should be better. Using: 800>800>800.

## Experiment B

### Model

INFO : Model:

INFO : DataParallel(

(module): RNN(

(dropout): Dropout(p=0, inplace=False)

(main\_rnn\_ls): ModuleList(

(0): LSTM(300, 800)

(1): LSTM(800, 800)

(2): LSTM(800, 800)

)

(linear2global): Linear(in\_features=800, out\_features=25693, bias=True) ))

INFO : Parameters:

INFO : ('module.X', torch.Size([126462, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([800]), torch.float32, False)

('module.embedding\_zeros', torch.Size([1, 300]), torch.float32, False)

('module.memory\_hn', torch.Size([3, 2, 800]), torch.float32, False)

('module.memory\_cn', torch.Size([3, 2, 800]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([3, 2, 800]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([3, 2, 800]), torch.float32, False)

('module.main\_rnn\_ls.0.weight\_ih\_l0', torch.Size([3200, 300]), torch.float32, True)

('module.main\_rnn\_ls.0.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_ih\_l0', torch.Size([3200, 800]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_ih\_l0', torch.Size([3200, 800]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('module.linear2global.weight', torch.Size([25693, 800]), torch.float32, True)

('module.linear2global.bias', torch.Size([25693]), torch.float32, True)

INFO : Number of trainable parameters=72.298M

where embeddings=37.939M, softmax= 20.580M, core=13.779M

### Mini-exp B

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with **3** layers (800>800>800) | 1) The word embedding of the current global (d=300) | batch\_size=2  TBPTT length=3 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.001 |
| dropout=none |

|  |  |
| --- | --- |
| Epoch | Training PPL on globals |
| 1 | 14819.45 |
| 2 | 124.24 |
| 5 | 37.07 |
| 10 | 36.87 |
| 50 | 36.7 |
| 100 | 36.6 |
| 125 | 20.36 |
| 150 | 10.96 |
| 175 | 1.89 |
| 200 | 1.07 |

### Experiment B

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with **3** layers (800>800>800) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.0001 |
| dropout=none |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL, globals* |
| 1 | 1047.88 | 929.41 |
| 2 | 764.56 | 803.49 |
| 3 | 561.83 | 638.41 |
| 4 | 458.6 | 599.91 |
| 5 | 411.59 | 554.26 |
| 6 | 348.63 | 542.52 |
| 7 | 324.45 | 526.51 |
| 8 | 302.71 | 517.78 |
| 9 | 265.26 | 508.91 |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |
| 15 |  |  |
| 16 |  |  |
| 17 |  |  |
| 18 |  |  |
| 19 |  |  |

Error: #BadResults

Let us try the old 1024>1024>512 architecture, that GRU operating on 21.9K globals managed to obtain ~185 validation perplexity.

## Experiment C

### Model

INFO : Model:

INFO : DataParallel(

(module): RNN(

(dropout): Dropout(p=0, inplace=False)

(main\_rnn\_ls): ModuleList(

(0): LSTM(300, 1024)

(1): LSTM(1024, 1024)

(2): LSTM(1024, 512)

)

(linear2global): Linear(in\_features=512, out\_features=25693, bias=True)

)

)

INFO : Parameters:

INFO : ('module.X', torch.Size([126462, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([1024]), torch.float32, False)

('module.embedding\_zeros', torch.Size([1, 300]), torch.float32, False)

('module.memory\_hn', torch.Size([3, 3, 1024]), torch.float32, False)

('module.memory\_cn', torch.Size([3, 3, 1024]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([3, 3, 1024]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([3, 3, 1024]), torch.float32, False)

('module.main\_rnn\_ls.0.weight\_ih\_l0', torch.Size([4096, 300]), torch.float32, True)

('module.main\_rnn\_ls.0.weight\_hh\_l0', torch.Size([4096, 1024]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_ih\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_hh\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_ih\_l0', torch.Size([4096, 1024]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_hh\_l0', torch.Size([4096, 1024]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_ih\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_hh\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_ih\_l0', torch.Size([2048, 1024]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_hh\_l0', torch.Size([2048, 512]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_ih\_l0', torch.Size([2048]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_hh\_l0', torch.Size([2048]), torch.float32, True)

('module.linear2global.weight', torch.Size([25693, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([25693]), torch.float32, True)

INFO : Number of trainable parameters=68.097M,

where embeddings=37.938M, softmax=13.180M, core=16.979M

### Mini-exp C

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with **3** layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=3  TBPTT length=3 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.0001 |
| dropout=none |

|  |  |
| --- | --- |
| Epoch | Training PPL on globals |
| 1 | 25605.21 |
| 2 | 24340.06 |
| 5 | 765.19 |
| 10 | 51.87 |
| 50 | 33.67 |
| 100 | 33.45 |
| 150 | 33.38 |
| 200 | 33.36 |
| 250 | 32.72 |
| 275 | 9.3 |
| 300 | 3.26 |
| 350 | 1.41 |

### Experiment C

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with **3** layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.0001 |
| dropout=none |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL, globals* |
| 1 | 1078.07 | 920.93 |
| 2 | 794.26 | 944.78 |
| 3 | 774.57 | 839.96 |
| 4 | 581.55 | 650.03 |
| 5 | 467.55 | 590.52 |
| 6 | 407.1 | 548.06 |
| 7 | 365.03 | 521.39 |
| 8 | 334.51 | 496.88 |
| 9 | 310.52 | 480.57 |
| 10 | 289.88 | 475.81 |
| 11 | 272.67 | 469.21 |
| 12 | 258.31 | 462.41 |
| 13 | 244.95 | 451.61 |
| 14 | 233.27 | **445.73** |
| 15 | 222.85 | 446.91 |
| 16 | 213.19 | 446.62 |
| 17 | 204.91 | 448.06 |
| 18 | 196.88 | 450.08 |
| 19 | 182.93 | 459.88 |

## Observations

### On the results

445 Validation Perplexity is a very bad result.

However, the architecture appears to work correctly on WT-2.

