[Current Next Steps 4](#_Toc50110887)

[University side 4](#_Toc50110888)

[Ordbogen side 6](#_Toc50110889)

[Preliminary information (Datasets, graphs) 7](#_Toc50110890)

[SemCor – current 7](#_Toc50110891)

[SemCor stats (80-10-10, lowercased) 7](#_Toc50110892)

[Graph (min\_freq=2, with dummySense) 7](#_Toc50110893)

[SemCor + MASC 8](#_Toc50110894)

[Stats 8](#_Toc50110895)

[Graph 8](#_Toc50110896)

[SemCor – other settings 8](#_Toc50110897)

[Graph (vocab. min. frequency=2, no dummy Sense, <num>processed) 8](#_Toc50110898)

[Graph (vocab. min. frequency=2, with dummy Sense) 9](#_Toc50110899)

[SemCor + MASC + OMSTI(300MB) 10](#_Toc50110900)

[Number of documents / sentences 10](#_Toc50110901)

[Graph 10](#_Toc50110902)

[Recap of experiments made on SemCor, and new directions 12](#_Toc50110903)

[Verifying the RNN setup on the standard LM task 15](#_Toc50110904)

[Table & Graph – WT-2 15](#_Toc50110905)

[Mini-experiment on fragment of WT-2 16](#_Toc50110906)

[Model 16](#_Toc50110907)

[Mini-exp A 17](#_Toc50110908)

[2LSTMs: Standard LM on WT-2 19](#_Toc50110909)

[Experiment A 19](#_Toc50110910)

[Model B 21](#_Toc50110911)

[Mini-exp B 21](#_Toc50110912)

[Experiment B 22](#_Toc50110913)

[Standard LM task on SemCor 24](#_Toc50110914)

[Table & Graph 24](#_Toc50110915)

[Mini-experiment on fragment of SemCor 25](#_Toc50110916)

[Model 29](#_Toc50110917)

[Observations 33](#_Toc50110918)

[On the results 33](#_Toc50110919)

[Reviewing the input loading and vocabulary 33](#_Toc50110920)

[Next steps 36](#_Toc50110921)

[Replicating the old result (lowercased, <num>, Vocab from Training set) 37](#_Toc50110922)

[Graph 37](#_Toc50110923)

[Model 37](#_Toc50110924)

[Experiment 38](#_Toc50110925)

[Gathering more Sense-Labeled Data 39](#_Toc50110926)

[Review of Sense-Labeled Corpuses 39](#_Toc50110927)

[Running the pipeline, graph 40](#_Toc50110928)

[Number of documents / sentences in the corpuses 40](#_Toc50110929)

[Graph 41](#_Toc50110930)

[Graph retrieval 42](#_Toc50110931)

[SLC: Experiment A 43](#_Toc50110932)

[Model 43](#_Toc50110933)

[Exp A 44](#_Toc50110934)

[Observations: 44](#_Toc50110935)

[Freezing standard LM and then activating Senses 45](#_Toc50110936)

[Notes & ideas 45](#_Toc50110937)

[Implementation 47](#_Toc50110938)

[Implementation, II 50](#_Toc50110939)

[Gradually building a model 51](#_Toc50110940)

[Basic NN 51](#_Toc50110941)

[Adding the RNNs 52](#_Toc50110942)

[Adding the embeddings 52](#_Toc50110943)

[Full RNN (RNNs, embeddings, GATs) 53](#_Toc50110944)

[Lowercased SemCor, attempt 1 54](#_Toc50110945)

[Stats & Graph 54](#_Toc50110946)

[Experiment A: standard LM 55](#_Toc50110947)

[Model A 55](#_Toc50110948)

[Mini-exp A 55](#_Toc50110949)

[Exp A 56](#_Toc50110950)

[Lowercased SemCor + MASC, part I 58](#_Toc50110951)

[Preliminary 58](#_Toc50110952)

[Graph stats 58](#_Toc50110953)

[Graph retrieval 59](#_Toc50110954)

[Input signal: word embedding. Task: Standard LM 60](#_Toc50110955)

[Input signal: word embedding. Senses: standard GRU 62](#_Toc50110956)

[Debugging the multi-senses Perplexity 65](#_Toc50110957)

[Mini-Exp B (with freezing) 65](#_Toc50110958)

[Exp B (with freezing) 67](#_Toc50110959)

[Comparison between the 2 base cases, with and without freezing 69](#_Toc50110960)

[Perplexity 69](#_Toc50110961)

[Correct predictions 70](#_Toc50110962)

[Observations 70](#_Toc50110963)

[Input signals: word embedding, global nodestate. Senses: standard GRU 71](#_Toc50110964)

[POTENTIAL MAJOR ISSUE 71](#_Toc50110965)

[LOWERCASED SEMCOR + MASC, attempt II 72](#_Toc50110966)

[Preliminary 72](#_Toc50110967)

[Graph stats 72](#_Toc50110968)

[Graph retrieval 72](#_Toc50110969)

[Node & input loading 73](#_Toc50110970)

[Input signal: word embedding. Senses: standard GRU 74](#_Toc50110971)

[Model A 74](#_Toc50110972)

[Mini-exp A 75](#_Toc50110973)

[Exp A 75](#_Toc50110974)

[Input signal: word embedding. Senses: Freezing, standard GRU 76](#_Toc50110975)

[Mini-exp B 76](#_Toc50110976)

[Lowercased SemCor, attempt 2 78](#_Toc50110977)

[Corpus & graph 78](#_Toc50110978)

[Experiment A 78](#_Toc50110979)

[Model 78](#_Toc50110980)

[Trying to debug through mini-experiments 79](#_Toc50110981)

[Experiment on SemCor 80](#_Toc50110982)

[Check: Experiments on WikiText-2 81](#_Toc50110983)

[Standard LM, no lowercasing 81](#_Toc50110984)

[Further developments: loading saved model, updating senses methods 82](#_Toc50110985)

[Loading a saved model 82](#_Toc50110986)

[Saving & loading, mini-experiment on SemCor 83](#_Toc50110987)

[Mini-testing: Saving, loading, predicting senses, freezing 83](#_Toc50110988)

[Final Experiments 86](#_Toc50110989)

[WikiText-2 (standard LM) 86](#_Toc50110990)

[Experiment 86](#_Toc50110991)

[Simple GRU for senses 87](#_Toc50110992)

[Directly on SemCor 87](#_Toc50110993)

[On SemCor, input signal with GAT – version 1 91](#_Toc50110994)

[Revising and debugging the GNN input signal 92](#_Toc50110995)

[Parameter budget 93](#_Toc50110996)

[Graph creation 95](#_Toc50110997)

[GRUs + GNN input signal # GAT debug 96](#_Toc50110998)

[Mini-experiments 96](#_Toc50110999)

[Experiment: Simple GRUs + GAT input (same parameter budget) on SemCor 97](#_Toc50111000)

[GAT Debug … 100](#_Toc50111001)

[Debugging the graph (senses with dots) 104](#_Toc50111002)

[Simple GRUs + GNN input signal 105](#_Toc50111003)

# Current Next Steps

## University side

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 **186.4** 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 still ok

status: done. The freezing mechanism now works, and without any oscillation, as we can see in [Full RNN (RNNs, embeddings, GATs)](#_Full_RNN_(RNNs,).

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

Step 7a:

Set up and run, indicatively, the following experiments:

* On SemCor alone, and on pretrained on WT-2+SemCor
* With and without freezing
* **With and without GAT-global-nodestate**
* Standard GRU for globals and senses
* SelectK(1,5,10,50), with and without freezing
* Attention mechanism for senses

This would mean a total of: 2x2x2x3 = 24 experiments

(alternatives that work better get priority when it comes to computational resources)

Step 7b:

Start the write-up of the paper. Proceed with the structure of the experiments and the description of the methods, and also abstract + background

## Ordbogen side

Information 1:

The Desktop version is currently put aside and not developed. The version currently developed is the web browser add-on version.

The GrammaTip team is also working on a text editor.

Information 2:

You can (ideally) reliably convert to & fro Pytorch and Tensorflow (see instrument: Neural Network Exchange)

Information 3:

Instrument for deployment that could be used: MLFlow.

# Preliminary information (Datasets, graphs)

## SemCor – current

### SemCor stats (80-10-10, lowercased)

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

Training tokens: 646,038

Validation tokens: 80,760

|Vocabulary|= 22,782

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

INFO : Constructing X, matrix of node features

INFO : X\_senses.shape=torch.Size([**41206**, 100]) # senses with data: 30445

INFO : X\_globals.shape=torch.Size([**22782**, 100])

INFO : X\_definitions.shape=torch.Size([30446, 100])

INFO : X\_examples.shape=torch.Size([27921, 100])

INFO : E\_embeddings.shape=torch.Size([22782, 300])

Graph ranges: [0, senses, 41206, globals, 63988, defs, 94434, examples, 122355]

INFO : Defining the edges: def, exs

INFO : def\_edges\_se.\_\_len\_\_()=30446

INFO : exs\_edges\_se.\_\_len\_\_()=27921

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\_\_()=7044

INFO : sc\_edges.\_\_len\_\_()=48250

INFO : get\_edges\_selfloops>max\_sense=41205

INFO : []

INFO : len(edges\_ls)==0

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()= 48250

INFO : Defining the edges: syn, ant

INFO : syn\_edges.\_\_len\_\_()= 19804 (fewer, it was 24272…(?)(is this relevant or only a modification from an older version)

INFO : ant\_edges.\_\_len\_\_()=4070

INFO : Pre-computing and saving graphArea matrix, with area\_size=32

~~Data(edge\_index=[2,~~ **~~136630~~**~~], edge\_type=[136630], node\_types=[122355], num\_relations=[1], x=[~~**~~122355~~**~~, 300])~~

Data(edge\_index=[2, **130491**], edge\_type=[130491], node\_types=[122355], num\_relations=[1], x=[**122355**, 100]) ; file size =~151MB

## SemCor + MASC

### Stats

### Graph

## SemCor – other settings

### Graph (vocab. min. frequency=2, no dummy Sense, <num>processed)

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 | 92.13 | 186.47 |
| 28 | 88.62 | **186.37** |
| 29 | 85.29 | 186.68 |
| 30 | 82.12 | 186.54 |
| 31 | 79.11 | 187.51 |
| 32 | 76.23 | 188.84 |

Best validation perplexity on WT-2: 186.4

# 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 activating 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL (same dataset, i.e. end-of-epoch), globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* |
| 3 | 24039.98 | 17576.88 | 1.0 | 1.0 |
| 4 | 16926.93 | 16848.63 | 74414.85 | 28.82 |
| 5 | 16848.63 | 16848.63 | 69725.17 | 28.24 |

Adding the RNN, it stabilizes after 1 epoch, so it’s still OK.

### Adding the embeddings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL (same dataset, i.e. end-of-epoch), globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* |
| 3 | 41949.14 | 35618.65 | 1.0 | 1.0 |
| 4 | 35162.18 | **35249.46** | 73450.24 | 28.7 |
| 5 | **35249.46** | **35249.46** | 70230.19 | 28.33 |

This version is actually ok. Rather than check line-by-line the original RNN, I add its other features (GATs, etc.) to this one.

### Full RNN (RNNs, embeddings, GATs)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL (same dataset, i.e. end-of-epoch), globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* |
| 3 | 41692.49 | 34909.69 | 1.0 | 1.0 |
| 4 | 34385.52 | **34496.78** | 71459.98 | 28.33 |
| 5 | **34496.77** | **34496.78** | 68011.08 | 27.92 |

done, see to the other tasks.

(although it must be noted that the GATs are not currently in use here)

# Lowercased SemCor, attempt 1

## Stats & Graph

|Vocabulary|=

Taken from the training split alone. **Lowercased**, only words with frequency **> 2**.

No other preprocessing

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

INFO : Training dataset will contain: documents , Validation dataset will contain: documents , Test dataset will contain: documents

Splits: 80% - 10% - 10%

Training tokens:

Validation tokens:

Test tokens:

**Graph**

INFO : X\_senses.shape=torch.Size([**41206**, 300])

INFO : X\_globals.shape=torch.Size([**22782**, 300])

INFO : X\_definitions.shape=torch.Size([30446, 300])

INFO : X\_examples.shape=torch.Size([27921, 300])

Graph ranges:

senses: [0,41206) # [0,30445) senses with data

globals: [41206, 63988)

definitions: [63988, 94434)

examples: [94434, 122355)

INFO : Defining the edges: def, exs

[63988, 94433) -> [0, 30445)

INFO : def\_edges\_se.\_\_len\_\_()=30446

INFO : exs\_edges\_se.\_\_len\_\_()=27921

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\_\_()=7213

INFO : sc\_edges.\_\_len\_\_()=48419

INFO : get\_edges\_selfloops>max\_sense=41205

INFO : []

INFO : len(edges\_ls)==0

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=48419

INFO : syn\_edges.\_\_len\_\_()=19804

INFO : ant\_edges.\_\_len\_\_()=4070

## Experiment A: standard LM

### Model A

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=22782, bias=True)

))

INFO : Parameters:

INFO : ('module.X', torch.Size([122355, 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([22782, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([22782]), torch.float32, True)

INFO : Number of trainable parameters=61.127M,

where embeddings=36.707M , softmax=11.687M, core=12.733M

### Mini-exp A

|  |  |  |
| --- | --- | --- |
| **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=2  TBPTT length=3 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.001 |
| dropout=none |

|  |  |
| --- | --- |
| Epoch | Training PPL on globals |
| 1 |  |
| 5 |  |
| 10 |  |
| 50 |  |
| 100 |  |
| 150 |  |
| 200 |  |
| 250 |  |
| 275 |  |
| 300 |  |
| 350 |  |

### Exp A

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

|  |  |  |
| --- | --- | --- |
| **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=0.0001 |
| dropout=none |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL, globals* |
| 1 | 532.76 | 414.46 |
| 2 | 291.85 | 316.54 |
| 3 | 229.79 | 279.71 |
| 4 | 197.33 | 256.46 |
| 5 | 175.1 | 241.66 |
| 6 | 159.01 | 232.42 |
| 7 | 146.74 | 226.81 |
| 8 | 136.83 | 223.29 |
| 9 | 128.41 | 221.07 |
| 10 | 121.02 | 219.97 |
| 11 | 114.36 | 219.51 |
| 12 | 108.27 | **219.4** |
| 13 | 102.65 | 219.68 |
| 14 | 97.45 | 220.44 |
| 15 | 92.6 | 221.64 |
| 16 |  |  |
| 17 |  |  |
| 18 |  |  |
| 19 |  |  |
| 20 |  |  |

note: as seen further on, having 219.4 instead of 185.95 may be due to reading all tokens that are not entirely lowercased as <unk>

I previously got 185.95 from the 21K vocabulary that came from the training set alone, instead of training + validation, and with the non-standard <num> processing.