Last time we tried the globals’ task on WikiText-2, at [Experiment A](#_Experiment_A), even a flawed architecture of 1150>1150>400 with Dropout=0.1 (it shouldn’t have had dropout) reached 200 Valid-PPL.

What causes the current bad performance on SemCor?

From the analysis of the predictions, we know that we send the words in the batches in the correct way (we even adjusted the len(train\_dataloader) to -1 to avoid the repetition of the last batch).

The previous experiment at [Experiment: GRUs on SemCor](#_Experiment:_GRUs_on) reached 185 Valid-PPL, although it operated on a restricted vocabulary of 21988 globals

(with (linear2global): Linear(in\_features=512, out\_features=21988, bias=True))

What things were different then?

* There was no dummySense. This should have no impact
* There was no lemmatized\_form column in the vocabulary. Again, this is used only when processing senses
* There was a different pre-processing for the vocabulary, 21988 vs. 25693

### Reviewing the input loading and vocabulary

Currently:

TextDataset > \_\_getitem\_\_():

self.current\_token\_tpl, self.next\_token\_tpl = NI.get\_tokens\_tpls(self.next\_token\_tpl, self.generator,   
 self.senseindices\_db\_c, self.vocab\_h5, self.grapharea\_matrix,  
 self.last\_sense\_idx, self.first\_idx\_dummySenses )

get\_tokens\_tpls():

convert\_tokendict\_to\_tpl()

convert\_tokendict\_to\_tpl():

word = VocabUtils.process\_word\_token(token\_dict) *# html.unescape*

etc.

**def** process\_word\_token(token\_dict):  
 token\_text = html.unescape(str(token\_dict[**'surface\_form'**]))

**return** token\_text

The oldest version (1):

|  |
| --- |
| def process\_slc\_token(token\_dict): |
|  | token\_text = html.unescape(token\_dict['surface\_form']) |
|  |  |
|  | if token\_text == token\_text.upper(): # if ALL CAPITALS -> must lowercase |
|  | token\_text = token\_text.lower() |
|  | token\_text = token\_text.replace('\_', ' ') # we keep phrases, but we should write 'Mr. Barcus' not Mr.\_Barcus |
|  |  |
|  | token\_latinorgreek = replace\_nonLatinGreek\_words([token\_text])[0] |
|  | token\_final = replace\_numbers([token\_latinorgreek])[0] |
|  | return token\_final |

The next version (2):

|  |  |
| --- | --- |
| def process\_word\_token(token\_dict): | |
|  | token\_text = html.unescape(token\_dict['surface\_form']) | |
|  | token\_text = convert\_symbols(token\_text) | |
|  |  | |
|  | # if token\_text == token\_text.upper(): # if ALL CAPITALS -> must lowercase | |
|  | # token\_text = token\_text.lower() # we are not lowercasing anymore, otherwise 'USA'->'usa' | |
|  |  | |
|  | # token\_text = token\_text.replace('\_', ' ') # we keep phrases, but we should write 'Mr. Barcus' not Mr.\_Barcus | |
|  |  | |
|  | # token\_latinorgreek = replace\_nonLatinGreek\_words([token\_text])[0] | |
|  | # token\_final = replace\_numbers([token\_latinorgreek])[0] | |
|  |  | |
|  | return token\_text | |

What happens if we:

* replace the word-tokens with non-Latin and non-Greek characters, such as: 匹, 枚, マギカ , etc. with <unk>?  
  Nothing. We still get 25693 tokens in the vocabulary. Evidently words formed of those characters can only be found in WT-2.
* replace phrases like go\_on with “go on”?  
  We still have [25693 rows x 4 columns], but now with ‘primary election’ instead of ‘primary\_election’. Since input reading also uses token processing, we will still match to it, but nothing changes.
* Replace numbers. All decimals, together with all numbers that are not 1 digit (basic) or 4 digits (years), get replaced with <num>  
  Now |V|=25439, and the <num> token has frequency 3507.

Still, none of these modifications brings the number of globals to 21988.

Maybe, it’s the fact that now we build the vocabulary from all splits: training, validation, test.

In Vocabulary.py > build\_vocabulary\_dict\_from\_senselabeled():

slc\_split\_names = [Utils.TRAINING, Utils.VALIDATION, Utils.TEST]

How was it originally?

It was

build\_vocabulary\_from\_senselabeled(slc\_split\_name)

used in:

get\_vocabulary\_df():

|  |
| --- |
| if senselabeled\_or\_text: |
|  | vocabulary = build\_vocabulary\_from\_senselabeled(slc\_split\_name) |

Probably because I built the vocabulary from the training set alone, so it was easier.

Trying again: no VocabUtils token processing, only the training split as base:

|V|=22235, close enough.

I can state that it is rational to build the vocabulary only from Training and Validation, ignoring the Test set.

This would give us |V|= 24122

Not a large enough difference from 25439 to justify the jump in perplexity from 185 to 445…

What if there is an error in the pipeline when building data&graph for the Sense-Labeled Corpus?

From the graph statistics and graph retrieval at [Table & Graph](#_Table_&_Graph), I know that the nodes are connected correctly. And if there were an error in the vectors, it would show up in WT2…

We consider that WikiText-2 has ~2 million tokens, and Penn-TreeBank has ~979K training tokens that are heavily pre-processed (words were lower-cased, numbers were replaced with N, newlines were replaced with <eos>, and all other punctuation was removed. The vocabulary is the most frequent 10k words with the rest of the tokens replaced by an <unk> token.)

Maybe the original vocabulary of 21988 was entirely lowercased, and thus had an easier Language Modeling task even on a small dataset such as SemCor, that contains only ~650K tokens?

We check the size of the lowercased vocabulary that comes only from the training set of SemCor: |**V|= 21988**

In fact, the earlier result of Valid-PPL=185 was obtained on the **lowercased, <num>-processed** SemCor.

### Next steps

Step 0: re-run the pipeline for SemCor, and try to replicate exactly the old result with 185 Valid-PPL. This would demonstrate that the architecture still works as intended.

Step 1: try a 1024>1024>512 architecture on WT-2, with no dropout, to try to bring the Valid-PPL well under 200.

Step 2: gather enough sense-labeled data to obtain a corpus of size comparable to WT-2, to try to obtain a similar Valid-PPL.

I need to review the other Sense-Labeled Corpuses.

Step 3: continue adjusting the code and running mini-experiments, to prepare the freezing mechanism.

## Replicating the old result (lowercased, <num>, Vocab from Training set)

### Graph

INFO : Constructing X, matrix of node features

INFO : X\_senses.shape=torch.Size([**38611**, 300])

INFO : X\_globals.shape=torch.Size([**21988**, 300])

INFO : X\_definitions.shape=torch.Size([25987, 300])

INFO : X\_examples.shape=torch.Size([26003, 300])

INFO : Defining the edges: def, exs

INFO : def\_edges\_se.\_\_len\_\_()=25987

INFO : exs\_edges\_se.\_\_len\_\_()=26003

INFO : Defining the edges: sc

INFO : Reading the sense-labeled corpus, to create the connections between globals and the senses that belong to other words.

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/semcor.xml

INFO : sc\_edges\_with\_external.\_\_len\_\_()=9021

INFO : sc\_edges.\_\_len\_\_()=47632

INFO : get\_edges\_selfloops>max\_sense=38610

INFO : []

INFO : len(edges\_ls)==0

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=47632

INFO : Defining the edges: syn, ant

INFO : syn\_edges.\_\_len\_\_()=26222

INFO : ant\_edges.\_\_len\_\_()=3780

### Model

INFO : Model:

INFO : DataParallel(

(module): RNN(

(dropout): Dropout(p=0, inplace=False)

(main\_rnn\_ls): ModuleList(

(0): LSTM(300, 1024)

(1): LSTM(1024, 1024)

(2): LSTM(1024, 512)

)

(linear2global): Linear(in\_features=512, out\_features=21988, bias=True))

)

INFO : Parameters:

INFO : ('module.X', torch.Size([112589, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([1024]), torch.float32, False)

('module.embedding\_zeros', torch.Size([1, 300]), torch.float32, False)

('module.memory\_hn', torch.Size([3, 40, 1024]), torch.float32, False)

('module.memory\_cn', torch.Size([3, 40, 1024]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([3, 40, 1024]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([3, 40, 1024]), torch.float32, False)

('module.main\_rnn\_ls.0.weight\_ih\_l0', torch.Size([4096, 300]), torch.float32, True)

('module.main\_rnn\_ls.0.weight\_hh\_l0', torch.Size([4096, 1024]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_ih\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_hh\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_ih\_l0', torch.Size([4096, 1024]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_hh\_l0', torch.Size([4096, 1024]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_ih\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_hh\_l0', torch.Size([4096]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_ih\_l0', torch.Size([2048, 1024]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_hh\_l0', torch.Size([2048, 512]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_ih\_l0', torch.Size([2048]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_hh\_l0', torch.Size([2048]), torch.float32, True)

('module.linear2global.weight', torch.Size([21988, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([21988]), torch.float32, True)

INFO : Number of trainable parameters=62.034M

where embeddings=33.777M, softmax=11.279M, core=16.978M

### Experiment

INFO : Hyperparameters: \_batchPerSeqlen1400\_area32\_lr0.0001\_epochs50

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL, globals* |
| 1 | 988.25 | 516.65 |
| 2 | 743.27 | 517.23 |
| 3 | 724.36 | 469.34 |
| 4 | 545.12 | 368.56 |
| 5 | 431.86 | 319.88 |
| 6 | 373.88 | 297.48 |
| 7 | 338.42 | 280.63 |
| 8 | 311.1 | 265.78 |
| 9 | 288.65 | 254.51 |
| 10 | 269.24 | 245.3 |
| 11 | 253.36 | 242.29 |
| 12 | 239.82 | 234.75 |
| 13 | 227.98 | 227.13 |
| 14 | 217.39 | 225.0 |
| 15 | 208.13 | 224.03 |
| 16 | 199.43 | 221.93 |
| 17 | 191.8 | 220.98 |
| 18 | 184.99 | 220.37 |
| 19 | 178.25 | 219.99 |
| 20 | 171.98 | 220.88 |
| 21 | 166.26 | 220.63 |
| 22 | 160.97 | 220.19 |
| 23 | 155.94 | **219.4** |
| 24 | 151.52 | 220.86 |
| 25 | 147.16 | 222.91 |
| 26 | 142.94 | 225.5 |

219.4 Valid PPL for SemCor: in reasonable range. Reminds us of the fact that on small datasets GRUs tend to perform better than LSTMs, as they have fewer parameters.

# Gathering more Sense-Labeled Data

### Review of Sense-Labeled Corpuses

Our objective is to have a volume of data >= WT-2, i.e. a number of tokens >=2millions.