This makes me inclined to consider:

If I put together SemCor + MASC, I would have

|  |  |  |
| --- | --- | --- |
| *SemCor* | 778,587 | 229,517 |
| *MASC* | 596,333 | 114,950 |

=1,374,920 tokens in the corpus (80%=1,099,936 for training) and I could have better-looking perplexity results on a still small & reasonably fast corpus.

\*\*\*\*\*\* Issue: \*\*\*\*\*\*\*

I seemed to have repeated generators on the SLC subcorpuses,

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

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

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

but not in a mini-experiment,

Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/fragment\_masc.xml

Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/Training/fragment\_semcor.xml

?

Time to debug with VNCViewer and remote desktop…

Issue in DataLoading, self.counter.

When we ask for the len\_ the first time, the self.counter is at 1143008. – which should be ok, it means we have read both SemCor and MASC.

Training epoch n.1:

full\_fpaths=['TextCorpuses/My Sense-Labeled Corpus/Training/masc.xml', 'TextCorpuses/My Sense-Labeled Corpus/Training/semcor.xml']

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

Maybe it’s just simpler to concatenate the 2 training (and validation, and test) files into a SLC.xml

# Lowercased SemCor + MASC, part I

## Preliminary

### Graph stats

INFO : Constructing X, matrix of node features

INFO : X\_senses.shape=torch.Size([**50993**, 300])

INFO : X\_globals.shape=torch.Size([**30564**, 300])

INFO : X\_definitions.shape=torch.Size([35221, 300])

INFO : X\_examples.shape=torch.Size([30378, 300])

Graph ranges:

senses=[0,50993) # with data: [0,35220)

globals=[50993, 81557)

definitions=[81557, 116778)

examples=[116778, 147156)

INFO : def\_edges\_se.\_\_len\_\_()=35221

INFO : exs\_edges\_se.\_\_len\_\_()=30378

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 : sc\_edges\_with\_external.\_\_len\_\_()=8170

INFO : sc\_edges.\_\_len\_\_()=59163

INFO : get\_edges\_selfloops>max\_sense=50992

INFO : []

INFO : len(edges\_ls)==0

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=59163

INFO : syn\_edges.\_\_len\_\_()=24224

INFO : ant\_edges.\_\_len\_\_()=4754

INFO : Pre-computing and saving graphArea matrix, with area\_size=32

kbgraph.dataobject, file size=181.5 MB

### Graph retrieval

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

Node n: 0

(tensor([ 0, 81557, 66875]), None, None)

Adjacent nodes:

|  |  |
| --- | --- |
| 0, 0th sense | 1870s.n.01 |
| 81557, 0th definition | the decade from 1870 to 1879 |
| 66875, 15882nd global | 1870s |

AD.get\_node\_data(grapharea\_matrix, **60000**, grapharea\_size=32, features\_mask=(True,False,False))

Node n: 60000

(tensor [20532, 24458, 24459, 24460, 24461, 24462, 24463, 51349, 52373, 54582, 59982, 60000, 66417])

|  |  |  |
| --- | --- | --- |
| 20532 | sense | motivate.v.01 (# must have been associated through the corpus) |
| 24458, 24459, 24460, 24461, 24462, 24463 | sense | prompt.n.01  prompt.n.02  prompt.s.01  prompt.s.02  prompt.v.02  prompt.v.03 |
| 51349 | 356th global | suggestion |
| 52373 | 1380th global | immediate |
| 54582 | 3589th global | remind |
| 59982 | 8989th global | inspire |
| 60000 | 9007th global | **prompt** |
| 66417 | 15424th global | motivate |

## Input signal: word embedding. Task: Standard LM

**Model A**

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=30564, bias=True)

))

INFO : Parameters:

INFO : ('module.X', torch.Size([147156, 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([30564, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([30564]), torch.float32, True)

INFO : Number of trainable parameters=72.56M

where embeddings=44.146M, softmax=15.679M, core=12.735M

**Exp A**

|  |  |  |
| --- | --- | --- |
| **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=0.0001 |
| dropout=none |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL, globals* |
| 1 | 459.14 | 420.21 |
| 2 | 259.06 | 291.96 |
| 3 | 193.17 | 254.43 |
| 4 | 164.28 | 237.3 |
| 5 | 146.27 | 227.1 |
| 6 | 133.02 | 219.99 |
| 7 | 122.34 | 215.02 |
| 8 | 113.32 | 211.69 |
| 9 | 105.51 | 210.36 |
| 10 | 98.67 | **210.34** |
| 11 | 92.64 | 210.79 |
| 12 | 87.28 | 211.64 |
| 13 | 82.42 | 212.39 |

We reach 210.3 Valid-PPL @ epoch 10 on SemCor+MASC with a vocabulary of 30564, as opposed to 219.4 @ epoch 12 on SemCor alone with a vocabulary of 22782.

Consideration: the learning rate of 10^-4 is too high, we repeat the experiment using 5\*10^-5.

**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=40  TBPTT length=35 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.000**05** |
| dropout=none |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL, globals* |
| 1 | 555.07 | 507.98 |
| 2 | 345.1 | 391.86 |
| 3 | 263.99 | 315.63 |
| 4 | 218.9 | 281.26 |
| 5 | 191.66 | 259.96 |
| 6 | 173.31 | 245.85 |
| 7 | 160.1 | 236.4 |
| 8 | 149.92 | 229.63 |
| 9 | 141.6 | 224.67 |
| 10 | 134.5 | 221.1 |
| 11 | 128.28 | 218.52 |
| 12 | 122.72 | 216.63 |
| 13 | 117.68 | 215.19 |
| 14 | 113.06 | 214.02 |
| 15 | 108.8 | 213.29 |
| 16 | 104.85 | 212.73 |
| 17 | 101.19 | 212.37 |
| 18 | 97.79 | 212.19 |
| 19 | 94.62 | **212.17** |
| 20 | 91.64 | 212.28 |

It appears that 5e05 is actually slightly worse, so we return to lr=10^-4.

## Input signal: word embedding. Senses: standard GRU

**Model A**

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)

)

(senses\_rnn\_ls): ModuleList(

(0): GRU(300, 1024)

(1): GRU(1024, 1024)

(2): GRU(1024, 512)

)

(linear2global): Linear(in\_features=512, out\_features=30564, bias=True)

(linear2senses): Linear(in\_features=512, out\_features=50993, bias=True)

)

)

INFO : Parameters:

INFO : ('module.X', torch.Size([147156, 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.senses\_rnn\_ls.0.weight\_ih\_l0', torch.Size([3072, 300]), torch.float32, True)

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

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

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

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

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

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

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

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

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

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

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

('module.linear2global.weight', torch.Size([30564, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([30564]), torch.float32, True)

('module.linear2senses.weight', torch.Size([50993, 512]), torch.float32, True)

('module.linear2senses.bias', torch.Size([50993]), torch.float32, True)

INFO : Number of trainable parameters=111.452M,

where embeddings=44.147M, softmax\_globals= 15.679M, softmax\_senses=26.159M, core=25.467M (~12.73M x2)

**Exp A**

|  |  |  |
| --- | --- | --- |
| **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 |
| Senses prediction: **GRU** with **3** layers (1024>1024>512) |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.0001 |
| dropout=none |

Important note: the Perplexity on globals that have multiple senses must be debugged.

As usual, top\_k==top\_10

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Globals* | | *Senses* | | *Multi-senses* | |
|  | Train | Valid | Train | Valid | Train | Valid |
| 1 | 502.74  'correct\_g': 192081, 'top\_k\_g': 465989, 'tot\_g': 1142400 | 524.58 'correct\_g': 26658, 'top\_k\_g': 62556, 'tot\_g': 151200 | 681.98 'correct\_all\_s': 169027, 'top\_k\_all\_s': 444407, 'tot\_all\_s': 1142400 | **647.14** 'correct\_all\_s': 25795, 'top\_k\_all\_s': 61438, 'tot\_all\_s': 151200 | 26.38 'correct\_multi\_s': 0, 'top\_k\_multi\_s': 13541, 'tot\_multi\_s': 408520 | 10524.44 'correct\_multi\_s': 0, 'top\_k\_multi\_s': 2236, 'tot\_multi\_s': 53600 |
| 2 | 414.07  'correct\_g': 192565, 'top\_k\_g': 474197, 'tot\_g': 1142400 | 537.72  'correct\_g': 26658, 'top\_k\_g': 62302,  'tot\_g': 151200 | 532.82  'correct\_all\_s': 169109, 'top\_k\_all\_s': 452265, 'tot\_all\_s': 1142400 | 667.97  'correct\_all\_s': 25795, 'top\_k\_all\_s': 61438, 'tot\_all\_s': 151200 | 24.73  'correct\_multi\_s': 0, 'top\_k\_multi\_s': 14599, 'tot\_multi\_s': 408520 | 11324.86  'correct\_multi\_s': 0, 'top\_k\_multi\_s': 2236, 'tot\_multi\_s': 53600} |
| 3 | 353.64 | 373.13 | 530.1 | 665.78 |  |  |
| 4 | 231.91 | 287.19 | 515.43  'correct\_all\_s': 169021, 'top\_k\_all\_s': 453250, 'tot\_all\_s': 1142400 | **661.51**  'correct\_all\_s': 25795, 'top\_k\_all\_s': 61513, 'tot\_all\_s': 151200 | 24.25  'correct\_multi\_s': 0, 'top\_k\_multi\_s': 14225, 'tot\_multi\_s': 408520 | 11360.57  'correct\_multi\_s': 0, 'top\_k\_multi\_s': 2126, 'tot\_multi\_s': 53600 |
| 5 | 182.83 | 259.88 | 490.2 | 663.88 |  |  |
| 6 | 158.39 | 246.8 | 465.7 | 673.63 |  |  |
| 7 | 142.35 | 239.53 | 443.29 | 685.59 |  |  |
| 8 | 130.33 | 234.9 | 423.2 | 699.45 |  |  |
| 9 | 120.56 | 231.76 | 404.62 | 709.98 |  |  |
| 10 | 112.4 | 229.94 | 387.62 | 720.65 |  |  |
| 11 | 105.45 | 228.54 | 372.14 | 729.6 |  |  |
| 12 | 99.35 | 227.24 | 357.77 | 738.43 |  |  |
| 13 | 93.93 | 225.59 | 344.35 | 746.05 |  |  |
| 14 | 89.09 | 223.85 | 331.92 | 750.54 |  |  |
| 15 | 84.72 | 222.34 | 320.28 | 754.94 |  |  |
| 16 | 80.66 | **221.2**  **'correct\_g': 38403, 'top\_k\_g': 75869, 'tot\_g': 151200** | 308.98 | 763.88  'correct\_all\_s': 25784, **'top\_k\_all\_s': 61962,** 'tot\_all\_s': 151200 |  | 15462.58  'correct\_multi\_s': 0, 'top\_k\_multi\_s': 1998, 'tot\_multi\_s': 53600 |
| 17 | 76.86 | 222.56 | 298.0 | 779.78 |  |  |
| 18 | 73.47 | 223.61 | 288.04 | 792.36 |  |  |
| 19 | … |  |  |  |  |  |
| 20 |  |  |  |  |  |  |

Observations:

* Executing the 2 tasks together causes worse performance for the globals, 221.2 Valid-PPL compared to 210.  
  The embeddings can be moved by both tasks.
* The senses’ task is harder: 647 Valid-PPL is extremely high.  
  We recall that we are using a parallel GRU to predict the next word at sense-granularity level.

### Debugging the multi-senses Perplexity

Let us examine what happened in the previous experiment:

Training, end of epoch 1. Global step n.816. Time = 9367.28. The training losses are:

INFO : Losses: Globals loss=6.22 Sense loss=**6.52**   
Loss on multi-senses=**6.52** Total loss=12.745

INFO : Perplexity: Globals perplexity=502.74

Perplexity on all senses=681.98 Perplexity on multi-senses=26.38

This means that the loss is correct, is only the perplexity that is wrong…

Checking the current status:

Training, end of epoch 1. Global step n.4. Time = 45.87. The training losses are:

epoch\_sumlosses\_tpl=(

sum\_epoch\_loss\_global=41.21, sum\_epoch\_loss\_sense=433.83, sum\_epoch\_loss\_multisense=119.24)

epoch\_step=4; num\_steps\_with\_sense=40; num\_steps\_with\_multisense=40;

num\_steps\_with\_multisense may be wrong

The bug may have been located:

epoch\_numsteps\_tpl =

epoch\_step, epoch\_senselabeled\_tokens, epoch\_senselabeled\_tokens

Turning it into:

epoch\_numsteps\_tpl =

epoch\_step, epoch\_senselabeled\_tokens, epoch\_multisense\_tokens

Now, we get a more reasonable

INFO : Losses: Globals loss=10.31

Sense loss=10.86 Loss on multi-senses=10.86 Total loss=21.169

INFO : Perplexity: Globals perplexity=29927.61

Perplexity on all senses=52188.02 Perplexity on multi-senses=50871.4

### Mini-Exp B (with freezing)

The freezing condition here is artificial: when the globals’ PPL is <5.

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals: **Main GRU** with **3** layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| Senses: **GRU** with **3** layers (1024>1024>512) |
| **Freezing:** in 1st part, update only globals. Then freeze global’s network + embeddings, and update senses | grapharea=32, hops=1 |
| learning rate=0.0001 |
| dropout=none |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Globals* | | *Senses* | | *Multi-senses* | |
|  | Train | Valid | Train | Valid | Train | Valid |
| 1 | 30087.08 | 28623.78 | 1.0 |  | 1.0 |  |
| 2 | 27737.78 | 26117.58 |  |  |  |  |
| 50 | 13.5 | 13.44 |  |  |  |  |
| 100 | 12.6 | 12.53 |  |  |  |  |
| 125 | 8.3 | 7.7 |  |  |  |  |
| 135 | 4.83 | 4.67 | 1.0 | 1.0 | 1.0 | 1.0 |
| 136 | 5.26 | 6.87 | 57673.13 |  | 55944.92 |  |
| 150 | 6.88 | 6.88 | 3962.93 |  | 4500.41 |  |
| 200 | 6.88 |  | 13.15 |  | 23.1 |  |
| 250 | 6.88 |  | 6.91 |  | 11.5 |  |
| 350 | 6.88 |  | 4.6 |  | 6.57 |  |
| 500 | 6.88 |  | 3.39 |  | 4.48 |  |

Let us examine the predictions: what is it that we don’t manage to get?

|  |  |
| --- | --- |
| Label: the next global is: <unk>(from 29392)  INFO : Label: the next sense is: group.n.01(from 14491) | INFO : The top- 2 predicted globals are:  INFO : Word: <unk> ; p=23.19%  INFO : Word: / ; p=8.37%  INFO : The top- 2 predicted senses are:  INFO : Sense: <unk>.dummySense.01 ; p = 47.5%  INFO : Sense: group.n.01 ; p = 13.27% |
| <unk> prediction correct, both global and sense | |
| Label: the next global is: said(from 171)  INFO : Label: the next sense is: state.v.01(from 30046) | INFO : The top- 2 predicted globals are:  INFO : Word: <unk> ; p=84.22%  INFO : Word: / ; p=1.85%  INFO : The top- 2 predicted senses are:  INFO : Sense: <unk>.dummySense.01 ; p = 55.34%  INFO : Sense: state.v.01 ; p = 11.57% |
| For the globals, it’s wrong to predict <unk>, but it’s ok because we stopped when we had ~5 PPL.  For the senses, we observe that <unk>’s dummySense overtakes the actual sense, which is wrong. | |
| Label: the next global is: <unk>(from 29392)  INFO : Label: the next sense is: friday.n.01(from 13358) | INFO : The top- 2 predicted globals are:  INFO : Word: <unk> ; p=93.03%  INFO : Word: / ; p=0.79%  INFO : The top- 2 predicted senses are:  INFO : Sense: <unk>.dummySense.01 ; p = 62.92%  INFO : **Sense: friday.n.01 ; p = 11.86%** |
| Label: the next global is: an(from 147)  INFO : Label: the next sense is: an.dummySense.01(from 37670) | INFO : The top- 2 predicted globals are:  INFO : Word: <unk> ; p=23.19%  INFO : Word: / ; p=8.37%  INFO : The top- 2 predicted senses are:  **INFO : Sense: an.dummySense.01 ; p = 44.55%**  INFO : Sense: probe.n.01 ; p = 17.84% |
| ok | |
| Label: the next global is: investigation(from 3445)  INFO : Label: the next sense is: probe.n.01(from 24290) | INFO : The top- 2 predicted globals are:  INFO : Word: irregularities ; p=15.95%  INFO : Word: recent ; p=12.82%  INFO : The top- 2 predicted senses are:  **INFO : Sense: probe.n.01 ; p = 26.07%**  INFO : Sense: of.dummySense.01 ; p = 23.69% |
| ok | |
| … | |
| Label: the next global is: recent(from 3444)  INFO : Label: the next sense is: late.s.03(from 17964) | INFO : The top- 2 predicted globals are:  INFO : Word: <unk> ; p=45.86%  INFO : Word: recent ; p=8.57%  INFO : The top- 2 predicted senses are:  INFO : Sense: late.s.03 ; p = 20.87%  INFO : Sense: s.dummySense.01 ; p = 13.27% |
| ok | |

The senses appear to be mostly ok, apart from the cases where the <unk>.dummySense is predicted.

ToDo on the side: scroll through the whole set of predictions again to double-check this.

### Exp B (with freezing)

Now the condition for freezing will be:

when the new Valid-PPL of globals is > old Valid PPL + 0.1

\*\*\*\*\*\* Major issue: \*\*\*\*\*\*

Training epoch n.15:

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "/home/andrealk3/ALMA\_PhD/Code/TASK1\_Gbwe/GNN/Training.py", line 262, in training\_loop

multisense\_globals\_set, verbose)

File "/home/andrealk3/ALMA\_PhD/Code/TASK1\_Gbwe/GNN/Training.py", line 125, in **compute\_model\_loss**

update\_predictions\_history\_dict(correct\_preds\_dict, predictions\_globals, predictions\_senses, batch\_labels\_tpl)

File "/home/andrealk3/ALMA\_PhD/Code/TASK1\_Gbwe/GNN/Training.py", line 74, in **update\_predictions\_history\_dict**

**(values\_s, indices\_s) = predictions\_senses.sort(dim=1, descending=True)**

**RuntimeError: CUDA out of memory. Tried to allocate 1.06 GiB (GPU 0; 10.76 GiB total capacity; 7.30 GiB already allocated; 1.01 GiB free; 8.91 GiB reserved in total by PyTorch)**

\*\*\*\*\*\*

It may be due to allocating space for the predictions\_senses at runtime.

I will decrease the batch size from 40 to 30, keeping the sequence length at 35, and moving the learning rate to 5e-5.

Temporary test – freezing after 2 epochs …

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals: **Main GRU** with **3** layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=**30**  TBPTT length=35 |
| Senses: **GRU** with **3** layers (1024>1024>512) |
| **Freezing:** in 1st part, update only globals. Then freeze global’s network + embeddings, and update senses | grapharea=32, hops=1 |
| learning rate=0.000**05** |
| dropout=none |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Globals* | | *Senses* | | *Multi-senses* | |
|  | Train | Valid | Train | Valid | Train | Valid |
| 1 | 520.75 | 436.96 | 1.0 | 1.0 | 1.0 | 1.0 |
| 5 | 172.77 | 246.68 |  |  |  |  |
| 10 | 125.22 | 221.16 |  |  |  |  |
| 15 | 101.26 | 215.3 |  |  |  |  |
| 16 | 97.64 | 214.76 |  |  |  |  |
| 17 | 94.27 | 214.41 |  |  |  |  |
| 18 | 91.11 | 214.19 |  |  |  |  |
| 19 | 88.14 | 214.04 |  |  |  |  |
| 20 | 85.33 | 213.97 |  |  |  |  |
| 21 | 82.66 | 214.06 |  |  |  |  |
| 22 | 80.12 | 214.31 | 1.0 | 1.0 | 1.0 | 1.0 |
|  | New validation worse than previous one. Freezing the weights in the standard LM, activating senses' prediction. | | | | | |
| 23 | 91.51 | 214.35  'correct\_g': 38749, 'top\_k\_g': 76413, 'tot\_g': 151200, 'correct\_all\_s': 25768 | 1611.06 | **693.9** 'correct\_all\_s': 25768, 'top\_k\_all\_s': 60907, 'tot\_all\_s': 151200 | 12511.93  'correct\_multi\_s': 0, 'top\_k\_multi\_s': 13595, 'tot\_multi\_s': 408520 | 13000.59 'correct\_multi\_s': 0, 'top\_k\_multi\_s': 1778, 'tot\_multi\_s': 53600 |
|  | New validation worse than previous one. Freezing the weights in the standard LM, activating senses' prediction. # *Bug*: we should not go through this branch again, must adjust the code | | | | | |
| 24 | 91.51 | **214.35** | **625.79** | 754.43  'correct\_all\_s': 25796, 'top\_k\_all\_s': 61535, 'tot\_all\_s': 151200 | 10316.18  'correct\_multi\_s': 0, 'top\_k\_multi\_s': 13647, 'tot\_multi\_s': 408520 | 15400.79 'correct\_multi\_s': 0, 'top\_k\_multi\_s': 1755, 'tot\_multi\_s': 53600 |
|  | New validation worse than previous one, etc. | | | | | |
| 25 | 91.51 | 214.35 | 633.92 | 867.0 | 10685.24 | 19276.3 |
| 26 | 91.51 | 214.35 | 652.49 | 949.61 | 11227.51 | 22637.89 |

…

Memory leak causes stop at epoch 29.

The freezing mechanism, while working as intended, does not give us better results.

It is true that we have been creating a new optimizer at the end of each epoch, thus losing the gradient at the end of the epoch, but we hypothesize this does not cause a very significant shift in the results.

Therefore:

we set aside 2 GPUs to repeat the experiment, also hoping that avoiding the creation a new optimizer fixes the late memory leak.

## Comparison between the 2 base cases, with and without freezing

Input signal: only 1, the 300d FastText word embeddings

Instruments:

- GRU (1024>1024>512) for the globals

- GRU (same) for the senses. No special additions, it just involves reading the 300d word embeddings and then predicting the next token at the granularity of senses, i.e. we operate with |Vsenses|=50993 instead of |V|=30564

### Perplexity

**Standard**, both GRUs at the same time, without freezing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Globals* | | *Senses* | | *Multi-senses* | |
|  | Train | Valid | Train | Valid | Train | Valid |
| 1 | 502.74 | 524.58 | 681.98 | **647.14** | 26.38 | **10524.44** |
| 2 | 414.07 | 537.72 | 532.82 | 667.97 | 24.73 | 11324.86 |
| 3 | 353.64 | 373.13 | 530.1 | 665.78 |  | 11573.15 |
| 4 | 231.91 | 287.19 | 515.43 | **661.51** | 24.25 | 11360.57 |
| 5 | 182.83 | 259.88 | 490.2 | 663.88 |  | 11279.84 |
| 6 | 158.39 | 246.8 | 465.7 | 673.63 |  | 11442.82 |
| 7 | 142.35 | 239.53 | 443.29 | 685.59 |  | 11716.66 |
| … |  |  |  |  |  |  |
| 16 | 80.66 | **221.2** | 308.98 | 763.88 |  | 15462.58 |
| 17 | 76.86 | 222.56 | 298.0 | 779.78 |  |  |
| 18 | 73.47 | 223.61 | 288.04 | 792.36 |  |  |

The senses’ task overfits immediately. High Valid-PPL, even worse on multi-sense cases.

**With freezing**:

### Correct predictions

**Standard**:

|  |  |  |  |
| --- | --- | --- | --- |
| *Epoch* | *Globals* | *Senses* | *Multi-senses* |
|  | Validation | Validation | Validation |
| 1 | 'correct\_g': 26658,  'top\_k\_g': 62556,  'tot\_g': 151200 | 'correct\_all\_s': 25795, 'top\_k\_all\_s': 61438, 'tot\_all\_s': 151200 | **'correct\_multi\_s': 0,** 'top\_k\_multi\_s': 2236, 'tot\_multi\_s': 53600 |
| 2 | 'correct\_g': 26658,  'top\_k\_g': 62302,  'tot\_g': 151200 | 'correct\_all\_s': 25795, 'top\_k\_all\_s': 61438, 'tot\_all\_s': 151200 | 'correct\_multi\_s': 0, **'top\_k\_multi\_s': 2236,** 'tot\_multi\_s': 53600 |
| 3 | 'correct\_g': 29368,  'top\_k\_g': 65387,  'tot\_g': 151200 | 'correct\_all\_s': 25785, 'top\_k\_all\_s': 61214, 'tot\_all\_s': 151200 | 'correct\_multi\_s': 0, 'top\_k\_multi\_s': 2063, 'tot\_multi\_s': 53600 |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  | **'correct\_all\_s': 25796,** 'top\_k\_all\_s': 61650, 'tot\_all\_s': 151200 |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |
| 11 |  |  |  |
| 12 |  |  |  |
| 13 |  |  |  |
| 14 |  |  |  |
| 15 | 'correct\_g': 38312,  'top\_k\_g': 75646,  'tot\_g': 151200 | 'correct\_all\_s': 25791, **'top\_k\_all\_s': 61976,** 'tot\_all\_s': 151200 |  |
| 16 | **'correct\_g': 38403, 'top\_k\_g': 75869,**  **'tot\_g': 151200** |  |  |
| 17 | 'correct\_g': 38368,  'top\_k\_g': 75792,  'tot\_g': 151200 |  |  |
| 18 |  |  |  |
| 19 |  |  |  |
| 20 |  |  |  |

**With freezing**:

### Observations

## Input signals: word embedding, global nodestate. Senses: standard GRU

**Debugging the GraphNet after refactoring**

With parameters batchsize=2 and seq\_len=3,

batchinput\_tensor.shape=torch.Size([2, 3, 1150])

batchinput\_tensor[:,:,0]=

tensor([[80385, 80385, 51164],

[80385, 51140, 54438]])

note: they correspond to the elements:

globals: tensor([[29392, 29392, 171], ([[<unk>, <unk>, said]

[29392, 147, 3445]]) [<unk>, an, investigation]

The text starts as:

<word surface\_form="**The**" pos="DT"/>

<word surface\_form="**Fulton\_County\_Grand\_Jury**" lemma="group" pos="NN" wn16\_key="group%1:03:00::" wn30\_key="group%1:03:00::"/>

<word surface\_form="**said**" lemma="say" pos="VBD" wn16\_key="say%2:32:00::" wn30\_key="say%2:32:00::"/>

<word surface\_form="**Friday**" lemma="friday" pos="NNP" wn16\_key="friday%1:28:00::" wn30\_key="friday%1:28:00::"/>

<word surface\_form="**an**" pos="DT"/>

<word surface\_form="**investigation**" lemma="investigation" pos="NN" wn16\_key="investigation%1:09:00::" wn30\_key="investigation%1:09:00::"/>

### POTENTIAL MAJOR ISSUE

I may be reading the tokens as-they-are when operating with a lowercased vocabulary, which means that \*all\* the tokens with an upper-cased character are read as <unk>. This means that the experiments made until now in “LOWERCASED SEMCOR + MASC, PART I” are more inaccurate than they should be.