From my own notes on the matter, in developer diary n.1:

“UFSAC: Unification of Sense Annotated Corpora and Tools”

“Our work consists in gathering all English corpora sense annotated with WordNet, and convert all of them to a unified format…”

The work contains several corpora that may be of use:

* *SemCor*, the subset of the Brown Corpus  
  [we know it has a wn\_30 key, that we use]
* The *OMSTI* (One Million Sense-Tagged Instances) (Taghipour and Ng, 2015), a corpus of approximately one million words sense annotated with WordNet 3.0
* The *MASC* (Manually Annotated Sub-Corpus), the version given in the article of (Yuan et al., 2016), annotated with NOAD but with corresponding WordNet 3.0 sense keys
* The *Ontonotes 5.*0 (Hovy et al., 2006), annotated with WordNet 3.0.
* The corpora of the WSD evaluation campaigns *SemEval-SensEval*: SensEval 2 (using WordNet 1.7), SensEval 3 (WN 1.7.1), SemEval 2007 (WN 2.1), SemEval 2013 (WN 3.0) and SemEval 2015 (WN 3.0).

On the UFSAC File format:

organized as: Corpus > Document > Paragraph > Sentence > Word

The statistics of the corpuses: again, not 100% of words are annotated:

|  |  |  |
| --- | --- | --- |
| Corpus | Words – total | Words – annotated |
| *SemCor* | 778,587 | 229,517 |
| *OMSTI* | 35,843,024 | 920,794 |
| *MASC* | 596,333 | 114,950 |
| *Ontonotes 5.*0 | 435,340 | 52,263 |
| *SemEval + SensEval* | etc. | etc. |

Taking into account that we have a 80-10-10 split, and that we use annotations for WordNet 3.0 (since it is the version that we access through the nltk tool),

Example 1:

SemCor=622800 + MASC=477040 + OntoNotes5.0=348240 = 1,448,080

Example 2:

(SemCor + MASC + OntoNotes5.0)= 1,448,080. + OMSTI=28,674,416 = 30,122,496

However, OntoNotes is in a complex format accessed via tools, it can not be extracted as easily as SemCor and MASC. We must look elsewhere.

I decide to use part of OMSTI (using 100% would be slower), plus MASC and SemCor.

Verifying the XML files,

SemCor has 778587 words (80%=622800)

MASC has 585353 words (80%=468282)

subset\_omsti\_aa.xml has approximately 1.5 million words (80%=1.2 mln)

## Running the pipeline, graph

I move the files and data on the Cheetah server and obtain the graph again, while Lambda is busy.

Statistics are also reported in the [Preliminary information (Datasets, graphs)](#_Preliminary_information_(Datasets,) section at the start of this document.

### Number of documents / sentences in the corpuses

INFO : Organizing subcorpus at: TextCorpuses/My Sense-Labeled Corpus/semcor.xml

INFO : The corpus at TextCorpuses/My Sense-Labeled Corpus/**semcor**.xml has **352** documents

INFO : Training dataset will contain: **281** documents , Validation dataset will contain: **35** documents , Test dataset will contain: **36** documents

INFO : Organizing subcorpus at: TextCorpuses/My Sense-Labeled Corpus/masc.xml

INFO : The corpus at TextCorpuses/My Sense-Labeled Corpus/**masc**.xml has 1 documents

INFO : Training dataset will contain: **25407** sentences , Validation dataset will contain: **3176** sentences , Test dataset will contain: **3176** sentences

INFO : Organizing subcorpus at: TextCorpuses/My Sense-Labeled Corpus/subset\_omsti\_aa.xml

INFO : The corpus at TextCorpuses/My Sense-Labeled Corpus/subset\_omsti\_aa.xml has 1 documents

INFO : Training dataset will contain: **83703** sentences , Validation dataset will contain: **10463** sentences , Test dataset will contain: **10463** sentences

INFO : \*\*\* Creating vocabulary at Vocabulary/vocabulary\_of\_globals.h5

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/semcor.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/masc.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/subset\_omsti\_aa.xml

INFO : Removing from the vocabulary words with frequency **< 2**

INFO : \*\*\* The vocabulary was created. Number of words= **52781**\*\*\*

### Graph

INFO : Constructing X, matrix of node features

INFO : X\_senses.shape=torch.Size([**73177**, 300])

INFO : X\_globals.shape=torch.Size([**52781**, 300])

INFO : X\_definitions.shape=torch.Size([35850, 300])

INFO : X\_examples.shape=torch.Size([31976, 300])

Graph ranges:

senses=[0, 73177) (senses with data: 35850)

globals=[73177, 125958)

definitions=[125958, 161808)

examples=[161808, 193784)

INFO : Defining the edges: def, exs

INFO : def\_edges\_se.\_\_len\_\_()=35850

INFO : exs\_edges\_se.\_\_len\_\_()=31976

INFO : Defining the edges: sc

INFO : Reading the sense-labeled corpus, to create the connections between globals and the senses that belong to other words.

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/semcor.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/masc.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/subset\_omsti\_aa.xml

INFO : sc\_edges\_with\_external.\_\_len\_\_()=13642

INFO : sc\_edges.\_\_len\_\_()=86819

INFO : get\_edges\_selfloops>max\_sense=73176

INFO : []

INFO : len(edges\_ls)==0

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=86819

INFO : Defining the edges: syn, ant

INFO : syn\_edges.\_\_len\_\_()=57814

INFO : ant\_edges.\_\_len\_\_()=5940

Data(edge\_index=[2, 218399], edge\_type=[218399], node\_types=[193784], num\_relations=[1], x=[193784, 300])

Files:

kbGraph.dataobject

nodes\_32\_areahops\_1\_graphArea\_matrix.npz

note: the values may be wrong if we have applied the lowercase…  
I didn’t apply the lowercase, but I mistakenly left in <num> processing. Undue. Rerun pipeline.

Also creating a Fragment of the Dataset, taking 1 sentence from each of the 3 corpuses.

### Graph retrieval

Temporary, for the sake of the mini-experiments on the new SLC corpus to implement the freezing mechanism.

Recalling the graph ranges:

senses=[0, 73177) (senses with data: 35850)

globals=[73177, 125958)

definitions=[125958, 161808)

examples=[161808, 193784)

AD.get\_node\_data(grapharea\_matrix, **0**, grapharea\_size=32, features\_mask=(True,False,False))

> (tensor([ 0, 125958, 107246]), None, None)

Node n: 0

Adjacent nodes:

|  |  |  |
| --- | --- | --- |
| 0 | sense | 1900s.n.01 |
| 125958 | 0th definition | the decade from 1900 to 1909 |
| 107246 | 34069th global | 1900s |

Node n: 32000

(tensor([ 32000, 157958, 190276, 190277, 190278, 190274, 190275, 90569]), None, None)

Adjacent nodes:

|  |  |  |
| --- | --- | --- |
| 32000 | sense | tattered.s.01 |
| 90569 | 17392nd global | tattered |
| 157958 | 32000th definition | worn to shreds; or wearing torn or ragged clothing |
| [190274, 190275, 190276, 190277, 190278] | 28466th,…, 28470th example | a man in a tattered shirt  the tattered flag  tied up in tattered brown paper  a tattered barefoot boy  a tatterdemalion prince |

## SLC: Experiment A

### Model

INFO : DataParallel(

(module): RNN(

(dropout): Dropout(p=0, inplace=False)

(main\_rnn\_ls): ModuleList(

(0): GRU(300, 1024)

(1): GRU(1024, 1024)

(2): GRU(1024, 512)

)

(linear2global): Linear(in\_features=512, out\_features=54937, bias=True))

)

INFO : Parameters:

INFO : ('module.X', torch.Size([199554, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([1024]), torch.float32, False)

('module.embedding\_zeros', torch.Size([1, 300]), torch.float32, False)

('module.memory\_hn', torch.Size([3, 40, 1024]), torch.float32, False)

('module.memory\_cn', torch.Size([3, 40, 1024]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([3, 40, 1024]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([3, 40, 1024]), torch.float32, False)

('module.main\_rnn\_ls.0.weight\_ih\_l0', torch.Size([3072, 300]), torch.float32, True)

('module.main\_rnn\_ls.0.weight\_hh\_l0', torch.Size([3072, 1024]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_ih\_l0', torch.Size([3072]), torch.float32, True)

('module.main\_rnn\_ls.0.bias\_hh\_l0', torch.Size([3072]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_ih\_l0', torch.Size([3072, 1024]), torch.float32, True)

('module.main\_rnn\_ls.1.weight\_hh\_l0', torch.Size([3072, 1024]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_ih\_l0', torch.Size([3072]), torch.float32, True)

('module.main\_rnn\_ls.1.bias\_hh\_l0', torch.Size([3072]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_ih\_l0', torch.Size([1536, 1024]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_hh\_l0', torch.Size([1536, 512]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_ih\_l0', torch.Size([1536]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_hh\_l0', torch.Size([1536]), torch.float32, True)

('module.linear2global.weight', torch.Size([54937, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([54937]), torch.float32, True)

INFO : Number of trainable parameters=100.782M

where embeddings=59.866M, softmax=28.182M, core=12.734M

note: we log:

Training epoch n.1:

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/masc.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/semcor.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/subset\_omsti\_aa.xml

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/masc.xml

Does that mean that we are repeating the last subcorpus?

### Exp A

INFO : Hyperparameters: \_batchPerSeqlen1400\_area32\_lr5e-05\_epochs50

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with **3** layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.00005 |
| dropout=none |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL, globals* |
| 1 | 673.66 | 409.54 |
| 2 | 288.57 | 277.38 |
| 3 | 212.31 | 232.49 |
| 4 | 175.85 | 208.6 |
| 5 | 152.94 | 192.77 |
| 6 | 136.41 | 181.22 |
| 7 | 123.65 | 172.03 |
| 8 | 113.37 | 164.73 |
| 9 | 104.83 | 158.83 |

### Observations:

The perplexity appears to be very good, 158.8, possibly lower than WikiText-2.

However, we need to:

# Freezing standard LM and then proceeding to Senses

## Notes & ideas

The idea is:

train a standard LM, obtaining the best possible validation perplexity. The senses’ RNN should be inactive / frozen / unused at 0.

Then, freeze: the main RNN, and also the embeddings in the graph (otherwise I would be moving them according to the senses’ task, and I would lose in performance).

Finally, activate the senses’ RNN and train on the sense-prediction task, that now includes the dummySenses.

Question 1: how to “keep aside” the senses’ RNN until needed?

I could put the loss of senses artificially at 0…

Or I could modify the predict\_senses flag, keeping it for the init() creation of the RNN, and dropping it for the forward() in the training loop iterations…

In the initialization function:

**if** predict\_senses:  
 self.**senses\_rnn\_ls** = torch.nn.ModuleList(  
 [getattr(torch.nn, self.model\_type)(input\_size=self.concatenated\_input\_dim **if** i == 0 **else** n\_hid\_units,

hidden\_size=n\_hid\_units **if** i == n\_layers - 1 **else** n\_hid\_units, num\_layers=1) **for** i **in** range(n\_layers)]) *# 400*

**if** predict\_senses:  
 self.**linear2senses** = torch.nn.Linear(in\_features=n\_hid\_units, *# 400* out\_features=self.last\_idx\_senses, bias=**True**)

In the forward() call:

**if** self.predict\_senses:  
 senses\_rnn\_output = **None** input = batch\_input\_signals  
 **for i in range(self.n\_layers)**:  
 **layer\_rnn = self.senses\_rnn\_ls[i]**  
 layer\_rnn.flatten\_parameters()  
 **if** self.model\_type.upper() == **"LSTM"**:  
 senses\_rnn\_output, (hidden\_i, cells\_i) = \  
 layer\_rnn(input, select\_layer\_memory(self, i, layer\_rnn))  
 update\_layer\_memory(self, i, layer\_rnn, hidden\_i, cells\_i)  
 **else**: *# GRU* senses\_rnn\_output, hidden\_i = \  
 layer\_rnn(input, select\_layer\_memory(self, i, layer\_rnn))  
 update\_layer\_memory(self, i, layer\_rnn, hidden\_i)  
  
 senses\_rnn\_output = self.dropout(senses\_rnn\_output)  
 input = senses\_rnn\_output  
  
 senses\_rnn\_output = senses\_rnn\_output.reshape(distributed\_batch\_size \* seq\_len, senses\_rnn\_output.shape[2])  
  
 logits\_sense = self.linear2senses(senses\_rnn\_output)  
  
 **predictions\_senses = tfunc.log\_softmax(logits\_sense, dim=1)  
else**:  
 **predictions\_senses = torch.tensor([0] \* self.batch\_size \* seq\_len).to(CURRENT\_DEVICE)**

In Training.py:

In training\_setup():

model = RNNs.RNN(**"LSTM"**, graph\_dataobj, grapharea\_size, grapharea\_matrix,  
 globals\_vocabulary\_df,  
 include\_globalnode\_input, include\_sensenode\_input,predict\_senses,  
 batch\_size=batch\_size, n\_layers=2, n\_hid\_units=1024, dropout\_p=0)

(taken care of above)

In train\_loop():

**if** model\_forParameters.predict\_senses:  
 sum\_epoch\_loss\_sense = sum\_epoch\_loss\_sense + loss\_sense.item() \* num\_batch\_sense\_tokens  
 epoch\_senselabeled\_tokens = epoch\_senselabeled\_tokens + num\_batch\_sense\_tokens  
 sum\_epoch\_loss\_multisense = sum\_epoch\_loss\_multisense + loss\_multisense.item() \* num\_batch\_multisense\_tokens  
 epoch\_multisense\_tokens = epoch\_multisense\_tokens + num\_batch\_multisense\_tokens  
 loss = loss\_global + loss\_sense

**else**:  
 loss = loss\_global

In compute\_model\_loss():

**if** model\_forParameters.predict\_senses:  
 loss\_all\_senses = tfunc.nll\_loss(predictions\_senses, batch\_labels\_all\_senses, ignore\_index=-1)  
 loss\_multi\_senses = tfunc.nll\_loss(predictions\_senses, batch\_labels\_multi\_senses, ignore\_index=-1)  
**else**:  
 loss\_all\_senses = torch.tensor(0)  
 loss\_multi\_senses = torch.tensor(0)

In evaluation():

including\_senses = model\_forParameters.predict\_senses

**for** b\_idx **in** range(len(evaluation\_dataloader)):

**…**

**if** including\_senses:  
 sum\_eval\_loss\_sense = sum\_eval\_loss\_sense + loss\_sense.item() \*   
 num\_batch\_sense\_tokens  
 evaluation\_senselabeled\_tokens = evaluation\_senselabeled\_tokens +   
 num\_batch\_sense\_tokens  
 sum\_eval\_loss\_multisense = sum\_eval\_loss\_multisense +   
 loss\_multisense.item() \* num\_batch\_multisense\_tokens  
 evaluation\_multisense\_tokens = evaluation\_multisense\_tokens +   
 num\_batch\_multisense\_tokens

**if** including\_senses:  
 senses\_evaluation\_loss = sum\_eval\_loss\_sense / evaluation\_senselabeled\_tokens  
 multisenses\_evaluation\_loss = sum\_eval\_loss\_multisense /   
 evaluation\_multisense\_tokens

So we should be able to set predict\_senses=True for the init(), predict\_senses=False for the 1st part and then True again.

At that point,

(comment 1: “When you set the requires\_grad=False, the parameters won’t be updated during backward pass.”)

(comment 2: “I would recommend to create a new optimizer (or have two before) because many optimizers have a momentum term that may cause changes in parameters even when the gradients are zero.”)

## Implementation

Adding the with\_freezing flag parameter to train\_loop.

Examining the predict\_senses occurrences and how to modify it:

It can be kept in training\_setup: so the layers of the rnn\_senses and the linear2Senses FF-NN can be created, even if they are not used in the 1st phase.

If the current validation PPL is worse than the best validation PPL by >0.01 point, we proceed:

examining the named parameters of the model, the ones of the main rnn are named 'main\_rnn\_ls.0.weight\_ih\_l0',

'main\_rnn\_ls.0.weight\_hh\_l0',

'main\_rnn\_ls.0.bias\_ih\_l0',

'main\_rnn\_ls.0.bias\_hh\_l0',

'main\_rnn\_ls.1.weight\_ih\_l0', …,

'main\_rnn\_ls.2.bias\_hh\_l0'.

We should set requires\_grad=False

And at the same time, set model\_forParameters.predict\_senses =True in order to “activate” that part of the model.

To freeze the embeddings / node states, we should set requires\_grad=False for the matrix X.

While the experiment of the Standard LM on SemCor goes on on the lambda machine, I check: can I execute the mini-experiments for the freezing mechanism on the Cheetah server?

Yes.