# LOWERCASED SEMCOR + MASC, attempt II

## Preliminary

### Graph stats

INFO : Constructing X, matrix of node features

INFO : X\_senses.shape=torch.Size([**50993**, 300])

INFO : X\_globals.shape=torch.Size([**30564**, 300])

INFO : X\_definitions.shape=torch.Size([35221, 300])

INFO : X\_examples.shape=torch.Size([30378, 300])

senses=[0,50993) # with data: [0,35220)

globals=[50993, 81557)

definitions=[81557, 116778)

examples=[116778, 147156)

INFO : def\_edges\_se.\_\_len\_\_()=35221

INFO : exs\_edges\_se.\_\_len\_\_()=30378

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/semcormasc.xml

INFO : sc\_edges\_with\_external.\_\_len\_\_()=9946 (with the lowercase error, they were 8170)

INFO : sc\_edges.\_\_len\_\_()=60939

INFO : get\_edges\_selfloops>max\_sense=50992

INFO : []

INFO : len(edges\_ls)==0

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=60939

INFO : syn\_edges.\_\_len\_\_()=30288

INFO : ant\_edges.\_\_len\_\_()=4764

kbgraph.dataobject, file size~= 181 MB

### Graph retrieval

AD.get\_node\_data(grapharea\_matrix, **60000**, grapharea\_size=32, features\_mask=(True,False,False))

Node n: 0

(tensor([60000, 24459, 24460, 24461, 20532, 24462, 24458, 24463, 59982, 54582,

51349, 52373, 66417]), None, None)

Adjacent nodes:

|  |  |  |
| --- | --- | --- |
| [20532, | sense | motivate.v.01 |
| 24458, | sense | prompt.n.01 |
| 24459, | sense | prompt.n.02 |
| 24460, | sense | prompt.s.01 |
| 24461, | sense | prompt.s.02 |
| 24462, | sense | prompt.v.02 |
| 24463, | sense | prompt.v.03 |
| 51349, | global | suggestion |
| 52373, | global | immediate |
| 54582, | global | remind |
| 59982, | global | inspire |
| 60000, | global | **prompt** |
| 66417] | global | motivate (here a synonym) |

### Node & input loading

bsz=2, seq\_len=5

batchinput\_tensor[:,:,0]=

tensor([[50998, 80385, 51164, 51811, 51140],

[54438, 51004, 59023, 55677, 54437]])

Corresponding to the globals:

[5, 29392, 171, 818, 147], = [the <unk> said Friday an]

[3445, 11, 8030, 4684, 3444] = [investigation of atlanta s recent]

In run\_graphnet():

t\_globals\_indices\_ls[i\_sample] = tensor([50998, 45750])

where 45750 is the.dummySense.01

After the lemmatization, x\_indices is unchanged at tensor([50998, 45750]) – since ‘the’ is not modified.

## Input signal: word embedding. Senses: standard GRU

### Model A

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)

)

(senses\_rnn\_ls): ModuleList(

(0): GRU(300, 1024)

(1): GRU(1024, 1024)

(2): GRU(1024, 512)

)

(linear2global): Linear(in\_features=512, out\_features=30564, bias=True)

(linear2senses): Linear(in\_features=512, out\_features=50993, bias=True))

)

INFO : Parameters:

INFO : ('module.X', torch.Size([147156, 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.senses\_rnn\_ls.0.weight\_ih\_l0', torch.Size([3072, 300]), torch.float32, True)

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

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

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

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

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

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

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

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

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

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

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

('module.linear2global.weight', torch.Size([30564, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([30564]), torch.float32, True)

('module.linear2senses.weight', torch.Size([50993, 512]), torch.float32, True)

('module.linear2senses.bias', torch.Size([50993]), torch.float32, True)

INFO : Number of trainable parameters=111.45M,

where embeddings=44.15M , softmax(globals)= 15.68M, softmax(senses)= 26.16M, core=25.46M (~ 12.73M x2)

### Mini-exp A

On fragment\_semcormasc.xml

|  |  |  |  |
| --- | --- | --- | --- |
| *Epoch* | *Training PPL* | | |
|  | *Globals* | *Senses* | *Multi-senses* |
| 1 | 29319.07 | 48896.32 | 50981.49 |
| 10 | 27.64 | 29.46 | 31.96 |
| 100 | 26.64 | 28.26 | 33.55 |
| 150 | 20.82 | 22.5 | 27.81 |
| 200 | 14.19 | 15.12 | 18.53 |
| 250 | 8.45 | 9.08 | 11.62 |
| 300 | 5.25 | 5.7 | 7.05 |
| 350 | 3.63 | 3.87 | 4.64 |
| 400 | 2.55 | 2.67 | 3.1 |
| 450 | 1.93 | 1.99 | 2.22 |

### Exp A

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Globals* | | *Senses* | | *Multi-senses* | |
|  | Train | Valid | Train | Valid | Train | Valid |

## Input signal: word embedding. Senses: Freezing, standard GRU

We set up the freezing mechanism that was established before.

In the mini-experiment, we freeze the globals’ network and activate the senses’ network after the Globals PPL is < 3.

In the experiment, we freeze after the Valid PPL is > 0.1 + the best one so far.

### Mini-exp B

|  |  |  |  |
| --- | --- | --- | --- |
| *Epoch* | *Training PPL* | | |
|  | *Globals* | *Senses* | *Multi-senses* |
| 1 | 30040.03 | 1.0 | 1.0 |
| 10 | 28.03 |  |  |
| 100 | 26.08 |  |  |
| 200 | 17.86 |  |  |
| 250 | 11.54 |  |  |
| 300 | 7.03 |  |  |
| 350 | 4.65 |  |  |
| 400 | 3.3 |  |  |
| 434 | 2.95 | 1.0 | 1.0 |
| 435 | 2.95 | 62183.76 | 73451.25 |
| 436 | “ | 5189.81 | 6581.97 |
| 500 | “ | 7.68 | 9.52 |
| 600 | “ | 3.33 | 3.86 |
| 700 | “ | 2.33 | 2.58 |

**Exp B**

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals: **Main GRU** with **3** layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=**32**  TBPTT length=35 |
| Senses: **GRU** with **3** layers (1024>1024>512) |
| **Freezing:** in 1st part, update only globals. Then freeze global’s network + embeddings, and update senses | grapharea=32, hops=1 |
| learning rate=0.0001 |
| dropout=none |

\*\*\*\*\*\* **Issue** \*\*\*\*\*\*

It would seem that I have broken SemCor and SemCor+MASC with the latest modifications.

The experiments get

Train PPL 188.52 , Valid PPL 494.57 (with freezing) on SemCor

Train PPL 783.04, Valid PPL 1244.3 (standard, without freezing) on SemCor+MASC

What were the last modifications?

In GNN/NumericalIndices.py:

|  |
| --- |
| word = VocabUtils.process\_word\_token(token\_dict) # html.unescape |
| word = VocabUtils.process\_word\_token(token\_dict, lowercasing=True) # html.unescape + currently lowercasing |

best\_valid\_loss instead of previous\_valid\_loss in Training.py

In VocabularyUtilities.py:

|  |
| --- |
| def process\_word\_token(token\_dict, lowercasing): |
| # ------- the only essential step |
| token\_text = html.unescape(str(token\_dict['surface\_form'])) |
| # ------- superfluous on not-WT2 |
| # token\_text = convert\_symbols(token\_text) |
|  |
| # ------- we are already lowercasing on SemCor now |
| # ------- actually, in the current version we need to lowercase, because the vocabulary is lowercased |
| if lowercasing: |
| token\_text = token\_text.lower() |
| # if token\_text == token\_text.upper(): # if ALL CAPITALS -> must lowercase |
| # token\_text = token\_text.lower() # we are not lowercasing anymore, otherwise 'USA'->'usa' |

In Graph/DefineGraph.py:

In get\_additional\_edges\_sensechildren\_from\_slc():

|  |
| --- |
| word = VocabUtils.process\_word\_token(token\_dict) # html.unescape |
| word = VocabUtils.process\_word\_token(token\_dict, lowercasing=True) |
|  |

In get\_edges\_nyms():

|  |
| --- |
| word1 = VocabUtils.process\_word\_token({'surface\_form': word1}) |
| word1 = VocabUtils.process\_word\_token({'surface\_form': word1}, lowercasing=True) |  |  |

I decide to use ½ of the training and the validation set of SemCor, to have a faster data set for debugging operations.

First, I try to re-execute the whole pipeline, keeping all the lowercased modifications.

On: the entirety of SemCor. It makes for a faster development set. Then we can extend some of the experiments to SemCor + MASC

# Lowercased SemCor, attempt 2

### Corpus & graph

INFO : Constructing X, matrix of node features

INFO : X\_senses.shape=torch.Size([**41206**, 300]) # senses with data: 30445

INFO : X\_globals.shape=torch.Size([**22782**, 300])

INFO : X\_definitions.shape=torch.Size([30446, 300])

INFO : X\_examples.shape=torch.Size([27921, 300])

Graph ranges: [0, senses, 41206, globals, 63988, defs, 94434, examples, 122355]

Quick check on graph retrieval:

From sense 35, ablaze.s.03

[35, 64023-> definition 35=resembling flame in brilliance or color,   
60469 -> global 19263=ablaze ,

94475-> example 41=maple trees ablaze in autumn]

Quick check on input nodes:

## 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)

)

(senses\_rnn\_ls): ModuleList(

(0): GRU(300, 1024)

(1): GRU(1024, 1024)

(2): GRU(1024, 512)

)

(linear2global): Linear(in\_features=512, out\_features=22782, bias=True)

(linear2senses): Linear(in\_features=512, out\_features=41206, bias=True)

))

INFO : Parameters:

INFO : ('module.X', torch.Size([122355, 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.senses\_rnn\_ls.0.weight\_ih\_l0', torch.Size([3072, 300]), torch.float32, True)

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

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

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

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

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

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

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

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

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

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

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

('module.linear2global.weight', torch.Size([22782, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([22782]), torch.float32, True)

('module.linear2senses.weight', torch.Size([41206, 512]), torch.float32, True)

('module.linear2senses.bias', torch.Size([41206]), torch.float32, True)

INFO : Number of trainable parameters=95.00M, where

embeddings=36.71M, softmax(globals)=11.69M, softmax(senses)= 21.14M , core=25.47M (~12.73M x2)

### Trying to debug through mini-experiments

From NumericalIndices.py, we get the:

{ current\_token\_tpl= (0, tensor(34558)), -> (the, the.dummySense.01)

next\_token\_tpl= (22213, 12610) -> (<unk>, group.n.01) }

{ current\_token\_tpl=(22213, 12610),

next\_token\_tpl= (1, 26023) -> (said, state.v.01) } …

tpl=(2,11654)=(friday, friday.n.01)

In the very first forward() call of the RNN,

t\_current\_globals\_indices\_ls = <class 'list'>: [tensor(41206), tensor(41210)] -> (the, investigation)

Currently, with lowercasing, in the forward() call the t\_globals\_indices\_ls contains:

INFO : t\_current\_globals\_indices\_ls=[tensor(41206), tensor(41210)] 'the', 'investigation'

INFO : t\_current\_globals\_indices\_ls=[tensor(63419), tensor(41211)] ‘<unk>', 'of'

INFO : t\_current\_globals\_indices\_ls=[tensor(41207), tensor(41212)] 'said', 'atlanta'

INFO : t\_current\_globals\_indices\_ls=[tensor(41208), tensor(46384)] 'friday', 's'

INFO : t\_current\_globals\_indices\_ls=[tensor(41209), tensor(41214)] ‘an’, ‘recent’

Currently,

batch\_labels\_globals=tensor([22213, 1, 2, 3, 4, 5, 6, 5178, 8, 9])

and

batch\_labels\_all\_senses=tensor([12610, 26023, 11654, 35103, 21009, 36523, 1623, 36603, 15571, 20931])

### Experiment on SemCor

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

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals: **Main GRU** with 3 layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| Senses: **GRU**, as above | grapharea=32, hops=1 |
| **Freezing:** in 1st part, update only globals. Then freeze global’s network + embeddings, and update senses | learning rate=10^(-4) |
| dropout=none |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Globals* | | *Senses* | | *Multi-senses* | |
|  | Train | Valid | Train | Valid | Train | Valid |
| 1 | 924.12 | 834.26 | 1.0 | 1.0 | 1.0 | 1.0 |
| 2 | 654.34 | 694.56 |  |  |  |  |
| 3 | 487.08 | 541.36 |  |  |  |  |
| 4 | 377.2 | 452.41 |  |  |  |  |
| 5 | 307.69 | 408.11 |  |  |  |  |
| 6 | 267.13 | 386.4 |  |  |  |  |
| 7 | 239.57 | 373.72 |  |  |  |  |
| 8 | 218.27 | 366.05 |  |  |  |  |
| 9 | 200.79 | 361.31 |  |  |  |  |
| 10 | 185.96 | 358.38 |  |  |  |  |

Very slow, and quite bad.

Expected Valid-PPL: around 340.

## Check: Experiments on WikiText-2

I feel the need to verify the functionality of the current architecture.

I will try it on the standard LM task on WikiText-2.

We wish to replicate results close to the 186 Valid PPL of [Experiment B](#_Experiment_B)

And we also rebuild the pipeline from scratch, just to be sure that everything works as intended (or doesn’t).

(side idea: I could fine-tune a model coming from WikiText-2 into SemCor… and then apply the senses’ task.

Otherwise, SemCor + MASC is still a valid idea.)

### Standard LM, no lowercasing

We set the lowercasing and lower flags in the code to False.

(The previous result, obtained with a GRU with the same parameters at [Experiment B](#_Experiment_B), is Valid.PPL=186.4 @ epoch 28)

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

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL* | *Valid PPL* |
| 1 | 1435.94 | 707.43 |
| 2 | 684.78 | 444.6 |
| 3 | 483.73 | 361.77 |
| 4 | 395.68 | 318.97 |
| 5 | 342.02 | 290.98 |
| 6 | 304.07 | 270.97 |
| 7 | 275.02 | 255.8 |
| 8 | 251.53 | 243.59 |
| 9 | 231.93 | 233.66 |
| 10 | 215.3 | 225.65 |
| 11 | 200.98 | 219.12 |
| 12 | 188.48 | 213.68 |
| 13 | 177.43 | 209.16 |
| 14 | 167.57 | 205.39 |
| 15 | 158.7 | 202.28 |
| 16 | 150.65 | 199.6 |
| 17 | 143.31 | 197.28 |
| 18 | 136.56 (original 134.9/193.8, presumably due to random initialization) | 195.32 |
| 19 | 130.34 | 193.66 |
| 20 | 124.57 | 192.21 |
| 21 | 119.2 | 190.99 |
| 22 | 114.17 | 189.95 |
| 23 | 109.46 | 189.2 |
| 24 | 105.02 | 188.6 |
| 25 | 100.85 | 188.29 |
| 26 | 96.89 | 188.04 |
| 27 | 93.15 | **188.02** |
| 28 | 89.6 | 188.04 |
| 29 | 86.23 | 188.35 |
| 30 | 83.03 | 188.76 |

We can now confirm that the current Pipeline and RNN works as intended.