### Mini-experiment with freezing on SemCor, v. 1.0

|  |  |  |  |
| --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* |
| 1 | 12552.44 | 1.0 | 1.0 |
| 2 | 122.81 |  |  |
| 5 | 37.33 |  |  |
| 10 | 36.83 |  |  |
| 50 | 36.64 |  |  |
| 100 | 36.61 | 1.0 | 1.0 |
| 150 | 8.93 | 1.0 | 1.0 |
| 151 | 21.86 | 24178.33 | 77.67 |
| 152 | 11.86 | 189.72 | 12.58 |

What could this be due to?

* + - * I am not freezing linear2Globals, so part of the pipeline for the globals can still get modified
      * moreover, the perplexity on senses-of-globals-with-multiple-senses still really needs checking

### Mini-experiment with freezing on SemCor, v. 1.1

|  |  |  |  |
| --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* |
| 1 | 13415.22 | 1.0 | 1.0 |
| 2 | 124.95 | 1.0 | 1.0 |
| 3 | 45.41 | 1.0 | 1.0 |
| 4 | 36.58 | 23791.8 | 78.21 |
| 5 | 37.27 | 204.25 | 12.8 |

Let us examine what happens with the model’s parameters:

In Epoch 1:

INFO : ('X', torch.Size([126462, 300]), torch.float32, True)

('select\_first\_indices', torch.Size([800]), torch.float32, False)

('embedding\_zeros', torch.Size([1, 300]), torch.float32, False)

('memory\_hn', torch.Size([3, 2, 800]), torch.float32, False)

('memory\_cn', torch.Size([3, 2, 800]), torch.float32, False)

('memory\_hn\_senses', torch.Size([3, 2, 800]), torch.float32, False)

('memory\_cn\_senses', torch.Size([3, 2, 800]), torch.float32, False)

('main\_rnn\_ls.0.weight\_ih\_l0', torch.Size([3200, 300]), torch.float32, True)

('main\_rnn\_ls.0.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('main\_rnn\_ls.0.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('main\_rnn\_ls.0.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('main\_rnn\_ls.1.weight\_ih\_l0', torch.Size([3200, 800]), torch.float32, True)

('main\_rnn\_ls.1.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('main\_rnn\_ls.1.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('main\_rnn\_ls.1.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('main\_rnn\_ls.2.weight\_ih\_l0', torch.Size([3200, 800]), torch.float32, True)

('main\_rnn\_ls.2.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('main\_rnn\_ls.2.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('main\_rnn\_ls.2.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('senses\_rnn\_ls.0.weight\_ih\_l0', torch.Size([3200, 300]), torch.float32, True)

('senses\_rnn\_ls.0.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('senses\_rnn\_ls.0.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('senses\_rnn\_ls.0.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('senses\_rnn\_ls.1.weight\_ih\_l0', torch.Size([3200, 800]), torch.float32, True)

('senses\_rnn\_ls.1.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('senses\_rnn\_ls.1.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('senses\_rnn\_ls.1.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('senses\_rnn\_ls.2.weight\_ih\_l0', torch.Size([3200, 800]), torch.float32, True)

('senses\_rnn\_ls.2.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('senses\_rnn\_ls.2.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('senses\_rnn\_ls.2.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('linear2global.weight', torch.Size([25693, 800]), torch.float32, True)

('linear2global.bias', torch.Size([25693]), torch.float32, True)

('linear2senses.weight', torch.Size([43559, 800]), torch.float32, True)

('linear2senses.bias', torch.Size([43559]), torch.float32, True)

In Epoch 4:

INFO : ('**X**', torch.Size([126462, 300]), torch.float32, **False**)

('select\_first\_indices', torch.Size([800]), torch.float32, False)

('embedding\_zeros', torch.Size([1, 300]), torch.float32, False)

('memory\_hn', torch.Size([3, 2, 800]), torch.float32, False)

('memory\_cn', torch.Size([3, 2, 800]), torch.float32, False)

('memory\_hn\_senses', torch.Size([3, 2, 800]), torch.float32, False)

('memory\_cn\_senses', torch.Size([3, 2, 800]), torch.float32, False)

(**'main\_rnn\_ls**.0.weight\_ih\_l0', torch.Size([3200, 300]), torch.float32, **False**)

('main\_rnn\_ls.0.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, False)

('main\_rnn\_ls.0.bias\_ih\_l0', torch.Size([3200]), torch.float32, False)

('main\_rnn\_ls.0.bias\_hh\_l0', torch.Size([3200]), torch.float32, False)

('main\_rnn\_ls.1.weight\_ih\_l0', torch.Size([3200, 800]), torch.float32, False)

('main\_rnn\_ls.1.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, False)

('main\_rnn\_ls.1.bias\_ih\_l0', torch.Size([3200]), torch.float32, False)

('main\_rnn\_ls.1.bias\_hh\_l0', torch.Size([3200]), torch.float32, False)

('main\_rnn\_ls.2.weight\_ih\_l0', torch.Size([3200, 800]), torch.float32, False)

('main\_rnn\_ls.2.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, False)

('main\_rnn\_ls.2.bias\_ih\_l0', torch.Size([3200]), torch.float32, False)

(**'main\_rnn\_ls**.2.bias\_hh\_l0', torch.Size([3200]), torch.float32, **False**)

(**'senses\_rnn\_ls.**0.weight\_ih\_l0', torch.Size([3200, 300]), torch.float32, **True**)

('senses\_rnn\_ls.0.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('senses\_rnn\_ls.0.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('senses\_rnn\_ls.0.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('senses\_rnn\_ls.1.weight\_ih\_l0', torch.Size([3200, 800]), torch.float32, True)

('senses\_rnn\_ls.1.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('senses\_rnn\_ls.1.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

('senses\_rnn\_ls.1.bias\_hh\_l0', torch.Size([3200]), torch.float32, True)

('senses\_rnn\_ls.2.weight\_ih\_l0', torch.Size([3200, 800]), torch.float32, True)

('senses\_rnn\_ls.2.weight\_hh\_l0', torch.Size([3200, 800]), torch.float32, True)

('senses\_rnn\_ls.2.bias\_ih\_l0', torch.Size([3200]), torch.float32, True)

(**'senses\_rnn\_ls**.2.bias\_hh\_l0', torch.Size([3200]), torch.float32, **True**)

('**linear2global**.weight', torch.Size([25693, 800]), torch.float32, **False**)

(**'linear2globa**l.bias', torch.Size([25693]), torch.float32, **False**)

('**linear2senses**.weight', torch.Size([43559, 800]), torch.float32, **True**)

('linear2senses.bias', torch.Size([43559]), torch.float32, True)

INFO : Number of trainable parameters=48669959

Maybe its due to the momentum?

It seems that the changes in the last version are less relevant.

What happens if I do not recreate the optimizer?

Still the same.

And the micro-differences are not due to the optimizer, they are there with SGD or with Adam.

### Mini-experiment with freezing on SemCor, v. 1.2

|  |  |  |  |
| --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* |
| 1 | 14341.38 | 1.0 | 1.0 |
| 2 | 133.97 | 1.0 | 1.0 |
| 3 | 46.1 | 1.0 | 1.0 |
| 4 | 32.42 | 21037.7 | 76.17 |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |

## Implementation, II

On the new larger Sense-Labeled Corpus (actually on the temporary version that by mistake has <num> processing in the vocabulary. It does not make a difference when we are operating on a mini-fragment for testing purposes).

After we arrive at the freezing point, we set after\_freezing\_flag=True, and from this point on loss=loss\_sense, loss\_global is not taken into account anymore.

### Mini-experiment with freezing on SLC, v. 1.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL (same dataset, i.e. end-of-epoch), globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* |
| 1 | 46685.21 | 5812.2 | 1.0 | 1.0 |
| 2 | 2247.96 | 342.82 |  |  |
| 3 | 202.15 | 79.71 |  |  |
| 4 | 120.39 | 109.52 | 57512.14 | 26.47 |
| 5 | 89.76 | 80.9 | 3084.55 | 12.72 |

This makes no sense. I am using loss=loss\_sense after the freezing point…

We need to create an artificial, fixed matrix of embeddings (or even 2 different matrices X) and examine whether this persists. The networks should not be sharing parameters.

## Gradually building a model

### Basic NN

We build a small NN model that operates on the same input and output labels.

**Version 1.0**, characteristics:

* operating on 2 distinct, random matrices of embeddings, embs\_A and embs\_B, for the 2 tasks.
* There are no GRUs, the architecture for the 2 tasks is just:   
  2 FF-NNs with 1 Linear layer > 2 FF-NNs (linear2Globals, linear2Senses)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL (same dataset, i.e. end-of-epoch), globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* |
| 1 | 113822.32 | 49945.73 | 1.0 | 1.0 |
| 2 | 39601.31 | 22493.28 |  |  |
| 3 | 15906.56 | 8416.76 |  |  |
| 4 | 6506.86 | 3968.08 |  |  |
| 5 | 2678.44 | 1578.62 | 1.0 | 1.0 |
| Freezing the weights in the standard LM, activating senses' prediction. | | | | |
| 6 | 1530.4 | 1400.91 | 120091.27 | 31.5 |
| 7 | 1530.4 | 1632.99? | 40984.65 | 22.7 |
| 8 | 1530.4 | 1410.32 | 16282.92 | 17.23 |
| 9 | 1530.4 | 1648.31 | 6592.94 | 13.18 |
| 10 | 1530.4 | 1578.62 | 2686.81 | 10.1 |

The Training PPL is finally fixed, as intended. But why does the Validation Perplexity change?

**Version 1.1**, identical to v.1.0 but logging the validation predictions:

Freezing at Epoch 3, on a total of 5 epochs:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL (same dataset, i.e. end-of-epoch), globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* |
| 3 | 10280.32 | 6083.3 | 1.0 | 1.0 |
| 4 | 5883.69 | 6222.33 | 93134.75 | 30.38 |
| 5 | 5883.69 | 6717.51 | 32240.8 | 22.06 |

The first 5 Validation predictions, @ epoch 4:

|  |  |
| --- | --- |
| Label: the next global is: matter(from 1804) | INFO : Word: Tennessee ; p=0.04%  INFO : Word: Channel ; p=0.04% |
| Label: the next global is: of(from 5) | INFO : Word: Eliminating ; p=0.06%  INFO : Word: demand-led ; p=0.03% |
| Label: the next global is: <unk>(from 52780) | INFO : Word: Atlanta ; p=0.18%  INFO : Word: course ; p=0.08% |
| Label: the next global is: ,(from 26) | INFO : Word: . ; p=0.1%  INFO : Word: said ; p=0.09% |
| Label: the next global is: of(from 5) | INFO : Word: Pleasant ; p=0.04%  INFO : Word: attempting ; p=0.04% |

Hypothesis:

I still have the dataloader error that requests 1 batch more than the number stored in the dataset, since I currently have:

**for** b\_idx **in** range(len(train\_dataloader)-1)

and

**for** b\_idx **in** range(len(evaluation\_dataloader))

Adding -1, see what changes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL (same dataset, i.e. end-of-epoch), globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* |
| 3 | 11355.56 | 6484.84 | 1.0 | 1.0 |
| 4 | 6484.84 | 6484.84 | 99565.56 | 31.56 |
| 5 | 6484.84 | 6484.84 | 34659.66 | 22.87 |

OK

### Adding the RNNs