We also store away the model we saved when the validation started increasing. It may be useful, e.g. to use it as a pre-trained base and fine-tune it later on SemCor.

There are, therefore, 2 hypotheses:

1. Training over SemCor alone (650k tokens) or even SemCor + MASC (~1million tokens) is not as effective as training on WikiText-2, and thus we get worse perplexity.
2. There is a bug in how we set up / read / operate on sense-labeled corpuses. I mostly exclude this hypothesis, because:
   1. the graph retrieval works: if I select a global, the adjacent nodes are its senses and synonyms, etc.
   2. the input (that we use to retrieve the word embeddings) and the labels are the ones we expect

# Further developments: loading saved model, updating senses methods

## Loading a saved model

Currently,

**if** exp(valid\_loss\_globals) > exp(best\_valid\_loss\_globals) + 0.1: *# if \_new\_ Valid PPL worse than \_best\_ by >0.1*

**if not** with\_freezing: 🡺 early stop

**else:** 🡺 freeze globals, activate senses:   
 … p.requires\_grad=**False,** model\_forParameters.predict\_senses = **True**

Then, either when we early-stop, or when we use a Keyboard Interrupt,

torch.save(model, os.path.join(F.FOLDER\_GNN, hyperparams\_str +  
 **'step\_'** + str(overall\_step) + **'.~~rgcn~~model'**))

For instance, we can manually rename it to:

‘pretrained\_model.pt’.

And also make ‘pretrained\_model\_copy.pt’

When we execute the training\_setup(), we should have a load\_saved\_model parameter. If it is set, then we do not create a model from scratch with the usual settings (3 layers, 1024, etc.) but we load GNN/pretrained\_model.pt

n: also renaming GNN to NN

### Saving & loading, mini-experiment on SemCor

When we load a model, it would be opportune to first try it on the validation set with no modification, and see how it performs.

Epoch n.3:

Training: Globals PPL=**57.67**; PPL on all senses=112.52; PPL on multi-senses=667.7

Validation: Globals PPL=183548.64 PPL on all senses=145310.3   
Perplexity on multi-senses=122174.41

Let us try to rename the saved model into pretrained\_model.pt and load it…

INFO : After training 0 epochs, validation:

Globals PPL=178711.71 PPL on all senses=146706.49 PPL on multi-senses=123387.89

-------

INFO :

Epoch n.1:

Training: Globals PPL=**34.6** PPL on all senses=53.51 PPL on multi-senses=218.14

Yes, it appears that we load the model stored in pretrained\_model.pt. Training can resume.

The only problem is that if the model was not created with predict\_senses=True, it will not have the senses’ GRU and thus we will be unable to use it to predict senses.

It is therefore necessary to change where we specify the parameter predict\_senses.

The model is built with predict\_senses=True in any setup\_train(). Then, in run\_train() we specify if we are actually using the predict-senses part or not.

### Mini-testing: Saving, loading, predicting senses, freezing

Let us try to:

**1.** Build a model, without predicting senses, and save it. If we are operating with freezing, we should have 2 save points, one at early-stopping on globals (the one we use, in 2 copies), and a final one.

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch | Training PPL | | |
|  | Globals | Senses | Multi-senses |
| 1 | 14621.58 | 1.0 | 1.0 |
| 2 | 268.44 | 1.0 | 1.0 |
| 3 | 49.37 | 1.0 |  |
|  | New validation worse than previous one. Freezing the weights in the standard LM, activating senses' prediction. | | |
| 4 | 31.31 | 29166.77 | 38295.26 |
| 5 | 35.81 | 2109.95 | 3600.8 |
| 6 | 36.01 | 147.82 | 277.46 |
| 7 | 36.11 | 54.12 | 98.12 |
|  | Early stopping on senses. | | |

The freezing mechanism is acceptable.

We now train only on globals, with no predict\_senses.

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch | Training PPL | | |
|  | Globals | Senses | Multi-senses |
| 1 | 11963.73 | 1.0 | 1.0 |
| 2 | 199.84 | 1.0 | 1.0 |
| 3 | 44.73 | 1.0 | 1.0 |

We use this as starting point.

**2.** Load the pretrained model, and start training&validation on both globals and senses (no freezing).

For the globals, check that we resume training from the same spot.

AttributeError: 'DataParallel' object has no attribute 'grapharea\_size' … 2 nested DataParallel objects are saved, solved by loading model.module

It seems I am not predicting senses… I should make the assignment on model\_forParameters…

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch | Training PPL | | |
|  | Globals | Senses | Multi-senses |
| 1 | 86.02 | 32551.27 | 38074.3 |
| 2 | 43.45 | 558.86 | 767.94 |
| 3 | 39.29 | 71.3 | 109.75 |

Loading the model trained only on globals, and proceeding to train on globals and senses both; ok.

**3.** Load the latest pretrained model, and start training&validation on both globals and senses – this time, with freezing.

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch | Training PPL | | |
|  | Globals | Senses | Multi-senses |
| 1 | 67.0 | 1.0 | 1.0 |
| 2 | 40.04 | 1.0 | 1.0 |
| 3 | 36.6 | 1.0 | 1.0 |
|  | New validation worse than previous one. Freezing the weights in the standard LM, activating senses' prediction. | | |
| 4 | 37.9 | 99.77 | 182.44 |
| 5 | 33.88 | 75.14 | 130.12 |
| 6 | 37.9 | 43.95 | 79.25 |
| 7 | 37.9 | 39.87 | 67.82 |
|  | Early stopping on senses. | | |

We confirmed that we can load models saved previously and train them correctly, with or without freezing.

# Final Experiments

## WikiText-2 (standard LM)

Standard Language Model task, i.e. globals only.

Objective: train a model on WikiText-2, that will be stored and used as a pretrained base to operate on SemCor.

### Experiment

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals: **Main GRU** with 3 layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| Senses:nothing. We create a GRU, but we do not use it. | grapharea=32, hops=1  learning rate=0.00005 |
| *Later on we may also add other mechanisms and parameters to the model object.* |  |

Expecting a Valid-PPL in the range 185-190

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Validation PPL, globals* |
| 1 | 1424.14 | 699.05 |
| 2 | 673.08 | 439.06 |
| 3 | 476.61 | 357.24 |
| 4 | 390.06 | 314.98 |
| 5 | 337.25 | 287.66 |
| 6 | 299.44 | 267.93 |
| 7 | 270.14 | 252.68 |
| 8 | 246.53 | 240.78 |
| 9 | 227.07 | 231.31 |
| 10 | 210.71 | 223.59 |
| 11 | 196.69 | 217.14 |
| 12 | 184.48 | 211.69 |
| 13 | 173.72 | 207.19 |
| 14 | 164.13 | 203.43 |
| 15 | 155.5 | 200.33 |
| 16 | 147.66 | 197.71 |
| 17 | 140.49 | 195.51 |
| 18 | 133.89 | 193.63 |
| 19 | 127.79 | 192.01 |
| 20 | 122.13 | 190.65 |
| 21 | 116.86 | 189.53 |
| 22 | 111.93 | 188.62 |
| 23 | 107.31 | 187.87 |
| 24 | 102.96 | 187.41 |
| 25 | 98.86 | 187.06 |
| 26 | 94.99 | 186.96 |
| 27 | 91.32 | **186.94** |
| 28 | 87.84 | 187.17 |

We saved the model @epoch 28. We store it, to be used later as a pretrained model.

## Simple GRU for senses

### Directly on SemCor

In these experiments, we aim to examine the basic - expected to be mediocre - performance of having 1 GRU for globals and 1 GRU for senses, operating on SemCor alone, without using any pre-trained model.

There are 2 experiments because 1 of them will use the freezing mechanism: first train on globals, then freeze globals’ GRU + embeddings and train the senses’ part (that here is a simple GRU)

#### Without freezing:

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals: **Main GRU** with 3 layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=**32**  TBPTT length=35 |
| Senses: **GRU** with 3 layers, as above. |  | grapharea=32, hops=1  learning rate=0.00005 |

INFO : Hyperparameters: \_batchPerSeqlen1120\_area32\_lr5e-05\_epochs50

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Epoch | Training | | | Validation | | |
|  | Globals | Senses | Multi-senses | Globals | Senses | Multi-senses |
| 1 | 624.7 | 984.94 | 11219.87 | 509.6 | 636.94 | 10688.14 |
| 2 | 421.37 | 589.57 | 8046.9 | 516.33  'correct\_g': 12157, 'top\_k\_g': 33203, 'tot\_g': 80640 | **622.13** **'correct\_all\_s': 11761, 'top\_k\_all\_s': 32807, 'tot\_all\_s': 80640** | 9850.86  'correct\_multi\_s': 0,  'top\_k\_multi\_s': 1139, 'tot\_multi\_s': 29741 |
| 3 | 387.33 | 584.73 | 7865.8 | 434.09  'correct\_g': 12149, 'top\_k\_g': 33463, 'tot\_g': 80640 | 624.57  'correct\_all\_s': 11761, 'top\_k\_all\_s': 32850, 'tot\_all\_s': 80640 | **9794.35**  **'correct\_multi\_s': 0, 'top\_k\_multi\_s': 1218, 'tot\_multi\_s': 29741** |
| 4 | 331.03 | 582.65 | 7818.38 | 377.57 | 633.32 | 10159.82 |
| 5 | 275.48 | 580.82 | 7783.37 | 332.07 | 640.72 | 10419.21 |
| 6 | 239.38 | 574.53 | 7663.76 | 301.53 | 648.67 | 10576.26 |
| 7 | 213.68 | 566.62 | 7500.19 | 281.85 | 655.55 | 10694.57 |
| 8 | 195.61 | 556.64 | 7291.15 | 269.3 | 661.72 | 10856.34 |
| 9 | 182.22 | 545.42 | 7047.44 | 260.87 | 667.51 | 10995.44 |
| 10 | 171.65 | 532.9 | 6772.09 | 254.96 | 672.27 | 11062.62 |
| 11 | 162.92 | … | … | 250.74 | … | … |
| 12 | 155.45 |  |  | 247.77 |  |  |
| 13 | 148.89 |  |  | 245.65 |  |  |
| 14 | 143.07 |  |  | 243.82 |  |  |
| 15 | 137.91 |  |  | 242.02 |  |  |
| 16 | 133.08 |  |  | 240.75 | 'correct\_all\_s': 11761, 'top\_k\_all\_s': **33004**, 'tot\_all\_s': 80640 |  |
| 17 | 128.49 |  |  | 240.09 |  |  |
| 18 | 124.27 |  |  | 239.91 |  |  |
| 19 | 120.34 |  |  | 239.63 |  |  |
| 20 | 116.69 |  |  | 238.34 |  |  |
| 21 | 113.16 | 413.6 | 4364.63 | **237.85**  **'correct\_g': 19585, 'top\_k\_g': 38771, 'tot\_g': 80640** | 747.85  'correct\_all\_s': 11761, 'top\_k\_all\_s': 32962, 'tot\_all\_s': 80640 | 13655.03  'correct\_multi\_s': 0,  'top\_k\_multi\_s': 1445, 'tot\_multi\_s': 29741 |
| 22 | 109.89 |  |  | 237.94 |  |  |
| 23 | 106.8 |  |  | 238.45 | 776.66  'correct\_all\_s': **11763**, 'top\_k\_all\_s': 32921, 'tot\_all\_s': 80640 |  |

Observations:

* The Language Modeling task at the granularity of WordNet senses is more difficult than standard LM. When choosing among the 41206 senses, that include a dummySense for the words without a WordNet sense (e.g. stopwords like ‘the’, ‘and’, ‘of’, etc.) we obtain at best **622.13** Validation-PPL.   
  The standard word prediction gives us **237.85** on a vocabulary of 22782 tokens.

#### With freezing:

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals: **Main GRU** with 3 layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=**32**  TBPTT length=35 |
| Senses: **GRU** with 3 layers, as above. |  | grapharea=32, hops=1  learning rate=0.00005 |
| **Freezing:** in 1st part, update only globals. Then freeze global’s network + embeddings, and update senses |  |  |

INFO : Hyperparameters: \_batchPerSeqlen1120\_area32\_lr5e-05\_epochs50

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Epoch | Training | | | Validation | | |
|  | Globals | Senses | Multi-senses | Globals | Senses | Multi-senses |
| 1 | 624.12 | 1.0 | 1.0 | 506.65 | 1.0 | 1.0 |
| 2 | 353.24 |  |  | 392.99 |  |  |
| 3 | 285.21 |  |  | 325.3 |  |  |
| 4 | 239.58 |  |  | 295.35 |  |  |
| 5 | 213.44 |  |  | 275.35 |  |  |
| 6 | 194.04 |  |  | 260.43 |  |  |
| 7 | 179.33 |  |  | 250.25 |  |  |
| 8 | 168.0 |  |  | 243.06 |  |  |
| 9 | 158.87 |  |  | 237.73 |  |  |
| 10 | 151.15 |  |  | 233.66 |  |  |
| 11 | 144.47 |  |  | 230.68 |  |  |
| 12 | 138.57 |  |  | 228.5 |  |  |
| 13 | 133.27 |  |  | 226.94 |  |  |
| 14 | 128.45 |  |  | 225.84 |  |  |
| 15 | 124.01 |  |  | 225.12 |  |  |
| 16 | 119.9 |  |  | 224.7 |  |  |
| 17 | 116.07 |  |  | **224.53** |  |  |
| 18 | 112.47 |  |  | 224.55 |  |  |
| 19 | 109.09 |  |  | 224.72 |  |  |
|  | INFO : New validation worse than previous one. Freezing the weights in the standard LM, activating senses' prediction. | | | | | |
| 20 | 110.88 | 922.08 | 11809.34 | 227.43 | **734.85** | 13915.21 |
| 21 | 110.91 | 667.26 | 10058.32 | 227.43 | 878.89 | 19483.47 |
| 22 | 110.91 | 597.93 | 8125.0 | 227.43 | 832.14 | 17024.97 |
| 23 | 110.91 | 561.7 | 7298.96 | 227.43 | **806.4** | 15764.11 |
|  | early stopping on senses. | | | | | |

Observations:

* In the first part, the model is entirely dedicated to the standard word-prediction task, and we reach **224.53** Valid-PPL (instead of the 237.85 of joint optimization).
* We expect the senses’ task to be more difficult, because while in joint optimization the embeddings are moved based on both the globals’ loss and the senses’ loss, now they are only moved by the globals’ task in the 1st part. To predict the senses, we modify only the senses’ GRU.
* The freezing mechanism can be more useful when we use a specific prediction mechanism for the senses that depends on the performance of the globals’ task – for instance, when we select the senses of the first *k* most likely globals and then choose the next sense among them.
* Nevertheless, for the sake of completeness, I can dedicate 1 GPU to repeating this experiment, changing the condition for early stop on senses to “0.1 + *previous* validation-PPL” instead of *best*

#### With freezing:

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals: **Main GRU** with 3 layers (1024>1024>512) | 1) The word embedding of the current global (d=300) | batch\_size=**32**  TBPTT length=35 |
| Senses: **GRU** with 3 layers, as above. |  | grapharea=32, hops=1  learning rate=0.00005 |
| **Freezing:** in 1st part, update only globals. Then freeze global’s network + embeddings, and update senses |  |  |

INFO : Hyperparameters: \_batchPerSeqlen1120\_area32\_lr5e-05\_epochs50

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Epoch | Training | | | Validation | | |
|  | Globals | Senses | Multi-senses | Globals | Senses | Multi-senses |
| 1 | 622.19 | 1.0 | 1.0 | 510.02 | 1.0 | 1.0 |
| 2 | 388.92 |  |  | 415.91 |  |  |
| 3 | 304.69 |  |  | 340.73 |  |  |
| 4 | 251.32 |  |  | 304.58 |  |  |
| 5 | 219.76 |  |  | 279.36 |  |  |
| 6 | 197.37 |  |  | 262.58 |  |  |
| 7 | 181.3 |  |  | 251.41 |  |  |
| 8 | 169.26 |  |  | 243.85 |  |  |
| 9 | 159.71 |  |  | 238.4 |  |  |
| 10 | 151.77 |  |  | 234.33 |  |  |
| 11 | 144.93 |  |  | 231.29 |  |  |
| 12 | 138.88 |  |  | 229.08 |  |  |
| 13 | 133.43 |  |  | 227.46 |  |  |
| 14 | 128.46 |  |  | 226.3 |  |  |
| 15 | 123.87 |  |  | 225.51 |  |  |
| 16 | 119.61 |  |  | 225.03 |  |  |
| 17 | 115.61 |  |  | 224.76 |  |  |
| 18 | 111.86 |  |  | **224.68** |  |  |
| 19 | 108.31 |  |  | 224.76 |  |  |
| 20 | 104.95 | 1.0 | 1.0 | 224.99 | 1.0 | 1.0 |
|  | New validation worse than previous one. Freezing the weights in the standard LM, activating senses' prediction. | | | | | |
| 21 | 107.51 | 2876.43 | 13139.11 | 227.57 | 831.64 | 11441.93 |
| 22 | 107.55 | 680.43 | 9600.36 | 227.57 | 807.23 | 16079.94 |
| 23 | “ | 596.68 | 7981.31 | “ | 770.89 | 14526.66 |
| 24 |  | 562.87 | 7266.92 |  | 755.05 | 13816.96 |
| 25 |  | 539.36 | 6782.77 |  | 748.74 | 13489.71 |
| 26 |  | 521.16 | 6415.38 |  | **747.72** | 13372.17 |
| 27 |  | 506.12 | 6116.87 |  | 749.9 | 13381.47 |
|  | INFO : Early stopping on senses. | | | | | |

In the latest experiment of (Simple GRUs, with freezing), we reach:

Globals Valid-PPL=**224.68** , Senses Valid-PPL=**747.72**

Compared with the **237.85/622.13** of joint optimization.

Nevertheless, with freezing or not, it is evident that the senses’ task (language model prediction at the granularity of senses) is too difficult to be handled appropriately by a simple GRU. We need to devise other mechanisms.

What could be the next steps?

* Adding the GAT nodestate of the global should provide additional information, and ideally improve both the standard LM and the senses’ prediction.  
  From this, 2 experiments would follow: Simple GRUs + GAT input, with and without freezing.
* Using the pretrained WT-2 model? This should improve the performance of the standard LM task on globals…  
  However, it is likely that using the pretrained WT-2 is more significant on methods that actually rely on the globals’ performance (SelectK).
* Then, moving on to the other mechanisms

### On SemCor, input signal with GAT – version 1

We use a Graph Attention Network to obtain the state of the *global* node of word *w*.

The input signal to the GRUs, thus, becomes:

word embedding of *w* ++ node-state of *w*’s global node.

Graph Attention Network, 2 heads, 150x2=d300

Note:

However, are we sure that we are not just replicating the word embedding and modifying it?

If the word embedding of *wi* is the *i*-th row of the embeddings’ matrix, and the embeddings’matrix is just the matrix X of the dictionary graph, are we sure we should not duplicate X, to have the 2 versions (1) word embeddings and (2) global node-states, modified by the GAT through the graph?

**Preliminary experiment: On SemCor, joint optimization (no freezing), with the old version of the GAT input**

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals: **Main GRU** with 3 layers (1024>1024>512) | 1) The word embedding of the current global *wi* (d=300)  2) The state vector of the global node *I*, obtained applying a GNN (GAT) on the dictionary graph.  For now, using the same matrix X for (1) and (2) | batch\_size=**32**  TBPTT length=35 |
| Senses: **GRU** with 3 layers, as above. | grapharea=32, hops=1  learning rate=0.00005 |

INFO : Model:

INFO : DataParallel(

(module): RNN(

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

(gat\_globals): GATConv(300, 150, heads=2)

(main\_rnn\_ls): ModuleList(

(0): GRU(600, 1024)

(1): GRU(1024, 1024)

(2): GRU(1024, 512))

(senses\_rnn\_ls): ModuleList(

(0): GRU(600, 1024)

(1): GRU(1024, 1024)

(2): GRU(1024, 512))

(linear2global): Linear(in\_features=512, out\_features=22782, bias=True)

(linear2senses): Linear(in\_features=512, out\_features=41206, bias=True)))

INFO : Parameters:

INFO : ('module.X', torch.Size([122355, 300]), torch.float32, True)

…

('module.linear2global.weight', torch.Size([22782, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([22782]), torch.float32, True)

('module.linear2senses.weight', torch.Size([41206, 512]), torch.float32, True)

('module.linear2senses.bias', torch.Size([41206]), torch.float32, True)

INFO : Number of trainable parameters=96.933M,

where embeddings=36.706M , softmax=11.687+21.139=32.826M, core=27.40M (~13.7x2)

INFO : Hyperparameters: \_batchPerSeqlen1120\_area32\_lr5e-05\_epochs50

\*\*\*\*\*\*\* Issue \*\*\*\*\*\*\*

Globals perplexity=nan Perplexity on all senses=nan Perplexity on multi-senses=nan

There is a bug.

\*\*\*\*\*\*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Epoch | Training | | | Validation | | |
|  | Globals | Senses | Multi-senses | Globals | Senses | Multi-senses |
| 1 |  |  |  |  |  | 1.0 |

Next: if we duplicate the matrix X, can we obtain better results? Do we exceeed the parameter budget?

## Revising and debugging the GNN input signal

We should not use the same vector as both word embedding, modified through the line of the word embedding input signal, and as node-state of the global node. We end up concatenating the same vector in the input.

Immediate problem: if we just duplicate the matrix X and keep the rest of the architecture unchanged, the number of parameters is going to increase significantly (last time we checked, the embeddings matrix on SemCor was 122355x300=36.706M).

However, we can:

1) Decrease the size of the architecture. Instead of having 1024>1024>512, we can have 800>800>400 (or 900>900>450), to “make space”. Can our high-quality, separate signal coming from the graph improve the performance, maintaining the same parameter budget?

2) Use 2 separate matrices: the matrix of word embeddings, and the graph matrix X.   
Instead of having 122355x(d=300), we could use d=100, or d=80.  
3) Obtain the graph matrix X by applying PCA on the matrix of word embeddings. (Or we can just initialize it randomly).

### Parameter budget

We examine the amount of parameters of a SimpleGRU model on SemCor.

From the previous SimpleGRU experiments,

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))

(senses\_rnn\_ls): ModuleList(

(0): GRU(300, 1024)

(1): GRU(1024, 1024)

(2): GRU(1024, 512))

(linear2global): Linear(in\_features=512, out\_features=22782, bias=True)

(linear2senses): Linear(in\_features=512, out\_features=41206, bias=True)))

INFO : Parameters:

INFO : ('module.X', torch.Size([122355, 300]), torch.float32, True)

…

('module.linear2global.weight', torch.Size([22782, 512]), torch.float32, True)

('module.linear2global.bias', torch.Size([22782]), torch.float32, True)

('module.linear2senses.weight', torch.Size([41206, 512]), torch.float32, True)

('module.linear2senses.bias', torch.Size([41206]), torch.float32, True)

INFO : Number of trainable parameters=95.0M, where

embeddings = 36.71M , softmax=11.69+21.14=32.83M, core=25.46M

Let us try to maintain the parameter budget of **95.0M**, for instance, with:

INFO : Model:

INFO : RNN(

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

(gat\_globals): GATConv(100, 50, heads=2)

(main\_rnn\_ls): ModuleList(

(0): GRU(400, 900) (1): GRU(900, 900) (2): GRU(900, 450))

(senses\_rnn\_ls): ModuleList(

(0): GRU(400, 900) (1): GRU(900, 900) (2): GRU(900, 450))

(linear2global): Linear(in\_features=450, out\_features=22782, bias=True)

(linear2senses): Linear(in\_features=450, out\_features=41206, bias=True))

INFO : Parameters:

INFO : ('E', torch.Size([122355, 300]), torch.float32, True)

('X', torch.Size([122355, 100]), torch.float32, True)

…

('gat\_globals.weight', torch.Size([100, 100]), torch.float32, True)

('gat\_globals.att', torch.Size([1, 2, 100]), torch.float32, True)

('gat\_globals.bias', torch.Size([100]), torch.float32, True)

('main\_rnn\_ls.0.weight\_ih\_l0', torch.Size([2700, 400]), torch.float32, True)

('main\_rnn\_ls.0.weight\_hh\_l0', torch.Size([2700, 900]), torch.float32, True)

('main\_rnn\_ls.0.bias\_ih\_l0', torch.Size([2700]), torch.float32, True)

('main\_rnn\_ls.0.bias\_hh\_l0', torch.Size([2700]), torch.float32, True)

…

('main\_rnn\_ls.2.weight\_ih\_l0', torch.Size([1350, 900]), torch.float32, True)

('main\_rnn\_ls.2.weight\_hh\_l0', torch.Size([1350, 450]), torch.float32, True)

…

('senses\_rnn\_ls.2.bias\_hh\_l0', torch.Size([1350]), torch.float32, True)

…

('linear2senses.bias', torch.Size([41206]), torch.float32, True)

INFO : Number of trainable parameters=98.222888M

This probably needs some fine-tuning to be effective. Nevertheless, we can try either decreasing further the GRU (maybe 800>800>400, or 800>800>500) or decreasing the GAT input to 40x2=80.

800>800>500 = 98.375M, no.

Eventually,

INFO : Model:

INFO : RNN(

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

(gat\_globals): GATConv(**100**, 50, heads=2)

(main\_rnn\_ls): ModuleList(

(0): GRU(400, **800**) (1): GRU(800, **800**) (2): GRU(800, **450**)

(senses\_rnn\_ls): ModuleList(

(0): GRU(400, 800) (1): GRU(800, 800) (2): GRU(800, 450))

(linear2global): Linear(in\_features=450, out\_features=22782, bias=True)

(linear2senses): Linear(in\_features=450, out\_features=41206, bias=True))

INFO : Parameters:

INFO : ('E', torch.Size([122355, 300]), torch.float32, True)

('X', torch.Size([122355, 100]), torch.float32, True)

…

INFO : Number of trainable parameters=94.65M, < 95.0M

Note on dimensions:

When we separate the embeddings matrix and decrease the dimensions of sense nodes, global nodes, definitions, examples, we get considerably fewer parameters:

('module.E', torch.Size([22782, 300]), torch.float32, True)

('module.X', torch.Size([122355, 100]), torch.float32, True)

So we have to re-compare with the no-GAT architecture…

### Graph creation

We review the steps in: DefineGraph.create\_graph(method, slc\_corpus)

method (here Method.FASTTEXT), single\_prototypes\_file

globals\_vocabulary\_ (fpath, df, ls)

Followed by:

X\_definitions = load\_senses\_elements(method, Utils.DEFINITIONS)  
X\_examples = load\_senses\_elements(method, Utils.EXAMPLES)  
X\_globals = torch.tensor(np.load(os.path.join(F.FOLDER\_INPUT, single\_prototypes\_file))).to(torch.float32)  
X\_senses, num\_dummysenses = initialize\_senses(X\_definitions, X\_examples, X\_globals, globals\_vocabulary\_ls, average\_or\_random\_flag=**True**)

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

This is the point we have to modify: the globals should not be the single\_prototype vectors from FastText, those are the Word Embeddings!

Options:

* completely random initialization of 122K x 100
* average of the definitions’ vectors of the word ; PCA from d300 to d100
* copy the FastText word embeddings ; PCA d300 -> d100

When we load the definitions and the examples, we are loading the vectors in

vectorized\_FastText\_definitions.npy and vectorized\_FastText\_examples.npy ,

that were computed as the average of the word vectors in each definition/example.  
(see ComputeEmbeddings.compute\_elements\_embeddings(elements\_name, method), etc.)

Problem:

If we are operating with global-node-vectors of d=100, we need to:

first, use PCA to reduce the dimensionality of definitions and examples both to 100.

Then, create the globalnode vectors of d=100

and the senses, too, should be of d=100.

This fundamentally changes how we operate on the graph.

I could pre-compute the PCA for definitions, examples, and globals, and write it into files.

Note: initializing the global node from the average of its definitions is more problematic than expected… because we have to find \*all\* the senses of the word, and then pull the definitions to compute the average.

Or maybe I don’t need to do that, I can just iterate over the indices\_table, accumulate all the definitions from the senses where get\_word\_from\_sense(wn\_id) returns the word, and take the average, thus bypassing the senseChildren edges.

Finally, we send the embeddings matrix to the RNN as a separate parameter, while we load X (122k x 100) from the graph.

## GRUs + GNN input signal # GAT debug

### Mini-experiments

With the aim to spot any evident bugs, and also to review the parameter budget.

First, **SimpleGRUs on the FastText word embeddings, without GAT:** (no freezing either)

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) )

(senses\_rnn\_ls): ModuleList(

(0): GRU(300, 1024) (1): GRU(1024, 1024) (2): GRU(1024, 512) )

(linear2global): Linear(in\_features=512, out\_features=22782, bias=True)

(linear2senses): Linear(in\_features=512, out\_features=41206, bias=True) ))

INFO : Parameters:

INFO : ('module.E', torch.Size([22782, 300]), torch.float32, True)

('module.X', torch.Size([122355, 100]), torch.float32, True)

…

INFO : Number of trainable parameters=77.36M, where we should exclude the matrix X that is unused, so 77.363 – 12.235 = **65.13**M,

where embeddings = 6.83M, softmax = 11.69+21.14=32.83M, so core = 25.47M

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch | Training | | |
|  | Globals | Senses | Multi-senses |
| 1 | 22811.83 | 41055.27 | 41475.61 |
| 100 | 28.86 | 35.31 | 51.63 |
| 200 | 28.63 | 34.91 | 51.02 |
| 250 | 11.65 | 13.28 | 18.35 |
| 300 | 1.58 | 1.61 | 1.67 |
| 350 | 1.16 | 1.12 | 1.13 |

Now, **Simple GRUs + input from GAT**

As previously, we can modify the architecture from 1024>1024>512 into 800>800>450 and keep about the same parameter budget, **64.78M** in total

|  |  |  |  |
| --- | --- | --- | --- |
| Epoch | Training | | |
|  | Globals | Senses | Multi-senses |
| 1 | 22352.0 | 40719.8 | 41084.22 |
| 50 | 29.93 | 36.77 | 54.13 |
| 100 | 28.95 | 35.4 | 51.74 |
| 200 | 28.73 | 35.1 | 51.63 |
| 300 | 28.67 | 35.02 | 51.07 |
| 380 | 22.13 | 27.61 | 38.4 |
| 383 | 19.05 | 24.08 | 34.25 |

and then it goes into early stopping.

### Experiment: Simple GRUs + GAT input (same parameter budget) on SemCor

In parallel, we set aside 2 GPUs to run a standard experiment (Simple GRUs, usual architecture, joint optimization) to double-check that there are no bugs in the rest of the code.

It gives Globals’ Validation-PPL=236.39 @ epoch 19, and Senses’ Validation-PPL=632.75 @ epoch 2**, thereby confirming that the core RNN architecture works**.

**However, the version w/GAT has a problem**:

The initial validation is ok:

INFO : Validation - Correct predictions / Total predictions:

{'correct\_g': 1, 'top\_k\_g': 7, 'tot\_g': 80640, 'correct\_all\_s': 20, 'top\_k\_all\_s': 53, 'tot\_all\_s': 80640, 'correct\_multi\_s': 20, 'top\_k\_multi\_s': 52, 'tot\_multi\_s': 29741}

INFO : Losses: Globals loss=10.04 Sense loss=10.62 Loss on multi-senses=10.62 Total loss=20.659

INFO : Perplexity: Globals perplexity=22954.16 Perplexity on all senses=40840.76 Perplexity on multi-senses=41401.3

The very first training (and validation, and all the subsequent epochs) are not ok:

Training, end of epoch 1. Global step n.576. Time = 6353.44

INFO : Training - Correct predictions / Total predictions:

{'correct\_g': 33987, 'top\_k\_g': 55130, 'tot\_g': 645120, 'correct\_all\_s': 110, 'top\_k\_all\_s': 395, 'tot\_all\_s': 645120, 'correct\_multi\_s': 0, 'top\_k\_multi\_s': 30, 'tot\_multi\_s': 232393}

INFO : Losses: Globals loss=nan Sense loss=nan Loss on multi-senses=nan Total loss=nan

INFO : Perplexity: Globals perplexity=nan Perplexity on all senses=nan Perplexity on multi-senses=nan

We should record all the batch losses we obtain, and also all the the total loss per epoch. We restart the experiment and log these values:

The initial validation run:

INFO : After training 0 epochs, validation:

INFO : min & max (predictions\_senses) = (tensor([-10.7620, -10.7445, -10.7588, ..., -10.7495, -10.7743, -10.7493], device='cuda:0'),

tensor([-10.4819, -10.4899, -10.4756, ..., -10.5025, -10.4658, -10.4971], device='cuda:0'))

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (10.03, 10.63, 10.63)

…

**Epoch n.1**:

INFO : Generator over subcorpus at TextCorpuses/My Sense-Labeled Corpus/**Training**/half\_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

INFO : min & max (predictions\_senses) = (tensor([-10.7533, -10.7523, -10.7500, ..., -10.7500, -10.7487, -10.7447],

device='cuda:0', grad\_fn=<MinBackward0>), tensor([-10.5038, -10.5035, -10.5010, ..., -10.4982, -10.5013, -10.4931],

device='cuda:0', grad\_fn=<MaxBackward0>))

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (10.03, 10.63, 10.63)

…

INFO : min & max (predictions\_senses) = (tensor([-10.7511, -10.7546, -10.7550, ..., -10.7555, -10.7521, -10.7502],

device='cuda:0', grad\_fn=<MinBackward0>), tensor([-10.4936, -10.4933, -10.4950, ..., -10.4808, -10.4921, -10.4957],

device='cuda:0', grad\_fn=<MaxBackward0>))

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (9.99, 10.61, 10.62)

INFO : senses\_in\_batch, multisenses\_in\_batch= (1120, 413)

(batch 7. After: senses\_in\_batch, multisenses\_in\_batch= (1120, 413), before (1120, 437))

INFO : min & max (predictions\_senses) = (tensor([nan, nan, nan, ..., nan, nan, nan], device='cuda:0',

grad\_fn=<MinBackward0>), tensor([nan, nan, nan, ..., nan, nan, nan], device='cuda:0',

grad\_fn=<MaxBackward0>))

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (nan, nan, nan)

INFO : senses\_in\_batch, multisenses\_in\_batch= (1120, 437)

Why do we suddenly have a nan?

Debugging through remote desktop:

Training Epoch n.1:

In batch 7, in run\_graphnet():

x\_attention\_states is entirely made of nan.

because batch\_graph.x is made of nan. Whereas batch\_graph.edge\_index is valid.

batch\_elems\_at\_t is a tensor of shape torch.Size([8, 1150])

[sample[0].item() for sample in t\_globals\_indices\_ls] is

[41329, 42550, 41250, 42523, 43127, 63419, 41773, 41224]

which all appear to be in graph range. Recalling,

Graph ranges: [0, senses, 41206, globals, 63988, defs, 94434, examples, 122355]

The fault does not appear to reside with the globals, but with the GAT. We observe that:

model.gat\_globals has att, bias, and weight made up of nan.

I must examine: model.gat\_globals.weight[0][0:5] across batches

Trying to use batch size = 8

This time, we encounter nan from the 22nd batch.

Is it the same point?? 32 \* 35 \* 7 = 7840. 8 \* 35 \* 22 = 6160, no.

At some point, it happens:

INFO : model.gat\_globals.weight[0][0:5]=tensor([-0.0081, -0.0657, 0.1131, -0.0469, -0.0689], device='cuda:1', grad\_fn=<SliceBackward>)

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (9.76, 10.46, 10.6)

INFO : senses\_in\_batch, multisenses\_in\_batch= (280, 112)

INFO : model.gat\_globals.weight[0][0:5]=tensor([nan, nan, nan, nan, nan], device='cuda:0', grad\_fn=<SliceBackward>)

[In the worst case, I can return to pre-graph batching. It’s slower, but if I am operating on SemCor alone it won’t matter that much]

Everything seemed to go well, and then it broke:

INFO : x\_indices = tensor([41589, 37088], device='cuda:1')

INFO : x\_indices = tensor([63419, 32398], device='cuda:1')

INFO : model.gat\_globals.weight[0][0:5]=tensor([ 0.1057, 0.0296, -0.0547, 0.0261, 0.0654], device='cuda:2',

grad\_fn=<SliceBackward>)

INFO : model.gat\_globals.weight[0][0:5]=tensor([ 0.1057, 0.0296, -0.0547, 0.0261, 0.0654], device='cuda:1',

grad\_fn=<SliceBackward>)

INFO : x\_indices = tensor([42739, 15661, 6380, 15667, 15652, 15666, 12266, 23345, 15665, 15656,

694, 15655, 23356, 15654, 15660, 15653, 15659, 15658, 5979, 15664,

15735, 15657, 15663, 13025, 20732, 15662, 15668, 52104, 41536, 45901,

47821, 42985], device='cuda:3')

INFO : x\_indices = tensor([41230, 33507], device='cuda:3')

INFO : x\_indices = tensor([63419, 32398], device='cuda:2')

INFO : model.gat\_globals.weight[0][0:5]=tensor([ 0.1057, 0.0296, -0.0547, 0.0261, 0.0654], device='cuda:3',

grad\_fn=<SliceBackward>)

INFO : x\_indices = tensor([42933, 24022, 1201, 24021, 1212, 16352, 44260, 44426, 43999],

device='cuda:2')

INFO : model.gat\_globals.weight[0][0:5]=tensor([ 0.1057, 0.0296, -0.0547, 0.0261, 0.0654], device='cuda:2',

grad\_fn=<SliceBackward>)

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (6.16, 6.4, 9.2)

INFO : senses\_in\_batch, multisenses\_in\_batch= (40, 14)

INFO : x\_indices = tensor([**42740**, **27879**, **44719**], device='cuda:0')

INFO : x\_indices = tensor([**41238**, **31230**], device='cuda:0')

INFO : x\_indices = tensor([**41224**, **40808**], device='cuda:2')

INFO : model.gat\_globals.weight[0][0:5]=tensor([nan, nan, nan, nan, nan], device='cuda:0', grad\_fn=<SliceBackward>)

Let’s try to run everything on only 1 GPU and see if the problem presents itself again.

With batch size=1, on only 1 GPU. This time we encounter:

INFO : x\_indices = tensor([63419, 32398], device='cuda:0')

INFO : model.gat\_globals.weight[0][0:5]=tensor([ 0.0395, 0.1515, 0.1666, -0.1707, 0.1298], device='cuda:0',

grad\_fn=<SliceBackward>)

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (5.79, 6.12, 8.62)

INFO : senses\_in\_batch, multisenses\_in\_batch= (35, 13)

INFO : x\_indices = tensor([**41230**, **33507**], device='cuda:0')

INFO : model.gat\_globals.weight[0][0:5]=tensor([nan, nan, nan, nan, nan], device='cuda:0', grad\_fn=<SliceBackward>)

…

INFO : after run: model.gat\_globals.weight[0][0:5]=tensor([-0.0564, 0.0061, 0.1128, 0.0426, 0.0620], device='cuda:0',

grad\_fn=<SliceBackward>)

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (10.0, 10.62, 10.62)

INFO : senses\_in\_batch, multisenses\_in\_batch= (1120, 413)

INFO : before run: model.gat\_globals.weight[0][0:5]=tensor([nan, nan, nan, nan, nan], device='cuda:0', grad\_fn=<SliceBackward>)

INFO : after run: model.gat\_globals.weight[0][0:5]=tensor([nan, nan, nan, nan, nan], device='cuda:0', grad\_fn=<SliceBackward>)

The bug appears to be complex and not easy to find.

I rollback the graph batching, and see if the GAT works then.

Rollbacked.

Still nan after 1120, 413.

Time to debug in detail:

At the start of the forward call, self.gat\_globals is already ruined into nan. We have to debug in detail… the previous batch, (1120, 397)

self.gat\_globals is NOT ruined into nan.

At the doors of run\_graphnet(), weights are still ok.

i\_sample=0

sample\_edge\_index = tensor([[0], [1]], device='cuda:0')

sample\_edge\_type= tensor([2], device='cuda:0')

After lemmatizing,

x\_indices=tensor([63419, 32398], device='cuda:0')

sample\_x = tensor of torch.Size([2, 100])

run, ok. Going on…

### GAT Debug …

At the point where model.gat\_globals.weight = NANs:

i\_sample=0

sample\_edge\_index.T=tensor([

[ 0, 2], [ 0, 6], [ 0, 7], [ 0, 4], [ 0, 8], [ 0, 1], [ 0, 3],

[ 0, 5], [10, 0], [ 0, 10], [12, 0], [ 0, 12], [ 0, 9], [ 9, 0],

[ 0, 11], [11, 0], [13, 0], [ 0, 13], [15, 0], [ 0, 15], [15, 0],

[ 0, 15], [15, 0], [ 0, 15], [15, 0] [ 0, 15], [ 0, 15], [15, 0],

[ 0, 14], [14, 0], [ 0, 15], [15, 0], [ 0, 14], [14, 0], [ 0, 15],

[15, 0], [ 0, 15], [15, 0], [14, 0], [ 0, 14], [14, 0], [ 0, 14]], device='cuda:3')

sample\_edge\_type=tensor([2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4,

4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4], device='cuda:3')

**t\_globals\_indices\_ls[i\_sample]**= tensor([42568, 15570, 7579, 15571, 15568, 15572, 15566, 15567, 15569, 42091, 42488, 41214, 41865, 44298, 42431, 42731], device='cuda:3')

After lemmatization, **x\_indices**= tensor([42568, 15570, 7579, 15571, 15568, 15572, 15566, 15567, 15569, 42091, 42488, 41214, 41865, 44298, 42431, 42731], device='cuda:3')

note: 42568 – 41206 = global 1362,

(in case the +-1 offset is incorrect) it is one of

1361 ripened

1362 late

1363 today

sample\_x = tensor of size (16,100), no nan-s spotted.

In model.gat\_globals, **att** and **weight** are NANs, **bias** is still numeric.

We need 2 checks:

* on whether using batch size = 1 and seq\_len = 1 causes an error at the same point
* on min and max att and weight

x\_indices = tensor([41700, 27706, 32122, 41630], device='cuda:0')

INFO : model.gat\_globals.weight[0][0].item()=nan

INFO : >>>> Problem located. Time to debug now. <<<<

It does not coincide with the other nodes that come just before the problem emerged.

So it may be a numerical problem, due to the weights of the GAT\_globals going out of range.

On the side: a fast graph-check on the newly created graph with d100, just in case:

AD.get\_node\_data(grapharea\_matrix, 41700, grapharea\_size=32, features\_mask=(True,False,False))

-> (tensor([41700: global 494: “3”, 27706:sense : three.n.01, 32122: sense: 3.dummySense.01, 41630: global 424: “three”]), None, None)

Graph check confirmed.

INFO : model.gat\_globals.weight[0][0].item()=-0.04432…

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (15.2, 14.85, 0.0)

INFO : senses\_in\_batch, multisenses\_in\_batch= (1, 0)

INFO : min, max: (model.gat\_globals.weight)=(tensor(-0.2088, device='cuda:0', grad\_fn=<MinBackward1>), tensor(0.2054, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : min, max: (model.gat\_globals.att)=(tensor(-0.2549, device='cuda:0', grad\_fn=<MinBackward1>), tensor(0.2531, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : min, max: (model.gat\_globals.bias)=(tensor(-0.0078, device='cuda:0', grad\_fn=<MinBackward1>), tensor(0.0071, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : x\_indices = tensor([42736, 38555], device='cuda:0')

INFO : model.gat\_globals.weight[0][0].item()=-0.04432…

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (7.67, 8.35, 0.0)

INFO : senses\_in\_batch, multisenses\_in\_batch= (1, 0)

INFO : min, max: (model.gat\_globals.weight)=(tensor(**nan**, device='cuda:0', grad\_fn=<MinBackward1>), tensor(nan, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : min, max: (model.gat\_globals.att)=(tensor(**nan**, device='cuda:0', grad\_fn=<MinBackward1>), tensor(nan, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : min, max: (model.gat\_globals.bias)=(tensor(-0.0078, device='cuda:0', grad\_fn=<MinBackward1>), tensor(0.0071, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : x\_indices = tensor([41700, 27706, 32122, 41630], device='cuda:0')

INFO : model.gat\_globals.weight[0][0].item()=nan

42736 = global 1530, “1.24”

38555 = sense, “1.24.dummySense.01”

More in detail:

INFO : run\_graphnet, AFTER for cycle:

INFO : min, max: (model.gat\_globals.weight)=(tensor(-0.2035, device='cuda:0', grad\_fn=<MinBackward1>), tensor(0.1938, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : min, max: (model.gat\_globals.att)=(tensor(-0.2496, device='cuda:0', grad\_fn=<MinBackward1>), tensor(0.2367, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : min, max: (model.gat\_globals.bias)=(tensor(-0.0078, device='cuda:0', grad\_fn=<MinBackward1>), tensor(0.0077, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : loss\_global, loss\_all\_senses, loss\_multi\_senses= (7.15, 8.2, 0.0)

INFO : senses\_in\_batch, multisenses\_in\_batch= (1, 0)

INFO : run\_graphnet, BEFORE for cycle:

INFO : min, max: (model.gat\_globals.weight)=(tensor(nan, device='cuda:0', grad\_fn=<MinBackward1>), tensor(nan, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : min, max: (model.gat\_globals.att)=(tensor(nan, device='cuda:0', grad\_fn=<MinBackward1>), tensor(nan, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : min, max: (model.gat\_globals.bias)=(tensor(-0.0078, device='cuda:0', grad\_fn=<MinBackward1>), tensor(0.0077, device='cuda:0', grad\_fn=<MaxBackward1>))

INFO : x\_indices = tensor([41700, 27706, 32122, 41630], device='cuda:0')

Using torch.autograd.set\_detect\_anomaly(**True**),

**Warning: Error detected in MulBackward0**. Traceback of forward call that caused the error:

File "<stdin>", line 1, in <module>

File "/home/andrealk3/ALMA\_PhD/Code/TASK1\_Gbwe/NN/Training.py", line 225, in run\_train

multisense\_globals\_set, verbose)

File "/home/andrealk3/ALMA\_PhD/Code/TASK1\_Gbwe/NN/Loss.py", line 81, in **compute\_model\_loss**

predictions\_globals, predictions\_senses = model(batch\_input)

File "/home/andrealk3/venvs/torch15/lib/python3.6/site-packages/torch/nn/modules/module.py", line 550, in \_\_call\_\_

result = self.forward(\*input, \*\*kwargs)

File "/home/andrealk3/ALMA\_PhD/Code/TASK1\_Gbwe/NN/Models/RNNs.py", line 159, in forward

**t\_g\_nodestates = run\_graphnet(t\_input\_lts, batch\_elems\_at\_t,t\_globals\_indices\_ls, CURRENT\_DEVICE, self)**

File "/home/andrealk3/ALMA\_PhD/Code/TASK1\_Gbwe/NN/Models/RNNs.py", line 33, in run\_graphnet

**x\_attention\_states = model.gat\_globals(sample\_x, edge\_index)**

File "/home/andrealk3/venvs/torch15/lib/python3.6/site-packages/torch/nn/modules/module.py", line 550, in \_\_call\_\_

result = self.forward(\*input, \*\*kwargs)

File "/home/andrealk3/venvs/torch15/lib/python3.6/site-packages/torch\_geometric/nn/conv/gat\_conv.py", line 91, in forward

**return self.propagate(edge\_index, size=size, x=x)**

File "/home/andrealk3/venvs/torch15/lib/python3.6/site-packages/torch\_geometric/nn/conv/message\_passing.py", line 258, in propagate

**out = self.message(\*\*msg\_kwargs)**

File "/home/andrealk3/venvs/torch15/lib/python3.6/site-packages/torch\_geometric/nn/conv/gat\_conv.py", line 108, in message

return x\_j \* alpha.view(-1, self.heads, 1)

(print\_stack at /pytorch/torch/csrc/autograd/python\_anomaly\_mode.cpp:60)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

**File "/home/andrealk3/ALMA\_PhD/Code/TASK1\_Gbwe/NN/Training.py", line 243, in run\_train**

**loss.backward()**

File "/home/andrealk3/venvs/torch15/lib/python3.6/site-packages/torch/tensor.py", line 198, in backward

**torch.autograd.backward(self, gradient, retain\_graph, create\_graph)**

File "/home/andrealk3/venvs/torch15/lib/python3.6/site-packages/torch/autograd/\_\_init\_\_.py", line 100, in backward

allow\_unreachable=True) # allow\_unreachable flag

**RuntimeError: Function 'MulBackward0' returned nan values in its 0th output.**

Again, using bsz=32 X seq\_len=35:

… x\_indices = tensor([41875, 31670], device='cuda:0')

model.gat\_globals.weight[0][0].item()=0.018677160143852234

run\_graphnet, After for cycle:

min, max: (model.gat\_globals.weight)=(tensor(-0.1733, device='cuda:0', grad\_fn=<MinBackward1>), tensor(0.1734, device='cuda:0', grad\_fn=<MaxBackward1>))

loss\_global, loss\_all\_senses, loss\_multi\_senses= (10.0, 10.6, 10.62)

senses\_in\_batch, multisenses\_in\_batch= (1120, 413)

BEFORE loss.backward(): model.gat\_globals.weight[0][0].item()=0.018677160143852234

Warning: Error detected in MulBackward0.

from comment: “Could you check all parameters and gradients for invalid values?  
You could use torch.isfinite(tensor) to check for valid values.”

Let us examine the parameters of the model:

[(name, param.shape, param.dtype,

torch.isfinite(param.data).all(), torch.isnan(param.data).any())

**for** (name, param) **in** model.named\_parameters()]

We expect: True for isfinite.all(), and False for isnan.any():

Almost everything is ok, *except*:

('X', torch.Size([122355, 100]), torch.float32, **tensor(False)**, **tensor(True)**)

So, we may have NaN because the graph matrix X contains it. And inf, too.

Can we find where NaN is in X?

Recalling: Graph ranges: [0, **senses**, 41206, globals, 63988, defs, 94434, examples, 122355]

isnan\_rows = <class 'list'>: [30578=mrs.\_macready.dummySense.01, 30601=mr.\_gerosa.dummySense.01,

30620=a.\_m.\_a..dummySense.01,

30629=u.n.f.p..dummySense.01,

30642=john\_h.\_mercer.dummySense.01,

30721=3.5.dummySense.01,

30727, 30758, 30864, 30870, 30942, 30958, 31019, 31025, 31060, 31061, … , 40873,

40900,40903,40940,40963,41065,41105,41134,41202]

Can we find where inf is in X? [30578, 30601, 30620, 30629, 30642, 30721, 30727, 30758, 30864, 30870, 30942, 30958, 31019, 31025, 31060, 31061, 31084, 31206, 31246, 31307, 31349, 31379, 31392, 31404, 31426, 31451, 31461, 31482, 31496, 31513, 31526, 31575, 31607, 31631, 31653, 31696, 31807, 31827, 31972, 31978, 31988, 32023, 32067, 32092, 32100, 32109, 32133, 32137, 32230, 32235, 32276, 32298, 32312, 32327, 32355, 32394, 32397, 32443, 32452, 32454, 32493, 32625, 32702, 32805, 32817, 32837, 32853, 32869, 32909, 32937, 32944, 32950, 32982, 33000, 33032, 33040, 33063, 33123, 33182, 33287, 33347, 33349, 33352, 33399, 33404, 33409, 33459, 33471, 33503, 33505, 33519, 33567, 33585, 33658, 33684, 33710, 33724, 33851, 33875, 33891...

The rows that are not finite are the same, because nan is not counted as finite.

In practice, we get isinf\_rows == isnan\_rows : True.

### Debugging the graph (senses with dots)

The NaN problem is caused by dummySenses with dots, like ‘john\_h.\_mercer.dummySense.01’.

If

wn\_id= 'mrs.\_macready.dummySense.01'

pos = mtc.group(0)[1:-1] = '\_macready'

I do not see the dummySense, and then it breaks.

I have to modify the pattern from

pt = **r'\.([^.])+\.'**

into re.search(**r'\.([^.])+\.([0-9])+'**, db\_row[0])

After modifying that, we load the new graph and check if it has NaNs / infs.

This time, we haveisnan\_rows=

<class 'list'>: [30446, 30447, 30448, 30449, 30450, 30451, 30452, 30453, 30454, 30455, 30456, 30457, 30458, 30459, 30460, 30461, 30462, 30463, 30464, 30465, 30466, 30467, 30468, 30469, 30470, 30471, 30472, 30473, 30474, 30475, 30476, 30477, 30478, 30479, 30480, 30481, 30482, 30483, 30484, 30485, 30486, 30487, 30488, 30489, 30490, 30491, 30492, 30493, 30494, 30495, 30496, 30497, 30498, 30499, 30500, 30501, 30502, 30503, 30504, 30505, 30506, 30507, 30508, 30509, 30510, 30511, 30512, 30513, 30514, 30515, 30516, 30517, 30518, 30519, 30520, 30521, 30522, 30523, 30524, 30525, 30526, 30527, 30528, 30529, 30530, 30531, 30532, 30533, 30534, 30535, 30536, 30537, 30538, 30539, 30540, 30541, 30542, 30543, 30544, 30545...41205]

It's evident that the pattern does not work. We must readjust.

From wn\_id=zurich.n.01, we get pos=’n.0’ whereas we should only get ‘n’

We use a pattern with a lookahead, pt = **r'\.([^.])+\.(?=([0-9])+)'**

isnan\_rows=[].

Will try to run again the Simple GRUs + GAT experiment.

## Simple GRUs + GNN input signal

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals: **Main GRU** with 3 layers (800>800>450) | 1) The word embedding of the current global *wi* (d=300)  2) The state vector of the global node *I*, obtained applying a GNN (GAT) on the dictionary graph. (d=100). | batch\_size=**32**  TBPTT length=35 |
| Senses: **GRU** with 3 layers, as above. | grapharea=32, hops=1  learning rate=0.00005 |
| GAT has 2 heads, 2x50 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training* | | | *Validation* | | |
|  | Globals | Senses | Multi-senses | Globals | Senses | Multi-senses |
| 1 | 657.37 | 1068.85 | 11272.16 | 509.27 | 652.2 | 11401.9 |
| 2 | 421.17 | 594.35 | 8190.23 | 515.07 | **634.42** | 10344.81 |
| 3 | 401.2 | 588.25 | 7954.03 | 437.35 | 637.28 | 10287.84 |
| 4 | 332.88 | 588.4 | 7953.34 | 383.8 | 643.32 | 10598.24 |
| 5 | 280.39 | 585.82 | 7916.06 | 335.16 | 652.88 | 10895.56 |
| 6 | 243.87 | 580.57 | 7828.25 | 306.84 | 660.4 | 11087.77 |
| 7 | 219.61 | 574.06 | 7690.93 | 287.78 | 666.66 | 11270.57 |
| 8 | 201.67 | 566.58 | 7535.54 | 273.95 | 673.15 | 11451.97 |
| 9 | 188.13 | 558.36 | 7361.2 | 264.16 | 679.51 | 11582.22 |
| 10 | 177.54 | 549.24 | 7162.98 | 257.07 | 685.58 | 11714.25 |
| 11 | 168.84 | 539.33 | 6944.87 | 251.93 | 691.11 | 11839.77 |
| 12 | 161.46 | 528.96 | 6718.69 | 248.13 | 696.32 | 11953.58 |
| 13 | 154.97 | 518.31 | 6488.05 | 245.37 | 701.47 | 12086.85 |
| 14 | 149.18 | 507.55 | 6257.6 | 243.27 | 706.59 | 12219.87 |
| 15 | 143.95 | 496.89 | 6031.86 | 241.77 | 711.94 | 12363.8 |
| 16 | 139.18 | 486.4 | 5811.29 | 240.57 | 717.12 | 12492.02 |
| 17 | 134.8 | … | … | 239.71 | … | … |
| 18 | 130.74 | … | … | 239.01 | … | … |
| 19 | 126.98 | … | … | 238.51 | … | … |
| 20 | 123.44 |  |  | 238.16 |  |  |
| 21 | 120.12 |  |  | 237.83 |  |  |
| 22 | 117.03 |  |  | 237.49 |  |  |
| 23 | 114.18 |  |  | **237.3** |  |  |
| 24 | 111.36 |  |  | 237.35 |  |  |
| 25 | 108.64 |  |  | 237.88 | 779.89 | 14726.01 |

Valid-PPL results, for the 800>800>450 + GAT(50x2) architecture: