[Preliminary information (Datasets, graphs) 4](#_Toc48234635)

[SemCor 4](#_Toc48234636)

[SemCor stats 4](#_Toc48234637)

[Graph (min\_freq=2, with dummySense) 4](#_Toc48234638)

[Graph (vocab. min. frequency=2, no dummy Sense) 5](#_Toc48234639)

[Graph (vocab. min. frequency=2, with dummy Sense) 6](#_Toc48234640)

[Meetings 7](#_Toc48234641)

[IA 7](#_Toc48234642)

[IA, PSK, KF 7](#_Toc48234643)

[Meeting with IA, 12/05 8](#_Toc48234644)

[Multi-Sense LM. Part 0 9](#_Toc48234645)

[Structured prediction – select from K globals 15](#_Toc48234646)

[Design 15](#_Toc48234647)

[Mini-experiment – overfit on fragment of SemCor 16](#_Toc48234648)

[Graph check 17](#_Toc48234649)

[Senses.SelectK retrieval and selection check 18](#_Toc48234650)

[Modifications to SelectK 20](#_Toc48234651)

[Bugs & co 21](#_Toc48234652)

[Multi-sense LM. Part 1 23](#_Toc48234653)

[Gradient and loss 23](#_Toc48234654)

[GRUbase – debug on mini-experiments 24](#_Toc48234655)

[Overfitting base GRU on SemCor globals 31](#_Toc48234656)

[Preliminary experiment – GRUbase2 w/senses on SemCor 33](#_Toc48234657)

[Model Verification and Modification 35](#_Toc48234658)

[The loss when operating on senses 35](#_Toc48234659)

[Considerations 36](#_Toc48234660)

[Graph retrieval 37](#_Toc48234661)

[Boundaries of graph node types 37](#_Toc48234662)

[Analysis… 37](#_Toc48234663)

[Senses’ +1 displacement 39](#_Toc48234664)

[Adding lemmatization for the global node 40](#_Toc48234665)

[Structured prediction: SelectK – version 1 41](#_Toc48234666)

[Model 41](#_Toc48234667)

[Mini-Experiment 1 – Overfit on fragment of SemCor 42](#_Toc48234668)

[Losses 42](#_Toc48234669)

[Mini-experiment 1 – Analysis of the predictions 43](#_Toc48234670)

[Mini-Experiment 2 – Overfit on fragment of SemCor 44](#_Toc48234671)

[Losses 44](#_Toc48234672)

[Mini-experiment 2 – Analysis of the predictions 45](#_Toc48234673)

[Mini-experiment 3 47](#_Toc48234674)

[Losses 47](#_Toc48234675)

[Mini-experiment 3 – Analysis of the predictions 47](#_Toc48234676)

[Adding global-to-sense edges from the SLC 49](#_Toc48234677)

[Analysis of predictions – why do we have no senses for ‘act’, SensePerplexity=nan, etc. 49](#_Toc48234678)

[Mini-experiment 4 51](#_Toc48234679)

[Losses 51](#_Toc48234680)

[Mini-experiment 4 – Analysis of the predictions 51](#_Toc48234681)

[Mini-experiment 5 53](#_Toc48234682)

[Losses 53](#_Toc48234683)

[Mini-experiment 5 – Analysis of the predictions 54](#_Toc48234684)

[Experiment 1 – SelectK1 on SemCor 55](#_Toc48234685)

[Multi-sense LM. Part 2 57](#_Toc48234686)

[Checking & Debugging 57](#_Toc48234687)

[Baseline 0: 2 GRUs 58](#_Toc48234688)

[Model 58](#_Toc48234689)

[Mini-experiment: overfit on fragment of SemCor 59](#_Toc48234690)

[Experiment: GRUs on SemCor 59](#_Toc48234691)

[Experiment: GRUs on SemCor – version 2 60](#_Toc48234692)

[SelectK5 61](#_Toc48234693)

[Model 61](#_Toc48234694)

[Mini-experiment: overfit on fragment of SemCor 62](#_Toc48234695)

[Experiment: SelectK5 on SemCor 63](#_Toc48234696)

[Recording the number of correct predictions 64](#_Toc48234697)

[Batches bug while reading sense-labeled corpus 64](#_Toc48234698)

[Again on the number of correct predictions 69](#_Toc48234699)

[SelectK – with number of correct predictions 69](#_Toc48234700)

[SelectK1 on SemCor 70](#_Toc48234701)

[Multi-sense LM, part 3 73](#_Toc48234702)

[Reviewing the Vocabulary for SemCor 73](#_Toc48234703)

[Meeting with IA, 23/06 73](#_Toc48234704)

[Modifications: dummy sense label, pre-lemmatization, PPL 74](#_Toc48234705)

[Dummy sense label 74](#_Toc48234706)

[Pre-lemmatization 75](#_Toc48234707)

[Checks 75](#_Toc48234708)

[Creating the graph and the grapharea\_matrix 75](#_Toc48234709)

[Reviewing graph & edge creation 79](#_Toc48234710)

[Operating on a part of SemCor 81](#_Toc48234711)

[On the entirety of SemCor, new statistics 83](#_Toc48234712)

[Graph retrieval 84](#_Toc48234713)

[Perplexity 86](#_Toc48234714)

[Globals with =/>1 senses 86](#_Toc48234715)

[Loss computation and perplexity 87](#_Toc48234716)

[Multi-Sense LM, part 4 88](#_Toc48234717)

[Architectures 88](#_Toc48234718)

[LSTM, version 0 89](#_Toc48234719)

[Debug 89](#_Toc48234720)

[2LSTMs: Model 90](#_Toc48234721)

[2LSTMs: Experiment 0 on SemCor 91](#_Toc48234722)

[Observations 92](#_Toc48234723)

[Check predictions & labels, Parameters & Debug 93](#_Toc48234724)

[Sense labels 93](#_Toc48234725)

[LSTM, version 1 94](#_Toc48234726)

[2LSTMs, Mini-Experiment 1, overfit on fragment of SemCor 94](#_Toc48234727)

[2LSTMs: Experiment 1 on SemCor 95](#_Toc48234728)

[LSTM, version 2 97](#_Toc48234729)

[2LSTMs, Mini-Experiment 2, overfit on fragment of SemCor 97](#_Toc48234730)

[2LSTMs: Experiment 2 on SemCor 99](#_Toc48234731)

[LSTM, version 3 100](#_Toc48234732)

[2LSTMs: Experiment 3 on SemCor 100](#_Toc48234733)

[Recap of experiments made on SemCor, and new directions 102](#_Toc48234734)

[Verifying the RNN setup on the standard LM task 105](#_Toc48234735)

[Table & Graph – WT-2 105](#_Toc48234736)

[Mini-experiment on fragment of WT-2 106](#_Toc48234737)

[Model 106](#_Toc48234738)

[Mini-exp A 107](#_Toc48234739)

[2LSTMs: Standard LM on WT-2 109](#_Toc48234740)

[Experiment A 109](#_Toc48234741)

[Model B 111](#_Toc48234742)

[Mini-exp B 111](#_Toc48234743)

[Experiment B 112](#_Toc48234744)

[Standard LM task on SemCor 114](#_Toc48234745)

[Table & Graph 114](#_Toc48234746)

[Mini-experiment on fragment of SemCor 115](#_Toc48234747)

[Model A 115](#_Toc48234748)

[Mini-exp A 116](#_Toc48234749)

[Experiment A 118](#_Toc48234750)

[Experiment B 119](#_Toc48234751)

[Model 119](#_Toc48234752)

[Mini-exp B 119](#_Toc48234753)

[Experiment B 120](#_Toc48234754)

[Experiment C 121](#_Toc48234755)

[Model 121](#_Toc48234756)

[Mini-exp C 121](#_Toc48234757)

[Experiment C 122](#_Toc48234758)

[Observations 123](#_Toc48234759)

[On the results 123](#_Toc48234760)

[Reviewing the input loading and vocabulary 123](#_Toc48234761)

[Next steps 126](#_Toc48234762)

[Replicating the old result (lowercased, <num>, Vocab from Training set) 127](#_Toc48234763)

[Graph 127](#_Toc48234764)

[Model 127](#_Toc48234765)

[Experiment 128](#_Toc48234766)

[Gathering more Sense-Labeled Data 129](#_Toc48234767)

[Review of Sense-Labeled Corpuses 129](#_Toc48234768)

[Freezing standard LM and then proceeding to Senses 131](#_Toc48234769)

[Notes & ideas 131](#_Toc48234770)

[Implementation 133](#_Toc48234771)

[Mini-experiment with freezing on SemCor, v. 1.0 133](#_Toc48234772)

[Mini-experiment with freezing on SemCor, v. 1.1 134](#_Toc48234773)

[Mini-experiment with freezing on SemCor, v. 1.2 136](#_Toc48234774)

# Preliminary information (Datasets, graphs)

## SemCor

### SemCor stats

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

Training tokens: 646,038

Validation tokens: 80,760

Mini-dataset 1 : 180 tokens.

Mini-dataset 2: 63 tokens.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

get\_edges\_selfloops:

INFO : []

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

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

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

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

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

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

Constructing X, matrix of node features

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

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

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

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

Defining the edges: def, exs

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

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

Defining the edges: sc

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

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

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

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

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

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

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

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

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

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

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

Graph ranges:

senses: [0,43559)

globals: [43559,69252)

definitions: [69252,98385)

examples: [98385,126462)

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

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

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

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

INFO : Defining the edges: sc

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

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

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

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

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

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

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

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

INFO : []

INFO : len(edges\_ls)==0

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

INFO : Defining the edges: syn, ant

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

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

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

# Meetings

## IA

* Alternative: simple selection from best globals, no self-attention, also works as estimate [the current simple selection overfits on fragment, but it does not give good results]
* Alternative: proceed in parallel with senses’ model and the selection from globals, and then make the probability derived from both, e.g. with a product
* Additional: when I have stable results, it would be relevant to re-run *everything* using Distil/Al-BERT’s pre-trained embreddings
* For the section meeting, I can present the whole project instead of focusing too much on what I am doing now – possibly also using the poster for the Innovation Foundation
  + It can be 10/11 to 20 minutes
  + I can also explain what different senses are, and show how SemCor works

## IA, PSK, KF

* Transformer-LM: plug in the filter that handles partial words / word completion.
* To improve performance on WikiText-2, I can use:
  + better/larger GRU
  + AWD-LSTM
  + Pre-trained embeddings from DistilBERT instead of FastText
* (How does Perplexity on Senses compare to standard Perplexity:  
  hp: the number of tokens that we compute PPL on. (e.g. PPL on text corpus of 2Mln will be different from PPL on text corpus of 103Mln words)  
  hp2: I could consider all the words that have a sense-specification on SemCor, and then compute the PPL over WikiText-2 only for those words – how does it compare with the standard PPL on WikiText-2?  
  note: PPL is difficult to compare between different datasets and tasks. Since we are in a new task, we can also just use our measure. (WSD uses F1-score))
* Implement structured prediction from globals to senses, as a baseline for the senses part of the task
* (Go on with self-attention for the senses task)

# Meeting with IA, 12/05

(Reordered) list:

* Rerun MultiSense Evaluation, using the new GRU in the architecture
  + select k=few eg. 5 globals, consider their senses, and choose among them.  
      
    question: I still need to open up to the senses’ logits; I would only get a distribution over the e.g. 20 senses from the k globals. Should I copy paste the logits over the senses’ logits, keeping everything at 0?   
    I can multiply the senses’logits (obtained, for instance, with a standard GRU side-architecture) per what I get from here. Everything not selected would be x0.
  + Compare with alternatives (e.g. Multi-sense alternatives)
* Standard Language Modeling:
  + AWD-LSTM implementation
  + Mogrifier LSTM – search for PyTorch implementation

# Multi-Sense LM. Part 0

**Baseline 1: 2 GRUs, shared first layer – bugs in the model were not solved**

**Model**

DataParallel(

(module): GRU\_base(

(maingru\_ls): ModuleList(

(0): GRU(300, 1150)

(1): GRU(1150, 1150)

(2): GRU(1150, 1150)

)

(gru\_senses): GRU(1150, 1150)

(linear2global): Linear(in\_features=1150, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=1150, out\_features=25986, bias=True))

)

Parameters:

('module.X', torch.Size([99963, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([6]), torch.int64, False)

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

('module.memory\_hn', torch.Size([3, 4, 1150]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([1, 4, 1150]), torch.float32, False)

('module.maingru\_ls.0.weight\_ih\_l0', torch.Size([3450, 300]), torch.float32, True)

('module.maingru\_ls.0.weight\_hh\_l0', torch.Size([3450, 1150]), torch.float32, True)

('module.maingru\_ls.0.bias\_ih\_l0', torch.Size([3450]), torch.float32, True)

('module.maingru\_ls.0.bias\_hh\_l0', torch.Size([3450]), torch.float32, True)

('module.maingru\_ls.1.weight\_ih\_l0', torch.Size([3450, 1150]), torch.float32, True)

('module.maingru\_ls.1.weight\_hh\_l0', torch.Size([3450, 1150]), torch.float32, True)

('module.maingru\_ls.1.bias\_ih\_l0', torch.Size([3450]), torch.float32, True)

('module.maingru\_ls.1.bias\_hh\_l0', torch.Size([3450]), torch.float32, True)

('module.maingru\_ls.2.weight\_ih\_l0', torch.Size([3450, 1150]), torch.float32, True)

('module.maingru\_ls.2.weight\_hh\_l0', torch.Size([3450, 1150]), torch.float32, True)

('module.maingru\_ls.2.bias\_ih\_l0', torch.Size([3450]), torch.float32, True)

('module.maingru\_ls.2.bias\_hh\_l0', torch.Size([3450]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l0', torch.Size([3450, 1150]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l0', torch.Size([3450, 1150]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l0', torch.Size([3450]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l0', torch.Size([3450]), torch.float32, True)

('module.linear2global.weight', torch.Size([21988, 1150]), torch.float32, True)

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

('module.linear2senses.weight', torch.Size([25986, 1150]), torch.float32, True)

('module.linear2senses.bias', torch.Size([25986]), torch.float32, True)

Number of trainable parameters=114.04M, where softmax=29.90M + 25.28M=55.19M, embeddings=29.99M, core = 28.86M

**Mini-experiment on fragment of SemCor**

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| For the globals:  **GRU** w/ 3 layers, 1150x3 (layer 1 shared).  Followed by **FF-NN** to logits | 1. The word embedding of the current global   (d=300) | batch\_size=4 |
| For the senses:  **1 shared layer** of the main GRU > **1-layer GRU** -> **FF-NN** to logits | TBPTT length=8 |
|  | learning rate=10^(**-3**) |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |
| 1 | 4918.23 | 10418.61 |
| 2 | 105.08 | 134.04 |
| 3 | 77.19 | 68.57 |
| 4 | 65.73 | 56.92 |
| 5 | 66.21 | 54.71 |
| 10 | 62.92 | 51.64 |
| 30 | 62.92 | 19.82 |
| 50 | 56.17 | 4.42 |
| 75 | 37.24 | 1.44 |
| 100 | 23.13 | 1.1 |
| 150 | 9.02 | 1.03 |
| 200 | 3.57 | 1.01 |

**Experiment –GRUs w/ shared layer on SemCor (fault: not detaching the gru\_senses’ memory)**

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| For the globals:  **GRU** w/ 3 layers, 1150x3 (layer 1 shared).  Followed by **FF-NN** to logits | 1. The word embedding of the current global   (d=300) | batch\_size=40 |
| For the senses:  **1 shared layer** of the main GRU > **1-layer GRU** -> **FF-NN** to logits | TBPTT length=35 |
|  | learning rate=0.5\*10^(**-4**) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |  | *Validation perplexity on globals* | *Validation perplexity on senses* |
| 1 | 971.96 | 5169.47 |  | 490.31 | 835.0 |
| 2 | 540.3 | 3432.66 |  | 347.28 | **804.95** |
| 3 | 418.94 | 3129.2 |  | 315.93 | 831.75 |
| 4 | 373.93 | 2947.42 |  | 299.93 | 829.01 |
| 5 | 344.06 | 2821.13 |  | 289.18 | 886.89 |
| 6 | 321.49 | 2768.63 |  | 283.92 | 1034.11 |
| 7 | 302.41 | 3269.6 |  | 278.56 | 914.73 |
| 8 | 285.49 | 2665.07 |  | 270.99 | 1807.85 |
| 9 | 270.04 | 2528.11 |  | 267.79 | 1112.29 |
| 10 | 256.36 | 2502.85 |  | 262.02 | 970.6 |
| 11 | 243.94 | 2391.66 |  | 263.61 | 1200.81 |
| 12 | 232.49 | 2373.34 |  | 258.92 | 1399.38 |
| 13 | 221.78 | 2252.45 |  | 252.47 | 2179.9 |
| 14 | 211.73 | 2222.98 |  | 251.66 | 1309.55 |
| 15 | 202.54 | 2070.02 |  | 250.55 | 2772.74 |
| 16 | 193.66 | 1950.52 |  | 247.19 | 2650.16 |
| 17 | 185.68 | 1891.93 |  | 248.58 | 3095.87 |
| 18 | 177.93 | 1835.95 |  | 247.61 | 2480.83 |
| 19 | 170.97 | 1809.4 |  | 245.7 | 1232.44 |
| 20 | 164.49 | 1725.47 |  | 245.27 | 2917.69 |
| 21 | 158.3 | 1641.77 |  | 244.83 | 1680.78 |
| 22 | 152.17 | 1589.13 |  | 245.41 | 2118.22 |
| 23 | 146.72 | 1569.21 |  | 244.53 | 2307.36 |
| 24 | 140.69 | 1497.27 |  | 245.31 | 1977.77 |
| 25 | 134.79 | 1427.68 |  | 244.14 | 3117.87 |
| 26 | 129.49 | 1348.3 |  | 243.1 | 2807.55 |
| 27 | 124.34 | 1311.9 |  | **242.35** | 2739.16 |
| 28 | 119.55 | 1289.07 |  | 243.71 | 3153.67 |
| 29 | 115.16 | 1264.97 |  | 244.83 | 3012.75 |
| 30 | 110.88 | 1209.65 |  | 245.65 | 3195.36 |

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| For the globals:  **GRU** w/ 3 layers, 1150x3 (layer 1 shared).  Followed by **FF-NN** to logits | 1. The word embedding of the current global   (d=300) | batch\_size=40 |
| For the senses:  **1 shared layer** of the main GRU > **1-layer GRU** -> **FF-NN** to logits | TBPTT length=35 |
|  | **learning rate=10^(-5)** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |  | *Validation perplexity on globals* | *Validation perplexity on senses* |
| 1 | 1612.97 | 7651.04 |  | 522.04 | 1105.27 |
| 2 | 736.76 | 3579.03 |  | 510.6 | 1007.8 |
| 3 | 725.9 | 3486.44 |  | 510.74 | 983.15 |
| 4 | 723.02 | 3457.18 |  | 511.13 | **981.36** |
| 5 | 721.14 | 3443.05 |  | 508.15 | 991.13 |
| 6 | 720.06 | 3430.39 |  | 510.9 | 1001.09 |
| 7 | 719.66 | 3422.33 |  | 511.41 | 991.3 |
| 8 | 719.17 | 3410.85 |  | 508.73 | 1013.25 |
| 9 | 718.67 | 3410.02 |  | 511.87 | 1021.22 |
| 10 | 718.51 | 3407.2 |  | 508.25 | 1028.66 |
| 20 | 401.88 | 3258.57 |  | 309.43 | 1013.5 |
| 30 | 326.42 | 3116.9 |  | 274.1 | 1151.66 |
| 40 | 296.26 | 2955.15 |  | 267.23 | 1646.8 |
| 50 | 269.84 | 2696.74 |  | **263.68** | 1682.31 |
| 60 | 247.95 | 2453.59 |  | 264.04 | 2097.26 |

Observations:

a learning rate of 10^(-5) is too low. 10^(-4) is fast. Grid search suggests 0.5\*10^(-4)

I did not detach the senses’ memory!

**Experiment –GRUs w/ shared layer on SemCor**

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| For the globals:  **GRU** w/ 3 layers, 1150x3 (layer 1 shared).  Followed by **FF-NN** to logits | 1. The word embedding of the current global   (d=300) | batch\_size=40 |
| For the senses:  **1 shared layer** of the main GRU > **1-layer GRU** -> **FF-NN** to logits | TBPTT length=35 |
|  | learning rate=**0.5\*10^(-4)** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |  | *Validation perplexity on globals* | *Validation perplexity on senses* |
| 1 | 974.01 | 5184.52 |  | 495.82 | 835.31 |
| 2 | 543.19 | 3431.93 |  | 343.02 | **809.3** |
| 3 | 420.57 | 3115.14 |  | 316.4 | 830.73 |
| 4 | 375.15 | 2931.76 |  | 300.37 | 849.11 |
| 5 | 345.35 | 2806.64 |  | 289.39 | 919.22 |
| 6 | 322.07 | 2755.25 |  | 283.68 | 1344.15 |
| 7 | 302.68 | 2966.38 |  | 275.61 | 914.12 |
| 8 | 285.5 | 2683.41 |  | 269.84 | 1144.84 |
| 9 | 269.57 | 2614.12 |  | 266.23 | 1091.96 |
| 10 | 255.34 | 2522.42 |  | 260.59 | 1562.29 |
| 11 | 242.45 | 2525.91 |  | 262.01 | 1201.69 |
| 12 | 230.17 | 2335.81 |  | 255.8 | 3216.74 |
| 13 | 218.73 | 2203.31 |  | 249.31 | 1609.48 |
| 14 | 208.14 | 2198.41 |  | 248.36 | 2093.41 |
| 15 | 198.53 | 2077.28 |  | 247.43 | 1538.86 |
| 16 | 189.53 | 2036.01 |  | 242.89 | 2184.96 |
| 17 | 181.32 | 2029.97 |  | 243.98 | 1695.24 |
| 18 | 173.23 | 1921.61 |  | 241.94 | 2423.03 |
| 19 | 165.53 | 1774.17 |  | 241.25 | 2554.37 |
| 20 | 158.48 | 1748.05 |  | 241.11 | 1825.42 |
| 21 | 152.1 | 1682.55 |  | 240.31 | 2533.01 |
| 22 | 146.39 | 1696.8 |  | 239.08 | 1760.2 |
| 23 | 141.34 | 1531.6 |  | 239.15 | 2551.63 |
| 24 | 136.85 | 1468.94 |  | 240.69 | 3231.04 |
| 25 | 132.03 | 1453.37 |  | 239.0 | 2714.26 |
| 26 | 126.83 | 1346.67 |  | 238.2 | 3285.9 |
| 27 | 122.56 | 1297.54 |  | **237.42** | 2934.18 |
| 28 | 118.1 | 1255.5 |  | 238.0 | 2678.2 |
| 29 | 113.91 | 1216.16 |  | 239.22 | 3284.78 |
| 30 |  |  |  |  |  |

**Baseline 2 – 2GRUs (x3 and x2), no shared layers**

**Model**

DataParallel(

(module): GRU\_base2(

(maingru\_ls): ModuleList(

(0): GRU(300, 1150)

(1): GRU(1150, 1150)

(2): GRU(1150, 1150)

)

(gru\_senses): GRU(300, 1150, num\_layers=2)

(linear2global): Linear(in\_features=1150, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=1150, out\_features=25986, bias=True)

)

)

Number of trainable parameters=119.05M, where softmax=29.90M + 25.28M=55.19M, embeddings=29.99M, core = 33.87M

**Variant**: Including the sense node-state among the input signals:

DataParallel(

(module): GRU\_base2(

(maingru\_ls): ModuleList(

(0): GRU(600, 1150)

(1): GRU(1150, 1150)

(2): GRU(1150, 1150)

)

(gru\_senses): GRU(600, 1150, num\_layers=2)

(gat\_senses): GATConv(300, 75, heads=4)

(linear2global): Linear(in\_features=1150, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=1150, out\_features=25986, bias=True)

)

)

Number of trainable parameters=121.21M, where core=36.03M

**Experiment – 2 GRUs on SemCor**

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| For the globals:  **GRU** w/ 3 layers, 1150x3 (layer 1 shared).  Followed by **FF-NN** to logits | 1. The word embedding of the current global   (d=300) | batch\_size=40  TBPTT length=35 |
| For the senses:  **GRU** w/ 2 layers -> **FF-NN** to logits | learning rate=**0.5\*10^(-4)** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |  | *Validation perplexity on globals* | *Validation perplexity on senses* |
| 1 | 967.94 | 5149.14 |  | 494.29 | 828.62 |
| 2 | 527.79 | 3413.73 |  | 344.56 | **809.02** |
| 3 | 415.68 | 3060.8 |  | 314.78 | 876.36 |
| 4 | 373.02 | 2898.12 |  | 299.13 | 960.85 |
| 5 | 343.53 | 2741.66 |  | 286.95 | 1717.02 |
| 6 | 319.55 | 2697.53 |  | 279.82 | 906.0 |
| 7 | 298.75 | 2779.89 |  | 272.17 | 1035.97 |
| 8 | 280.14 | 2865.53 |  | 263.82 | 1036.54 |
| 9 | 263.37 | 2729.85 |  | 259.78 | 831.5 |
| 10 | 248.22 | 2986.64 |  | 253.18 | 1417.8 |
| 11 | 234.43 | 2479.41 |  | 252.78 | 1582.18 |
| 12 | 221.68 | 2397.1 |  | 246.32 | 1551.16 |
| 13 | 209.56 | 2695.38 |  | 239.0 | 881.0 |
| 14 | 198.55 | 2565.65 |  | 237.06 | 1204.42 |
| 15 | 188.63 | 2354.02 |  | 235.9 | 1339.91 |
| 16 | 179.54 | 2433.03 |  | 232.66 | 1072.53 |
| 17 | 171.34 | 2335.96 |  | 233.08 | 2045.94 |
| 18 | 163.51 | 2525.17 |  | 232.08 | 1157.81 |
| 19 | 156.38 | 2600.38 |  | 231.01 | 1704.28 |
| 20 | 149.69 | 2189.19 |  | 231.73 | 4222.49 |
| 21 | 143.11 | 2137.6 |  | 230.99 | 3757.52 |
| 22 | 136.94 | 2096.51 |  | 229.7 | 2110.82 |
| 23 | 131.19 | 1997.07 |  | **228.25** | 2498.45 |
| 24 | 125.67 | 1997.78 |  | 229.65 | 2904.29 |
| 25 | 120.53 | 1954.38 |  | 229.96 | 2206.7 |
| 26 | 115.94 | 1863.62 |  | 230.31 | 3183.38 |
| 27 | 111.5 | 1911.42 |  | 231.0 | 2748.2 |
| 28 | 107.35 | 1749.79 |  | 233.52 | 2585.29 |
| 29 | 103.34 | 1804.05 |  | 234.73 | 2605.26 |
| 30 | 99.47 | 1669.6 |  | 236.99 | 3488.63 |

The Valid-PPL on globals and senses is (228, 809). Better than the architecture that shares a layer, that reaches (237, 809). So there is no reason to share the first layer between the GRUs.

## Structured prediction – select from K globals

### Design

Select k=5 globals, consider their senses, and choose among them.  
  
I still need to open up to the senses’ logits.

I get a distribution over the e.g. 20 senses from the k=5 globals.

I can multiply the senses’logits (obtained, for instance, with a standard GRU side-architecture) per what I get from here. Everything not selected would be x0.

Possibility: an alternative version may include the sense input.

For the sake of speed when retrieving the senses of the selected k globals:

we pass the graph\_area\_matrix as a parameter, to get the indices of the neighbouring nodes (since we are starting from globals, they will be either senses through the *sc* edges or other globals throught *synonyms/antonyms*.). We filter the neighbours to keep only the nodes whose index is in the senses’ range.

We obtain batch\_size\*sequence\_len\*k (e.g 4x8x5=160) tensors of variable size, containing the indices of the senses.

For every tensor containing the senses, we can apply X.index\_select(…) and get the sense embeddings.

We should get a probability distribution over these senses.

Idea #1: do not retrieve the embeddings. Assign 1 to the selected senses’indices and 0 to all others. Multiply per the softmax from the “other line”(the GRU). Then possibly scale up the probabilities that survived the filter so that they sum up to 1.

Problem: the nll\_loss works with log\_softmax, not with softmax.

Hypothesis a): assign 10^(-10) to the all the not-selected senses, instead of exactly 0 that would break the softmax and nll\_loss numerically. Then, rescale the softmax values that came from the relevant logits so that they sum up to 1.

**Hypothesis b)**: mask out the logits we don’t care for, and apply the softmax over the selected senses. Then, we will have to “make space” for all the 10^(-10) values, so we will have to subtract a small quantity δ from the selected values that we computed.

DataParallel(

(module): SelectK(

(maingru\_ls): ModuleList(

(0): GRU(300, 1150)

(1): GRU(1150, 1150)

(2): GRU(1150, 1150)

)

(gru\_senses): GRU(300, 1150, num\_layers=2)

(linear2global): Linear(in\_features=1150, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=1150, out\_features=25986, bias=True)

)

)

Number of parameters: 119.05M, where core=33.87M

### Mini-experiment – overfit on fragment of SemCor

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| For the globals:  **GRU** w/ 3 layers, 1150x3 (layer 1 shared).  Followed by **FF-NN** to logits | 1. The word embedding of the current global   (d=300) | batch\_size=40  TBPTT length=35 |
| For the senses:  **GRU** w/ 2 layers -> **FF-NN** to logits  - > applying the softmax of the senses of the k most likely globals | learning rate=**0.5\*10^(-4)** |

I do not manage to overfit. Sense loss stuck at 13.82, sense perplexity at 10^(6).

Incidentally, I set all the senses that do not belong to the k globals to 10^(-6).

When we are at the start, the first predicted globals are random, and do not provide a useful indication.

However, in later stages of the training they will. Here, the possibility of a correct sense prediction depends on the quality of the prediction of globals.

Let us use k=5 and wait until the globals’ part of the model starts overfitting…

We have an error due to the retrieval of globals and indices.

If my current most likely global is ‘cotton’, the senses I retrieve should not contain ‘factor.n.2’.

AD.get\_node\_data(self.grapharea\_matrix\_lil, 8977, self.N, features\_mask=(**True**,**False**,**False**))

-> tensor([ 8977, 56951, 83334, 32288, 8972, 8973, 8974, 8976, 8975, 35033,

34527])

We should remember that we are retrieving nodes 2 steps away, in a maximum number of 32.

Through the following edges, we can get the following node types:

sc -> senses; sc + def -> definitions; sc + ex -> examples; syn, ant -> globals; syn/ant + sc -> senses.

Since: EP.get\_globalword\_fromindex(32288-self.last\_idx\_senses) = ‘factor’ {global}

we are retrieving its senses.

However, the synonyms of ‘cotton’ , in the processed\_synonyms.h5 archive, are:

6386 cotton\_fiber

6387 cotton\_wool

6388 cotton\_plant

It is time to apply a sanity check to the graph, and the nodes we retrieve from the grapharea\_matrix.

## Graph check

Node: 8977 in the vocabulary of globals (cotton)

GA.get\_indices\_area\_toinclude(graph\_dataobject.edge\_index, graph\_dataobject.edge\_type, 8977, area\_size=32, max\_hops=1)

We get: ([8977(n: index of the global itself here), 56951, 83334, 32288],)

Remember that the X matrix has the sections:

**senses**: (0, 25986). **globals**: (25986, 47974). definitions: (47974, 73960). examples: (73960, 120602)

56951-47974=definition 8977. factor.v.03 consider as relevant when making a decision

83334-73960=example 9374. factor.v.03 You must factor in the recent developments

32288-25986=global 6302. factor 35

Why am I retrieving ‘factor’? What is the mistake here?

Let us add self.last\_idx\_senses to the node number I am retrieving…

This time,

GA.get\_indices\_area\_toinclude(graph\_dataobject.edge\_index, graph\_dataobject.edge\_type, 8977+25986{=34963}, 32, 1)

node\_indices\_ls, all\_edges\_retrieved\_ls =

([**34963**, 5673, 5672, 5671, 5674, *5670*], [57659, 57660, 57661, 57662, 57663])

**34963-**25986: global 8977 (‘cotton’)

Senses: 5670 to 5674: ﻿costume.v.02, cotton.n.01, cotton.n.02, cotton.n.03, cotton.n.04.

*Note*: it should be 5675, cotton.v.01 not 5670-costume.v.02. Must add 1. When I create the graph, maybe?

From the indices\_table.db, I get:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| word\_sense | vocab\_index | start\_defs | end\_defs | start\_examples | end\_examples |
| cotton.n.01 | 5670 | 5670 | 5671 | 5821 | 5821 |
| cotton.n.02 | 5671 | 5671 | 5672 | 5821 | 5821 |
| cotton.n.03 | 5672 | 5672 | 5673 | 5821 | 5821 |
| cotton.n.04 | 5673 | 5673 | 5674 | 5821 | 5821 |
| cotton.v.01 | 5674 | 5674 | 5675 | 5821 | 5822 |

When I execute get\_node\_edges, I get:

<class 'list'>: [(34963, 5673, 2), (34963, 5672, 2), (34963, 5671, 2), (34963, 5674, 2), (34963, 5670, 2)]

Let us review the most likely global and the senses that we retrieve for it in the SelectK architecture.

### Senses.SelectK retrieval and selection check

In Epoch 1, in the very first batch, at the start of the training process on mini-fragment:

(the initial most likely globals depend on the random initialization of the main GRU)

**INFO : sample: 0; k\_globals=['microorganisms']**

*INFO : sample: 1; k\_globals=['unscrupulous']*

…

**INFO : sample: 16; k\_globals=['shrink']**

*INFO : sample: 17; k\_globals=['unscrupulous']*

…

**# sample 0 is missing, since the plural form ‘microorganisms’ has no senses**.

*INFO : Sample: 1; selected\_senses=['unscrupulous.a.01']*

…

**INFO : Sample: 16; selected\_senses=['shrink.v.03', 'shrink.v.05', 'shrink.v.04', 'reduce.v.15', 'reduce.v.02', 'reduce.v.14', 'reduce.v.11', 'reduce.v.20', 'reduce.v.09', 'reduce.v.08', 'reduce.v.05', 'reduce.v.04', 'reduce.v.03', 'reduce.v.01', 'reduce.v.13', 'reduce.v.06', 'reduce.v.18']**

*INFO : Sample: 17; selected\_senses=['unscrupulous.a.01']*

From Epoch 2 to epoch… 161 and counting (globals train-PPL @161: 25.97, descending…) … epoch :

The most likely global is either ‘”’,‘the’ or ‘<unk>’ that have no senses.

I do not manage to get relevant globals to be the first one.

I use the verbose log of the predictions every 10 epochs, thus training faster (on GPUs) for 300 epochs.

Training, end of epoch 289. Global step n.1734. Time = 1180.27.

INFO : Perplexity: Globals perplexity=1.08 Sense perplexity=1383.87

Training, end of epoch 290. Global step n.1740. Time = 1182.03. The training losses are:

INFO : Perplexity: Globals perplexity=1.1 Sense perplexity=1979.8

|  |  |  |
| --- | --- | --- |
| Globals predicted | Senses predicted | Comment |
| Label: the next global is: said(from 1)  INFO : Label: the next sense is: state.v.01(from 22232)  INFO : The top- 5 predicted globals are:  INFO : Word: said ; probability = 92.4%  INFO : Word: further ; probability = 2.03%  INFO : Word: to ; probability = 1.34%  INFO : Word: thanks ; probability = 0.72%  INFO : Word: the ; probability = 0.54%  INFO : The top- 5 predicted senses are:  INFO : |  | Since “said” has no senses to select, we do not find any.  (We find only “say” in the vocabulary) |
| Label: the next global is: Friday(from 2)  INFO : Label: the next sense is: None(from -1)  INFO : The top- 5 predicted globals are:  INFO : Word: Friday ; probability = 82.86%  INFO : Word: it ; probability = 3.96%  INFO : Word: conducted ; probability = 2.76%  INFO : Word: , ; probability = 1.7%  INFO : Word: Only ; probability = 1.64%  INFO : |  |  |
| Label: the next global is: an(from 3)  INFO : Label: the next sense is: None(from -1)  INFO : The top- 5 predicted globals are:  INFO : Word: an ; probability = 91.28%  INFO : Word: the ; probability = 4.16%  INFO : Word: which ; probability = 1.6%  INFO : Word: . ; probability = 0.66%  INFO : Word: evidence ; probability = 0.36%  INFO : |  |  |
| Label: the next global is: investigation(from 4)  INFO : Label: the next sense is: probe.n.01(from 17882)  INFO : The top- 5 predicted globals are:  INFO : Word: investigation ; probability = 89.53%  INFO : Word: considering ; probability = 1.04%  INFO : Word: a ; probability = 0.97%  INFO : Word: won ; probability = 0.88%  INFO : Word: Only ; probability = 0.82% | INFO : The top- 5 predicted senses are:  INFO : Sense: probe.n.01 ; probability = 11.23%  INFO : Sense: investigate.v.01 ; probability = 10.91%  INFO : Sense: investigation.n.02 ; probability = 10.91%  INFO : Sense: investigate.v.02 ; probability = 10.86%  INFO : Sense: probe.n.03 ; probability = 10.82% |  |
| of Atlanta s |  |  |
| Label: the next global is: recent(from 8)  INFO : Label: the next sense is: late.s.03(from 13363)  INFO : The top- 5 predicted globals are:  INFO : Word: recent ; probability = 80.44%  INFO : Word: charge ; probability = 5.42%  INFO : Word: registration ; probability = 3.93%  INFO : Word: act ; probability = 1.94%  INFO : Word: handful ; probability = 1.29% | INFO : The top- 5 predicted senses are:  INFO : Sense: late.a.01 ; probability = 12.44%  INFO : Sense: late.r.03 ; probability = 12.3%  INFO : Sense: late.a.06 ; probability = 12.21%  INFO : Sense: late.s.03 ; probability = 12.18%  INFO : Sense: late.r.01 ; probability = 12.17% | we do not manage to focus on the correct sense… we are just picking the senses of the most likely global. |
|  |  |  |

Training epoch n.295: Perplexity: Globals perplexity=1.09 Sense perplexity=1712.68

Training epoch n.296: Perplexity: Globals perplexity=1.08 Sense perplexity=1776.37

Training epoch n.297: Perplexity: Globals perplexity=1.07 Sense perplexity=1786.04

Training epoch n.298: Perplexity: Globals perplexity=1.06 Sense perplexity=2161.16

Training epoch n.299: Perplexity: Globals perplexity=1.06 Sense perplexity=1197.16

Training epoch n.300: Perplexity: Globals perplexity=1.08 Sense perplexity=1712.68

### Modifications to SelectK

When the globals have already gone into overfit on the fragment, the sense perplexity is oscillating around 1100 and 2100, but it’s still very high.

We can try several modifications.

#1: Since “said” has no senses to select, we do not find any, even if there is a sense label – here, state.v.01

We can lemmatize ‘said’ (into ‘say’), and get the senses again.

#2: It may happen that even the lemmatized form has no senses or does not change. This happens with phrases, like ‘full of’.

We may decide to send a tensor of full-zeros, and ignore that sense label. This may or may not require modyfing the condition for ignoring the label, from a tensor with [0] to a full-zeros… it does not, because we do not explicitly ignore that label, we just exploit the fact that a uniform full-zero logsoftmax gives no gradient in the nll\_loss.

Working example of lemmatization:

sample\_k\_indices=[32671] -> [‘arches’] -> [‘arch’(6687+25986=32673)] -> we get sense\_neighbours\_t = tensor([ 1135, 1136, 1134, 1131, 1130, 1133, 1129, 1132, 25588])

The corresponding senses we extract are:

0 = {str} 'arch.s.03'

1 = {str} 'arch.v.01'

2 = {str} 'arch.s.02'

3 = {str} 'arch.n.03'

4 = {str} 'arch.n.02'

5 = {str} 'arch.s.01'

6 = {str} 'arch.n.01'

7 = {str} 'arch.n.04'

8 = {str} 'wicked.a.01'

We are picking, erroneously, one of arch’s syonyms:

arch.n.01 arch

arch.n.02 arch

arch.n.03 arch

arch.n.03 archway

arch.n.04 arch

arch.s.01 patronize

arch.s.01 patronise

arch.s.01 arch

arch.s.01 condescend

arch.s.02 arch

arch.s.03 pixilated

arch.s.03 puckish

arch.s.03 prankish

arch.s.03 arch

arch.s.03 wicked

arch.s.03 impish

arch.s.03 implike

arch.s.03 mischievous

arch.v.01 arch

arch.v.01 arc

arch.v.01 curve

Where does 25588 come from? It’s on the 2nd hop.

In order to avoid pulling the senses of synonyms (and antonyms), we need to use a 1-hop graph and graph area matrix for this particular use case.

Later on I will send the grapharea\_matrix with 1 hop specifically for this purpose while using the 2-hops version for the model input. For now, I can just use that one as the current grapharea matrix of the experiment

### Bugs & co

Current problems:

segfault only on GPU

git version

The segfault:

lib/python3.6/site-packages/tables/.libs/**libhdf5**-933c8d2d.so.103.0.0

In particular, we get:

Thread 38 "python3" received signal SIGSEGV, Segmentation fault.

[Switching to Thread 0x7ffd7b7ad700 (LWP 25678)]

0x00007fff53a8aab8 in H5C\_protect ()

from /home/andrealk3/venvs/torch15/lib/python3.6/site-packages/tables/.libs/libhdf5-933c8d2d.so.103.0.0

Problem:

we moved the model to DataParallel. HDF5 does not reliably allow for multithread access, or multiprocess for that matter.

New error: corrupted double-linked list.

Hypothesis 1:

“Some googling told me that apparently this one is due to a kernel debugging option:

CONFIG\_DEBUG\_STACK\_USAGE

and that the message is generally benign.”

Hypothesis 2:

“The two likely causes I can see are:

1) Writing into a block after it is freed.

2) A buffer overrun in a memory block into an adjacent freed block.”

On Cheetah we do not get this error. It is due to DataParallel-ism and replication.

Parallel reading on HDF5 strikes again. From the stacktrace:

k\_globals\_words = [EP.get\_globalword\_fromindex(global\_relative\_idx) for global\_relative\_idx in k\_globals\_relative\_indices]

File "/home/andrealk3/ALMA\_PhD/Code/TASK1\_Gbwe/GNN/ExplorePredictions.py", line 19, in get\_globalword\_fromindex

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

…’

tables.exceptions.HDF5ExtError: Problems reading records.

# Multi-sense LM. Part 1

## Gradient and loss

Whenever I have no sense labels, I should make it so that there is no gradient sent back in the model.

**Note**: maybe the train\_dataloader should use the lemmatizer to find senses of word like ‘said’? No, there is no need for that, either the training dataset has a sense label or it does not.

loss\_global = tfunc.nll\_loss(predictions\_globals, batch\_labels\_globals)

loss\_global = tensor(9.9997, grad\_fn=<NllLossBackward>)

<NllLossBackward object at 0x1a898ae190>

predictions\_globals.shape= torch.Size([32, 21988])

predictions\_globals[0] .shape= torch.Size([21988])

.grad\_fn= <SelectBackward object at 0x1a8b0f8e50>

.requires\_grad=True

Let us now examine:

batch\_labels\_senses=

tensor([ -1, 17882, -1, 13363, 17809, 17913, 8606, -1, -1, 13063,

9122, 22232, 23451, 8298, -1, 10898, 16289, 16237, 14922, 8131,

6745, 17606, 23521, 13952, 14322, 8131, 5081, -1, -1, -1,

23451, 13063])

loss\_sense = tensor(10.1642, grad\_fn=<NllLossBackward>)

predictions\_senses.shape= torch.Size([32, 25986])

predictions\_senses[0].shape= torch.Size([25986])

.grad\_fn= <SelectBackward object at 0x1a898ae750>

.requires\_grad=True

I may have to modify the functions:

compute\_model\_loss(model,batch\_input, batch\_labels, verbose=**False**)

and

compute\_sense\_loss(predictions\_senses, batch\_labels\_senses)

In torch.nn.functional.nll\_loss():

**ignore\_index** ([*int*](https://docs.python.org/3/library/functions.html#int)*, optional*) – Specifies a target value that is ignored and does not contribute to the input gradient. When size\_average is True, the loss is averaged over non-ignored targets.

Default: -100

Now loss\_sense is: tensor(10.1725, grad\_fn=<PermuteBackward>)

## GRUbase – debug on mini-experiments

Do not add any specific mechanism. Just use another GRU for the prediction on the senses’ logits&log\_softmax.

INFO : Model:

INFO : DataParallel(

(module): GRU\_base2(

(main\_gru): GRU(300, 1150, num\_layers=3)

(gru\_senses): GRU(300, 1150, num\_layers=2)

(linear2global): Linear(in\_features=1150, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=1150, out\_features=25986, bias=True)

)

)

p.s. I should also find a reason for using only 2 layers.

Maybe because not all the words have a sense label and thus I have fewer samples.

It may also be opportune to find a source for another set of hyperparameters of the architecture.

INFO : Parameters:

INFO : ('module.X', torch.Size([99963, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([6]), torch.float32, False)

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

('module.memory\_hn', torch.Size([3, 4, 1150]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([2, 4, 1150]), torch.float32, False)

('module.main\_gru.weight\_ih\_l0', torch.Size([3450, 300]), torch.float32, True)

('module.main\_gru.weight\_hh\_l0', torch.Size([3450, 1150]), torch.float32, True)

('module.main\_gru.bias\_ih\_l0', torch.Size([3450]), torch.float32, True)

('module.main\_gru.bias\_hh\_l0', torch.Size([3450]), torch.float32, True)

('module.main\_gru.weight\_ih\_l1', torch.Size([3450, 1150]), torch.float32, True)

('module.main\_gru.weight\_hh\_l1', torch.Size([3450, 1150]), torch.float32, True)

('module.main\_gru.bias\_ih\_l1', torch.Size([3450]), torch.float32, True)

('module.main\_gru.bias\_hh\_l1', torch.Size([3450]), torch.float32, True)

('module.main\_gru.weight\_ih\_l2', torch.Size([3450, 1150]), torch.float32, True)

('module.main\_gru.weight\_hh\_l2', torch.Size([3450, 1150]), torch.float32, True)

('module.main\_gru.bias\_ih\_l2', torch.Size([3450]), torch.float32, True)

('module.main\_gru.bias\_hh\_l2', torch.Size([3450]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l0', torch.Size([3450, 300]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l0', torch.Size([3450, 1150]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l0', torch.Size([3450]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l0', torch.Size([3450]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l1', torch.Size([3450, 1150]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l1', torch.Size([3450, 1150]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l1', torch.Size([3450]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l1', torch.Size([3450]), torch.float32, True)

('module.linear2global.weight', torch.Size([21988, 1150]), torch.float32, True)

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

('module.linear2senses.weight', torch.Size([25986, 1150]), torch.float32, True)

('module.linear2senses.bias', torch.Size([25986]), torch.float32, True)

INFO : Number of trainable parameters=119.05M

**Mini-experiments – Overfit on fragment of SemCor**

INFO : Hyperparameters: \_batchPerSeqlen32\_area32\_lr0.0001\_epochs300

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| model: GRUbase2 | 1) The word embedding of the current global (d=300) | batch\_size=4 |
| Globals’ prediction: **Main GRU** with 3 layers (1150 x3) | TBPTT length=8 |
| Followed by linear2Globals **FF-NN** | learning rate=10^(-4) |
| Senses’ prediction: **GRU** with 2 layers (1150x2),  Followed by linear2Senses **FF-NN** |  | grapharea=32, hops=2 |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |
| 1 | 21007.48 | 25634.85 |
| 2 | 16496.35 | 23663.83 |
| 3 | 3434.95 | 19598.76 |
| 4 | 390.72 | 6683.16 |
| 5 | 206.94 | 335.28 |
| 6 | 105.66 | 108.93 |
| 7 | 77.35 | 73.31 |
| 8 | 69.2 | 61.01 |
| 9 | 62.86 | 58.51 |
| 10 | 65.38 | 52.86 |
| 30 | 59.68 | 47.63 |
| 50 | 59.34 | 47.45 |
| 75 | 59.16 | 47.26 |
| 100 | 59.17 | 47.1 |
| 125 | 59.08 | 43.13 |
| 150 | 59.1 | 17.43 |
| 175 | 59.09 | 1.49 |
| 200 | 59.05 | 1.1 |
| 225 | 58.72 | 1.04 |
| 250 | 21.44 | 1.03 |
| 275 | 1.38 | 1.02 |
| 300 | 1.11 | 1.01 |

**Important question:** given the way they are built, the 2 GRUs should be effectively independent – apart from moving the embeddings of words&nodes.

Is the fact that both move the embeddings enough to justify a degree of mutual influence?

We review how the main GRU alone overfits on the globals of the fragment of SemCor.

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| model: GRUbase2 | 1) The word embedding of the current global (d=300) | batch\_size=4 |
| Globals’ prediction: **Main GRU** with 3 layers (1150 x3) | TBPTT length=8 |
| Followed by linear2Globals **FF-NN** | learning rate=10^(-4) |
|  |  | grapharea=32, hops=2 |

|  |  |
| --- | --- |
| *Epoch* | *Training perplexity on globals* |
| 1 | 21184.4 |
| 2 | 16478.93 |
| 3 | 3339.3 |
| 4 | 359.31 |
| 5 | 192.21 |
| 6 | 100.5 |
| 7 | 74.33 |
| 8 | 68.74 |
| 9 | 61.69 |
| 10 | 64.21 |
| 30 | 59.63 |
| 50 | 59.37 |
| 75 | 59.34 |
| 100 | 59.2 |
| 150 | 59.1 |
| 200 | 59.12 |
| 250 | 59.09 |
| 300 | 58.38 |

This time we had a slightly better random initialization.

It never stops “hovering” at ~59, where it’s predicting the most common tokens - ", the, <unk>, of, . .

Let us try this again with a higher learning rate.

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| model: GRUbase2 | 1) The word embedding of the current global (d=300) | batch\_size=4 |
| Globals’ prediction: **Main GRU** with 3 layers (1150 x3) | TBPTT length=8 |
| Followed by linear2Globals **FF-NN** | learning rate=**10^(-3)** |
|  |  | grapharea=32, hops=2 |

|  |  |
| --- | --- |
| *Epoch* | *Training perplexity on globals* |
| 1 | 4839.46 |
| 2 | 98.55 |
| 3 | 73.22 |
| 4 | 65.45 |
| 5 | 65.05 |
| 6 | 61.82 |
| 7 | 57.81 |
| 8 | 60.41 |
| 9 | 58.0 |
| 10 | 63.08 |
| 30 | 63.02 |
| 50 | 48.98 |
| 75 | 36.56 |
| 100 | 27.55 |
| 150 | 19.68 |
| 200 | 14.46 |
| 250 | 11.29 |
| 300 | 8.68 |

Why do I not manage to overfit on SemCor?

¨The correct prediction is not the first one, as it should be. Examples:

Label: the next global is: Georgia(from 69)

INFO : Label: the next sense is: None(from -1)

INFO : The top- 5 predicted globals are:

INFO : Word: relative ; probability = 8.64%

INFO : Word: Georgia ; probability = 6.33%

INFO : Word: produced ; probability = 6.27%

INFO : Word: of ; probability = 6.2%

INFO : Word: primary election ; probability = 5.32%

INFO :

-----

Label: the next global is: s(from 5197)

INFO : Label: the next sense is: None(from -1)

INFO : The top- 5 predicted globals are:

INFO : Word: . ; probability = 15.55%

INFO : Word: in ; probability = 8.96%

INFO : Word: , ; probability = 7.06%

INFO : Word: irregularities ; probability = 6.73%

INFO : Word: s ; probability = 6.41%

INFO :

-----

Label: the next global is: registration(from 70)

INFO : Label: the next sense is: registration.n.01(from 19059)

INFO : The top- 5 predicted globals are:

INFO : Word: been ; probability = 7.61%

INFO : Word: over-all ; probability = 7.54%

INFO : Word: recent ; probability = 6.98%

INFO : Word: primary election ; probability = 6.82%

INFO : Word: registration ; probability = 6.58%

INFO :

-----

Let us try to decrease the dimensions of the network: for the 2nd and 3rd layer, hidden\_units=600 (from 1150)

**Mini-experiment – smaller network**

DataParallel(

(module): GRU\_base2(

(main\_gru): GRU(300, 600, num\_layers=3)

(linear2global): Linear(in\_features=600, out\_features=21988, bias=True)

)

)

INFO : Parameters:

INFO : ('module.X', torch.Size([99963, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([6]), torch.float32, False)

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

('module.memory\_hn', torch.Size([3, 4, 600]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([2, 4, 600]), torch.float32, False)

('module.main\_gru.weight\_ih\_l0', torch.Size([1800, 300]), torch.float32, True)

('module.main\_gru.weight\_hh\_l0', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.bias\_ih\_l0', torch.Size([1800]), torch.float32, True)

('module.main\_gru.bias\_hh\_l0', torch.Size([1800]), torch.float32, True)

('module.main\_gru.weight\_ih\_l1', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.weight\_hh\_l1', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.bias\_ih\_l1', torch.Size([1800]), torch.float32, True)

('module.main\_gru.bias\_hh\_l1', torch.Size([1800]), torch.float32, True)

('module.main\_gru.weight\_ih\_l2', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.weight\_hh\_l2', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.bias\_ih\_l2', torch.Size([1800]), torch.float32, True)

('module.main\_gru.bias\_hh\_l2', torch.Size([1800]), torch.float32, True)

('module.linear2global.weight', torch.Size([21988, 600]), torch.float32, True)

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

INFO : Number of trainable parameters=49.15M

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| model: GRUbase2 | 1) The word embedding of the current global (d=300) | batch\_size=4 |
| Globals’ prediction: **Main GRU** with 3 layers (hd=600) | TBPTT length=8 |
| Followed by linear2Globals **FF-NN** | learning rate=**10^(-3)** |
|  |  | grapharea=32, hops=2 |

|  |  |
| --- | --- |
| *Epoch* | *Training perplexity on globals* |
| 1 | 8470.51 |
| 2 | 168.78 |
| 3 | 75.62 |
| 4 | 63.13 |
| 5 | 63.36 |
| 6 | 61.18 |
| 7 | 56.09 |
| 8 | 58.49 |
| 9 | 56.59 |
| 10 | 61.41 |
| 30 | 61.02 |
| 50 | 61.01 |
| 75 | 44.54 |
| 100 | 30.0 |
| 150 | 14.0 |
| 200 | 8.75 |
| 250 | 5.52 |
| 300 | 3.99 |

This time we have the correct prediction as among the most likely – not always the most likely one, but we can surmise it would become such with ~50 or 100 epochs.

Label: the next global is: It(from 78)

INFO : Label: the next sense is: None(from -1)

INFO : The top- 5 predicted globals are:

INFO : Word: The ; probability = 18.3%

INFO : Word: " ; probability = 13.33%

INFO : **Word: It ; probability = 10.97%**

INFO : Word: to ; probability = 10.87%

INFO : Word: that ; probability = 8.55%

INFO :

-----

Label: the next global is: recommended(from 79)

INFO : Label: the next sense is: recommend.v.01(from 18880)

INFO : The top- 5 predicted globals are:

INFO : Word: the ; probability = 18.07%

INFO : Word: investigate ; probability = 17.02%

INFO : **Word: recommended ; probability = 15.33%**

INFO : Word: term ; probability = 9.32%

INFO : Word: city ; probability = 8.91%

INFO :

-----

Label: the next global is: that(from 14)

INFO : Label: the next sense is: None(from -1)

INFO : The top- 5 predicted globals are:

INFO : Wo**rd: that ; probability = 37.19%**

INFO : Word: the ; probability = 11.67%

INFO : Word: evidence ; probability = 5.46%

INFO : Word: conducted ; probability = 5.08%

INFO : Word: reports ; probability = 4.72%

INFO :

-----

Label: the next global is: Fulton(from 44)

INFO : Label: the next sense is: location.n.01(from 13952)

INFO : The top- 5 predicted globals are:

INFO : **Word: Fulton ; probability = 22.13%**

INFO : Word: the ; probability = 13.96%

INFO : Word: investigate ; probability = 11.38%

INFO : Word: recommended ; probability = 10.7%

INFO : Word: city ; probability = 6.97%

INFO :

-----

Label: the next global is: legislators(from 80)

INFO : Label: the next sense is: legislator.n.01(from 13543)

INFO : The top- 5 predicted globals are:

INFO : **Word: legislators ; probability = 56.71%**

INFO : Word: <unk> ; probability = 18.73%

INFO : Word: Fulton ; probability = 10.65%

INFO : Word: , ; probability = 2.35%

INFO : Word: jury ; probability = 2.15%

INFO :

-----

Label: the next global is: act(from 81)

INFO : Label: the next sense is: act.v.01(from 260)

INFO : The top- 5 predicted globals are:

INFO : **Word: act ; probability = 37.36%**

INFO : Word: find ; probability = 13.51%

INFO : Word: conducted ; probability = 6.28%

INFO : Word: The ; probability = 4.85%

INFO : Word: of this ; probability = 4.45%

It is easier to overfit a model with fewer parameters, because it has less inertia: it takes fewer epochs for it to move from a “predict the most common token” local minimum to predicting the current token at each step.

Let us now try to overfit on globals + senses.

DataParallel(

(module): GRU\_base2(

(main\_gru): GRU(300, 600, num\_layers=3)

(gru\_senses): GRU(300, 600, num\_layers=3)

(linear2global): Linear(in\_features=600, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=600, out\_features=25986, bias=True)

)

)

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| model: GRUbase2 | 1) The word embedding of the current global (d=300) | batch\_size=4 |
| Globals’ prediction: **Main GRU** with 3 layers (hd=600) | TBPTT length=8 |
| Followed by linear2Globals **FF-NN** | learning rate=10^(-3) |
| Senses’ prediction: **GRU** with 3 layers (hd=600),  Followed by linear2Senses **FF-NN** |  | grapharea=32, hops=2 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Train.PPL on globals (without sense prediction)* | *Training perplexity on senses* |
| 1 | 8902.51 | *8470.51* | 19133.89 |
| 2 | 173.04 | *168.78* | 304.25 |
| 3 | **76.14** | ***75.62*** | 67.81 |
| 4 | 62.71 | *63.13* | 55.04 |
| 5 | 63.94 | *63.36* | 50.21 |
| 10 | 61.41 | *61.18* | 49.39 |
| 30 | 61.01 | *56.09* | 49.12 |
| 50 | **61.01** | ***58.49*** | 49.13 |
| 75 | **43.16** | ***56.59*** | 48.61 |
| 100 | 23.94 | *61.41* | 32.57 |
| 125 | 16.34 | *61.02* | 21.9 |
| 150 | 11.62 | *61.01* | 13.04 |
| 175 | 8.91 | *44.54* | 6.18 |
| 200 | 7.18 | *30.0* | 2.86 |
| 225 | 6.19 | *14.0* | 1.56 |
| 250 | 4.85 | *8.75* | 1.2 |
| 275 | 3.95 | *5.52* | 1.08 |
| 300 |  | *3.99* |  |

For reproducibility, from now on we will use torch.manual\_seed(seed)

### Overfitting base GRU on SemCor globals

DataParallel(

(module): GRU\_base2(

(main\_gru): GRU(300, 600, num\_layers=3)

(linear2global): Linear(in\_features=600, out\_features=21988, bias=True)

)

)

INFO : Parameters:

INFO : ('module.X', torch.Size([99963, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([6]), torch.float32, False)

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

('module.memory\_hn', torch.Size([3, 4, 600]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([2, 4, 600]), torch.float32, False)

('module.main\_gru.weight\_ih\_l0', torch.Size([1800, 300]), torch.float32, True)

('module.main\_gru.weight\_hh\_l0', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.bias\_ih\_l0', torch.Size([1800]), torch.float32, True)

('module.main\_gru.bias\_hh\_l0', torch.Size([1800]), torch.float32, True)

('module.main\_gru.weight\_ih\_l1', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.weight\_hh\_l1', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.bias\_ih\_l1', torch.Size([1800]), torch.float32, True)

('module.main\_gru.bias\_hh\_l1', torch.Size([1800]), torch.float32, True)

('module.main\_gru.weight\_ih\_l2', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.weight\_hh\_l2', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.bias\_ih\_l2', torch.Size([1800]), torch.float32, True)

('module.main\_gru.bias\_hh\_l2', torch.Size([1800]), torch.float32, True)

('module.linear2global.weight', torch.Size([21988, 600]), torch.float32, True)

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

INFO : Number of trainable parameters=49.15M

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| model: GRUbase2 | 1) The word embedding of the current global (d=300) | batch\_size=4 |
| Globals’ prediction: **Main GRU** with 3 layers (hd=600) | TBPTT length=8 |
| Followed by linear2Globals **FF-NN** | learning rate=**10^(-3)** |
|  |  | grapharea=32, hops=2 |

|  |  |
| --- | --- |
| *Epoch* | *Training perplexity on globals* |
| 1 | 8908.56 |
| 2 | 163.35 |
| 3 | 76.22 |
| 4 | 62.64 |
| 5 | 62.95 |
| 10 | 61.31 |
| 30 | 61.03 |
| 50 | 61.04 |
| 75 | 44.11 |
| 100 | 37.3 |
| 150 | 19.39 |
| 200 | 12.73 |
| 250 | 7.89 |
| 300 | 4.8 |
| 400 | 1.89 |

Double-checking that setting the PyTorch manual random seed brings the same results in the experiment: 8908.56, 163.35, 76.22,… yes, confirmed.

**Experiment II**:

We set the word embeddings matrix X as requires\_grad=False in the GRUbase2 model.

We try again to overfit on the globals-only task with the GRU.

Then, we will add the senses’ prediction part of the model, and since there are no other elements in common apart from X, we expect the globals’ prediction to be unchanged.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* |  | *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |
| 1 | 8914.2 |  | 1 | 9238.8 | 16922.44 |
| 2 | 163.32 |  | 2 | 175.25 | 277.27 |
| 3 | 76.21 |  | 3 | 75.28 | 66.57 |
| 10 | 61.31 |  | 10 | 61.3 | 49.36 |
| 50 | 61.04 |  | 50 | 59.56 | 49.14 |
| 100 | 47.01 |  | 100 | 31.19 | 29.46 |
| 200 | 18.6 |  | 200 | 12.87 | 9.4 |
| 300 | 7.16 |  | 300 | ... |  |
| 400 | 5.11 |  | 400 | … |  |
|  |  |  | 500 | 3.13 | 1.18 |

It seems that we do have different results.

This may be due to loss=loss\_global+loss\_sense.

It can still be correct. However, we should **verify** that after a label (global=any, sense=-1) we do not modify the gru\_senses.

### Preliminary experiment – GRUbase2 w/senses on SemCor

INFO : Hyperparameters: \_batchPerSeqlen32\_area32\_lr5e-05\_epochs100

INFO : Model:

INFO : DataParallel(

(module): GRU\_base2(

(main\_gru): GRU(300, 600, num\_layers=3)

(gru\_senses): GRU(300, 600, num\_layers=3)

(linear2global): Linear(in\_features=600, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=600, out\_features=25986, bias=True)

)

)

INFO : Parameters:

INFO : ('module.X', torch.Size([99963, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([6]), torch.float32, False)

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

('module.memory\_hn', torch.Size([3, 40, 600]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([3, 40, 600]), torch.float32, False)

('module.main\_gru.weight\_ih\_l0', torch.Size([1800, 300]), torch.float32, True)

('module.main\_gru.weight\_hh\_l0', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.bias\_ih\_l0', torch.Size([1800]), torch.float32, True)

('module.main\_gru.bias\_hh\_l0', torch.Size([1800]), torch.float32, True)

('module.main\_gru.weight\_ih\_l1', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.weight\_hh\_l1', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.bias\_ih\_l1', torch.Size([1800]), torch.float32, True)

('module.main\_gru.bias\_hh\_l1', torch.Size([1800]), torch.float32, True)

('module.main\_gru.weight\_ih\_l2', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.weight\_hh\_l2', torch.Size([1800, 600]), torch.float32, True)

('module.main\_gru.bias\_ih\_l2', torch.Size([1800]), torch.float32, True)

('module.main\_gru.bias\_hh\_l2', torch.Size([1800]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l0', torch.Size([1800, 300]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l0', torch.Size([1800, 600]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l0', torch.Size([1800]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l0', torch.Size([1800]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l1', torch.Size([1800, 600]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l1', torch.Size([1800, 600]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l1', torch.Size([1800]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l1', torch.Size([1800]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l2', torch.Size([1800, 600]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l2', torch.Size([1800, 600]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l2', torch.Size([1800]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l2', torch.Size([1800]), torch.float32, True)

('module.linear2global.weight', torch.Size([21988, 600]), torch.float32, True)

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

('module.linear2senses.weight', torch.Size([25986, 600]), torch.float32, True)

('module.linear2senses.bias', torch.Size([25986]), torch.float32, True)

INFO : Number of trainable parameters=70.73M

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| model: GRUbase2 | 1) The word embedding of the current global (d=300) | batch\_size=40 |
| Globals’ prediction: **Main GRU** with 3 layers (hd=600) | TBPTT length=35 |
| Followed by linear2Globals **FF-NN** | learning rate=5\*10^(-5) |
| Senses’ prediction: **GRU** with 3 layers (hd=600),  Followed by linear2Senses **FF-NN** |  | grapharea=32, hops=2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* | *Validation PPL on globals* | *Validation PPL on senses* |
| 1 | 1117.21 | 5521.64 | 513.59 | 897.91 |
| 2 | 738.79 | 3522.27 | 506.68 | 849.49 |
| 3 | 617.83 | 3365.4 | 395.15 | **826.48** |
| 4 | 472.55 | 3233.66 | 338.72 | 827.52 |
| 5 | 419.23 | 3032.93 | 313.37 | 835.21 |
| 6 |  |  |  |  |
| 7 |  |  |  |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |
| 12 |  |  |  |  |
| 13 |  |  |  |  |
| 14 |  |  |  |  |
| 15 |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

(note: for SemCor’s 650K words, compared to WT-2’s 2Mln words, it may be opportune to increase the learning rate from 5e-05 to 1e-04)

We do not manage to learn the senses, but the validation PPL has a much more reasonable trend – using the ignore\_index flag on the torch.nn.functional.nll\_loss fixed the computation of the senses’ loss.

The inability to learn can be explained by the ‘snapshot’ way the gru\_senses works, as explained elsewhere: it does not read the whole text word-by-word, but only sees the locations that have a sense label. This means that the GRU memory does not work the way it’s supposed to.

# Model Verification and Modification

## The loss when operating on senses

We launch GRUbase2 on a fragment of SemCor, and examine what happens when we compute the loss and after we

call loss.backward()

We set batch\_size=1, sequence\_length=1…

batch\_labels = tensor([[ 2, -1]])

Shape of the batch\_input tensor: torch.Size([1, 1, 1150])

The model, at the start of the forward:

(objective: the weights of the gru\_senses should *not* change)

self.memory\_hn.shape=(3,1,600)

At the start, it’s a tensor made entirely of zeros.

self.memory\_hn.nonzero()=tensor([],…)

The same for self.memory\_hn\_senses. Shape=(3,1,600), .nonzero()=tensor([])

We examine

self.main\_gru.weight\_ih\_l1.mean()=tensor(-4.3736e-05, grad\_fn=<MeanBackward0>)

self.main\_gru.weight\_hh\_l2.mean()=tensor(-8.4784e-06, grad\_fn=<MeanBackward0>)

, that we expect to change, and

self.gru\_senses.weight\_ih\_l1.mean()=tensor(-1.2207e-05, grad\_fn=<MeanBackward0>)

self.gru\_senses.weight\_hh\_l2.mean()=tensor(-2.3060e-05, grad\_fn=<MeanBackward0>)

that we expect to remain the same.

x\_indices\_g= tensor([ 1, 47975, 34783, 0, 32616])

edge\_index\_g= tensor([[1, 2, 2, 2, 2, 4, 2],

[0, 3, 0, 2, 2, 2, 4]])

edge\_type\_g= tensor([0, 2, 2, 3, 3, 3, 3])

x\_indices\_s= tensor([22232, 70206, 96175, 96176, 96177, 96178, 26158, 22228, 22229, 22230, 22227, 22231, 36926, 26852, 28113, 34096, 29324, 29292, 37538, 26756,

30394, 29242

edge\_index\_s= tensor([[ 1, 2, 3, 4, 5, 6, 6, 6, 6, 6, 6, 17, 6, 19, 6, 15, 6, 6,

12, 6, 6, 6, 6, 6, 6, 6, 18, 6, 16, 6, 20, 6, 17, 6, 6, 6,

6, 6, 6, 6, 13, 6, 14, 21, 6],

[ 0, 0, 0, 0, 0, 10, 7, 8, 9, 11, 0, 6, 17, 6, 19, 6, 15, 12,

6, 6, 6, 6, 6, 6, 6, 18, 6, 16, 6, 20, 6, 17, 6, 6, 6, 6,

6, 6, 6, 13, 6, 14, 6, 6, 21]])

edge\_type\_s= tensor([0, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3])

We compute the logits for both globals and senses, but only the predictions\_globals should encounter a meaningful label in the next step…

loss\_global= tensor(10.0134, grad\_fn=<NllLossBackward>)

Then, we have:

loss\_sense = tfunc.nll\_loss(predictions\_senses, batch\_labels\_senses, ignore\_index=-1)

with predictions\_senses.shape=(1,25986) and batch\_labels\_senses=tensor([-1])

loss\_sense= tensor(0., grad\_fn=<NllLossBackward>)

Then, in the training loop we execute:

batch\_sense\_tokens = (batch\_labels.t()[1][batch\_labels.t()[1]!=-1].shape[0])  
sum\_epoch\_loss\_sense = sum\_epoch\_loss\_sense + loss\_sense.item() \* batch\_sense\_tokens

epoch\_senselabeled\_tokens = epoch\_senselabeled\_tokens + batch\_sense\_tokens

And

loss = loss\_global + loss\_sense

loss.backward()

Now,

model.main\_gru.weight\_ih\_l1.mean()=tensor(-4.3115e-05, grad\_fn=<MeanBackward0>)

(from -4.3736e-05)

model.gru\_senses.weight\_ih\_l1.mean()=tensor(-1.2207e-05, grad\_fn=<MeanBackward0>)

(from -1.2207e-05) # unchanged, as expected.

### Considerations

The gru\_senses does not “see” anything related to the processing of the text word-after-word, as instead the main\_gru does.

It sees the current “snapshot” of input signals (whether that be the word embedding alone or input from the graph). gru\_senses adjusts itself and then waits “dormant” for a number of words, until the next sense label comes along.

Thus, gru\_senses has no understanding of the text / of the language. It proceeds as follows:

Label: the next global is: <unk>(from 21987) (from: Fulton\_County\_Grand\_Jury)

Label: the next sense is: group.n.01(from 10898)

INFO : batch\_labels=tensor([[21987, 10898]])

INFO : loss\_sense=tensor(10.1451, grad\_fn=<NllLossBackward>)

Label: the next global is: said(from 1)

INFO : Label: the next sense is: state.v.01(from 22232)

INFO : batch\_labels=tensor([[ 1, 22232]])

INFO : loss\_sense=tensor(10.1698, grad\_fn=<NllLossBackward>)

Label: the next global is: Friday(from 2)

INFO : Label: the next sense is: None(from -1)

INFO : batch\_labels=tensor([[ 2, -1]])

INFO : loss\_sense=tensor(0., grad\_fn=<NllLossBackward>)

Label: the next global is: an(from 3)

INFO : Label: the next sense is: None(from -1)

INFO : batch\_labels=tensor([[ 3, -1]])

INFO : loss\_sense=tensor(0., grad\_fn=<NllLossBackward>)

Label: the next global is: investigation(from 4)

INFO : Label: the next sense is: probe.n.01(from 17882)

INFO : batch\_labels=tensor([[ 4, 17882]])

INFO : loss\_sense=tensor(10.2071, grad\_fn=<NllLossBackward>)

## Graph retrieval

### Boundaries of graph node types

(n: node types’ boundaries are at: 25986, 47974, 73960)

**Senses: (0, 25986). Globals: (25986, 47974). Definitions: (47974, 73960). Examples: (73960, end)**

### Analysis…

Let us examine:

x\_indices\_g= tensor([ 1, 47975, 34783, 0, 32616])

# Likely mistake: the 1st global has not been made a relative index to get the correct row from the X matrix.

x\_indices\_s= tensor([22232, 70206, 96175, 96176, 96177, 96178, 26158, 22228, 22229, 22230, 22227, 22231, 36926, 26852, 28113, 34096, 29324, 29292, 37538, 26756,

30394, 29242])

When I am starting to read the SemCor fragment, how do I retrieve nodes?

The DataLoader calls:

get\_forwardinput\_forelement(global\_idx, sense\_idx, grapharea\_matrix, area\_size)

with:

global\_idx=0, sense\_idx=-1, grapharea\_matrix=…, area\_size=32

**if** (sense\_idx == -1): we use all torch.zeros(…).

Let us see the globals:

area\_x\_indices\_global, edge\_index\_global, edge\_type\_global = AD.get\_node\_data(grapharea\_matrix, global\_idx, area\_size)

Inside that function:

*# Accessing sparse matrix. Everything was shifted +1, so now: we ignore 0 ; we shift -1; we get the data*nodes\_ls =list(map(**lambda** value: value - 1, filter(**lambda** num: num != 0, grapharea\_matrix[i, 0:k].todense().tolist()[0])))

and nodes\_ls=<class 'list'>: [0.0, 47974.0, 34783.0, 1.0, 32616.0].

Which nodes did we get from the matrix?

0: 0th sense

47974: 47974-47974=0th definition

34783: 34783-25986= 8797th global

1: 1st sense

32616: 32616-25986=6630th global

Let us review what we get from the graph, instead of the graph matrix. Starting node: 0.

GA.get\_indices\_area\_toinclude(graph\_dataobject.edge\_index, graph\_dataobject.edge\_type, node\_index=0, area\_size=32, max\_hops=1):

nodes\_queue\_at\_current\_level=[0]

node\_edges=<class 'list'>: [(47974, 0, 0), (34783, 0, 2)]

nodes\_queue\_at\_next\_hop=we add 47974 and 34783

In the end, node\_indices\_ls, all\_edges\_retrieved\_ls=([0, 47974, 34783], [0, 51989])

Issue #1: if we are extracting the 1st global, we should not operate on index 0, but on the index (start\_of\_globals+0).

Examining: NI.get\_tokens\_tpls(…)

When we are in the dataloader’s \_\_getitem\_\_(self, index):

current\_token\_tpl=<class 'tuple'>: (0, -1)

next\_token\_tpl=<class 'tuple'>: (21987, 10898)

So the 2nd is correct, but the 1st is not.

What happens when token\_dict={'surface\_form': 'said', 'lemma': 'say', 'pos': 'VBD', 'wn16\_key': 'say%2:32:00::', 'wn30\_key': 'say%2:32:00::'}?

wordnet\_sense=state.v.01 ; From querying the SQL indices\_table.db, we get: sense\_index\_queryresult=<class 'tuple'>: (22232,)

For the global,

global\_absolute\_index = Utils.select\_from\_hdf5(globals\_vocabulary\_h5, **'vocabulary'**, [**'word'**], [word]).index[0]

I took out the addition of the last\_sense\_index…

global\_index = global\_absolute\_index *# + last\_idx\_senses; do not add this to globals, or we go beyond the n\_classes*

For the label purpose, the consideration is correct. However, it is necessary to add that term for the purpose of retrieving graph data and rows of X.

Therefore, in the current version, global\_index=1.

next\_token\_tpl=<class 'tuple'>: (1, 22232)

Whereas the current\_token\_tpl was correct: <class 'tuple'>: (21987, 10898)

Then we go on:

{'surface\_form': 'Friday', 'lemma': 'friday', 'pos': 'NNP', 'wn16\_key': 'friday%1:28:00::', 'wn30\_key': 'friday%1:28:00::'}

The sense is friday.n.01… The global\_absolute\_index is 2.

So we have:

current\_token\_tpl=<class 'tuple'>: (1, 22232) , next\_token\_tpl=<class 'tuple'>: (2, -1)

the current tuple, in the meantime, leads us to get\_node\_data with i=1.

nodes=tensor([ 1(sense!), 47975(def), 34783(global), 0(sense), 32616(global)])

edgeindex.T=tensor([[1, 0],

[2, 3], [2, 0],

[2, 2], [2, 2], [4, 2], [2, 4]])

edgetype=tensor([0, 2, 2, 3, 3, 3, 3]) (edge types are: 0=defs, 1=examples, 2=SenseChildren, 3=synonyms, 4=antonyms). (n: considering that here we may be from the point of view of the sense=1, instead of the global 25987)

What are the nodes we got?

“1”: It should have been the global: said, but this is sense n.1=ab.n.04

47975: 47975-47974= definition n. 1= from ab.n.04, the blood group whose red cells carry both the A and B antigens

34783: 34783-25986=8797th global= 8797 word=ab frequency=3

0: Sense n.0 = ab.n.02

32616: 32616-25986= 6630th global= 6630 word=abdominal frequency=3

While the token\_tuple that is also used for the labels and can use the “absolute” global index, the retrieval of data from the graph must use the **relative global index** (i.e. + last\_sense\_idx).

If I retrieve what I should, i.e. the node index 25986+1, from the graph:

nodes, edges=([25987], [77976])

The only node is the global n.1 in the vocabulary of globals:

“said”.

Since it’s not “say”, it does not have any connection in the graph. We should apply lemmatization before attempting node retrieval.

“say” is the global n. 2127.

If we execute get\_indices\_area\_toinclude(edge\_index, edge\_type, node\_index*=25986+2127=28113*, area\_size, max\_hops):

Using the node itself (during the 1st iteration, at hop=1)

node\_edges=<class 'list'>: [

(28113, 20173, 2),

(28113, 20178, 2),

(28113, 20177, 2),

(28113, 20175, 2),

(28113, 20176, 2),

(28113, 20174, 2),

(26631, 28113, 3), (28113, 26631, 3),

(28113, 26158, 3), (26158, 28113, 3),

(28113, 29596, 3), (29596, 28113, 3)

(28113, 43985, 3), (43985, 28113, 3),

(28113, 26599, 3), (26599, 28113, 3),

(28113, 28113, 3)]

Senses: say.n.01, say.v.07, say.v.08, say.v.09, say.v.10, say.v.11

***~~Note~~***~~: I am still with a displacement of 1. This displacement could be due to <unk>… I will have to check sense 20173 and what I retrieve from it…~~

Global synonyms – to find them, subtract 25968:

[645:order, 172:state, 3610:suppose, 17999:aforesaid, 613:read]

Moreover, now that we are able to make the correct request to the graph, I decide to operate with 1 hop of distance as the starting hyperparameter.

### Senses’ +1 displacement

Lastly, we review the senses’ retrieval:

when we have global n.1 (‘said’), we will operate with:

area\_x\_indices\_sense: tensor([

22232: sense ~~state.n.06 (or so we get from consulting indices\_table.sql)~~ state v.01,

70206: 22232nd definition (state.v.01 – express in words),

96175: 22215th example (state.v.01 – He said that he wanted to marry her),

96176: 22216th example (state.v.01 – tell me what is bothering you),

96177: “” state your opinion, 96178: “” state your name,

26158: 172nd global “state”,

22228: sense – from consulting: state.n.02,

22229: sense – from consulting: state.n.03,

22230: sense – from consulting: state.n.04,

22227: sense – from consulting: state.n.01,

22231: sense – from consulting: state.n.06,

36926: 10940th global “province”,

26852: 866th global “tell”, # I still have 2 hops in the graph area in this particular experiment, so it’s: sense->global->synonym.

28113: 2127th global “say”,

34096: 8110th global “express”,

etc. 29324, 29292, 37538, 26756, 30394, 29242])

This may be just due to the row-counting in indices\_table.sql.

The rows that we visualize in the table go from 1 to 25986.

When my sense-node index is 0, I actually refer to row 1 in the table.

(I adjust the meaning of the indices above)

## Adding lemmatization for the global node

If we encounter a word like ‘said’, it would be informative to include the graph-input-signal from the global node ‘say’.

# Structured prediction: SelectK – version 1

## Model

The prediction of senses should build upon the standard-LM prediction of globals.

Select the senses of the *k* most likely globals. In particular, we select their logits from the senses’ GRU.

We apply the softmax over them alone. Then, we remove a small amount of probability mass to give ε=10^(-6)

INFO : Model:

INFO : DataParallel( (module): SelectK(

(main\_gru): GRU(300, 1150, num\_layers=3)

(gru\_senses): GRU(300, 575, num\_layers=3)

(linear2global): Linear(in\_features=1150, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=575, out\_features=25986, bias=True)

) )

INFO : Parameters:

INFO : ('module.X', torch.Size([99963, 300]), torch.float32, True)

('module.select\_first\_indices', torch.Size([6]), torch.float32, False)

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

('module.memory\_hn', torch.Size([3, 4, 1150]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([3, 4, 575]), torch.float32, False)

('module.main\_gru.weight\_ih\_l0', torch.Size([3450, 300]), torch.float32, True)

('module.main\_gru.weight\_hh\_l0', torch.Size([3450, 1150]), torch.float32, True)

('module.main\_gru.bias\_ih\_l0', torch.Size([3450]), torch.float32, True)

('module.main\_gru.bias\_hh\_l0', torch.Size([3450]), torch.float32, True)

('module.main\_gru.weight\_ih\_l1', torch.Size([3450, 1150]), torch.float32, True)

('module.main\_gru.weight\_hh\_l1', torch.Size([3450, 1150]), torch.float32, True)

('module.main\_gru.bias\_ih\_l1', torch.Size([3450]), torch.float32, True)

('module.main\_gru.bias\_hh\_l1', torch.Size([3450]), torch.float32, True)

('module.main\_gru.weight\_ih\_l2', torch.Size([3450, 1150]), torch.float32, True)

('module.main\_gru.weight\_hh\_l2', torch.Size([3450, 1150]), torch.float32, True)

('module.main\_gru.bias\_ih\_l2', torch.Size([3450]), torch.float32, True)

('module.main\_gru.bias\_hh\_l2', torch.Size([3450]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l0', torch.Size([1725, 300]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l0', torch.Size([1725, 575]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l0', torch.Size([1725]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l0', torch.Size([1725]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l1', torch.Size([1725, 575]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l1', torch.Size([1725, 575]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l1', torch.Size([1725]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l1', torch.Size([1725]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l2', torch.Size([1725, 575]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l2', torch.Size([1725, 575]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l2', torch.Size([1725]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l2', torch.Size([1725]), torch.float32, True)

('module.linear2global.weight', torch.Size([21988, 1150]), torch.float32, True)

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

('module.linear2senses.weight', torch.Size([25986, 575]), torch.float32, True)

('module.linear2senses.bias', torch.Size([25986]), torch.float32, True)

INFO : Number of trainable parameters=96.64M

## Mini-Experiment 1 – Overfit on fragment of SemCor

### Losses

batchPerSeqlen32\_area32\_lr0.0005\_epochs400

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| model: SelectK | 1) The word embedding of the current global (d=300) | batch\_size=4 |
| Globals’ prediction: **Main GRU** with 3 layers (d=1150) | TBPTT length=8 |
| Followed by linear2Globals **FF-NN** | learning rate=5\* 10^(-4) |
| Senses’ prediction:  **Senses GRU** (L=3, d=575), apply softmax on the selected logits of the senses of the most likely ***k*** globals |  | grapharea=32, hops=2 |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |
| 1 | 9026.6 | 1000000.32 |
| 2 | 178.7 | 1000000.46 |
| 3 | 81.98 | 854154.08 |
| 4 | 61.46 | 1000000.32 |
| 5 | 64.17 | 1000000.32 |
| 10 | 61.24 | 1000000.32 |
| 30 | 60.88 | 1000000.32 |
| 40 | 58.78 | 1000000.32 |
| 50 | 45.55 | 1000000.32 |
| 75 | 29.57 | 1000000.32 |
| 100 | 17.73 | 1000000.32 |
| 125 | 8.1 | 530171.45 |
| 150 | 3.8 | 14308.24 |
| 200 | 1.66 | 1378.08 |
| 250 | 1.23 | 616.07 |
| 300 | 1.1 | 616.07 |

### Mini-experiment 1 – Analysis of the predictions

|  |  |  |
| --- | --- | --- |
| Label: the next global is: jury(from 19)  INFO : Label: the next sense is: jury.n.01(from 13063) | INFO : The top- 5 predicted globals are:  INFO : Word: jury ; p=91.75%  INFO : Word: September ; p=5.21%  INFO : Word: <unk> ; p=1.17%  INFO : Word: term ; p=0.78%  INFO : Word: often ; p=0.25% | INFO : The top- 5 predicted senses are:  INFO : Sense: jury.n.02 ; p = 49.45%  INFO : Sense: jury.n.01 ; p = 47.95% |
| Label: the next global is: find(from 67)  INFO : Label: the next sense is: **rule.v.04**(from 19931) | INFO : The top- 5 predicted globals are:  INFO : Word: find ; p=93.61%  INFO : Word: interest ; p=1.1%  INFO : Word: relative ; p=0.96%  INFO : Word: inadequate ; p=0.58%  INFO : Word: took place ; p=0.44% | INFO : The top- 5 predicted senses are:  INFO : Sense: find.v.03 ; p = 16.9%  INFO : Sense: find.v.05 ; p = 16.86%  INFO : Sense: find.v.13 ; p = 16.21%  INFO : Sense: find.v.10 ; p = 16.06%  INFO : Sense: find.v.15 ; p = 16.05% |
| Label: the next global is: registration(from 70)  INFO : Label: the next sense is: registration.n.01(from 19059) | INFO : The top- 5 predicted globals are:  INFO : Word: registration ; p=93.85%  INFO : Word: recent ; p=1.64%  INFO : Word: of this ; p=1.47%  INFO : Word: had ; p=0.4%  INFO : Word: by ; p=0.34% | INFO : The top- 5 predicted senses are:  INFO : Sense: registration.n.04 ; p = 25.92%  INFO : Sense: registration.n.01 ; p = 24.63%  INFO : Sense: registration.n.02 ; p = 24.03%  INFO : Sense: registration.n.03 ; p = 22.83% |

Etc.

2 observations follow:

* As we see in *registration*, the model does not choose the correct sense among those of the predicted global.
* As wee see in *find+rule.v.04*, sometimes the sense of a word will not be one of those directly associated with the word (here the text is “the court did find”…)

If we were able to choose the correct sense among those of the predicted global…

I examine the computation graph, going backwards from predictions\_senses, to see where it breaks…

predictions\_senses has \_grad=None, \_grad\_fn=None

senses\_softmax : the same

logits\_sense.\_grad\_fn= <AddmmBackward object at 0x1a866d3ad0>

Moreover:

logits\_global = self.linear2global(main\_gru\_out)

logits\_globa.\_grad\_fn = <AddmmBackward object at 0x1a8ae6ba50>

k\_globals\_indices = logits\_global.sort(descending=**True**).indices[:, 0:self.k]

k\_globals\_indices.\_grad\_fn=None

I try to change how I pick the logits\_senses, let us try to use index\_select to maintain the gradient…

## Mini-Experiment 2 – Overfit on fragment of SemCor

### Losses

batchPerSeqlen32\_area32\_lr0.0005\_epochs400

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (d=1150) | 1) The word embedding of the current global (d=300) | batch\_size=4  TBPTT length=8 |
| linear2Globals **FF-NN** | grapharea=32, hops=2 |
| **Senses GRU** (L=3, d=575) into linear2Senses **FF-NN**, apply softmax on the logits of the senses of the most likely ***k*** globals | learning rate=5\* 10^(-4) |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |
| 1 | 9817.6 | 1000000.32 |
| 2 | 178.86 | 1000000.46 |
| 3 | 79.23 | 1000000.32 |
| 4 | 63.24 | 1000000.32 |
| 5 | 63.86 | 1000000.32 |
| 10 | 61.42 | 1000000.32 |
| 35 | 60.81 | 1000000.32 |
| 50 | 42.0 | 1000000.32 |
| 75 | 25.35 | 1000000.32 |
| 100 | 15.01 | 856529.48 |
| 125 | 7.66 | 155110.68 |
| 150 | 3.62 | 5586.52 |
| 200 | 1.51 | 733.13 |
| 225 | 1.27 | 617.06 |
| 250 | 1.16 | 617.06 |
| 300 | 1.08 | 617.06 |

### Mini-experiment 2 – Analysis of the predictions

|  |  |  |
| --- | --- | --- |
| Label: the next global is: jury(from 19)  INFO : Label: the next sense is: jury.n.01(from 13063) | INFO : The top- 5 predicted globals are:  INFO : Word: jury ; p=96.72%  INFO : Word: September ; p=1.73%  INFO : Word: <unk> ; p=0.29%  INFO : Word: end ; p=0.26%  INFO : Word: manner ; p=0.22% | INFO : The top- 5 predicted senses are:  INFO : Sense: jury.n.02 ; p = 48.72%  INFO : Sense: jury.n.01 ; p = 48.68% |
| Label: the next global is: find(from 67)  INFO : Label: the next sense is: rule.v.04(from 19931) | INFO : The top- 5 predicted globals are:  INFO : Word: find ; p=95.2%  INFO : Word: , ; p=1.75%  INFO : Word: Georgia ; p=0.58%  INFO : Word: ambiguous ; p=0.43%  INFO : Word: produced ; p=0.29% | INFO : The top- 5 predicted senses are:  INFO : Sense: find.v.13 ; p = 16.69%  INFO : Sense: find.v.05 ; p = 16.6%  INFO : Sense: find.v.01 ; p = 16.48%  INFO : Sense: find.v.15 ; p = 16.19%  INFO : Sense: find.v.03 ; p = 15.74% |
| Label: the next global is: registration(from 70)  INFO : Label: the next sense is: registration.n.01(from 19059) | INFO : The top- 5 predicted globals are:  INFO : Word: registration ; p=90.77%  INFO : Word: recent ; p=5.32%  INFO : Word: recommended ; p=0.83%  INFO : Word: in ; p=0.7%  INFO : Word: a ; p=0.57% | INFO : The top- 5 predicted senses are:  INFO : Sense: registration.n.02 ; p = 25.15%  INFO : Sense: registration.n.03 ; p = 24.6%  INFO : Sense: registration.n.01 ; p = 24.55%  INFO : Sense: registration.n.04 ; p = 23.1% |
| … |  |  |
| Label: the next global is: act(from 81)  INFO : Label: the next sense is: act.v.01(from 260) | INFO : The top- 5 predicted globals are:  INFO : Word: act ; p=95.13%  INFO : Word: relative ; p=1.86%  INFO : Word: October ; p=0.69%  INFO : Word: <unk> ; p=0.6%  INFO : Word: , ; p=0.28% | INFO : The top- 5 predicted senses are:  INFO : Sense: act.v.04 ; p = 7.72%  INFO : Sense: act.v.03 ; p = 7.6%  INFO : Sense: act.n.03 ; p = 7.58%  INFO : Sense: act.n.05 ; p = 7.56%  INFO : Sense: act.v.02 ; p = 7.55% |

Both problems spotted earlier remain:

**Some senses do not coincide with the global word.**

Examples:

|  |  |
| --- | --- |
| often | frequently.r.01 |
| outmoded | antique.s.02 |
| are | be.v.01 – we could find it if we lemmatized, but we retrieve INFO : Sense: are.n.01 ; p = 97.4%  This presents a case for lemmatizing always |
| find | rule.v.04 |

This could be solved by reading the training split of the sense-labeled corpus at graph creation, and connecting the global nodes to the sense nodes appropriately.

**We are unable to choose between the selected senses.**

We did not manage to reconnect the gradient to predictions\_senses yet.

logits\_selected\_senses.\_grad\_fn= <IndexSelectBackward object at 0x1a81070dd0>

…

softmax\_selected\_senses.\_grad\_fn=<SubBackward0 object at 0x1ae7219450>

I had to modify:

sample\_senses\_softmax.masked\_scatter\_(mask=i\_senseneighbours\_mask[s], source=softmax\_selected\_senses~~.data.clone())~~

**Temp – issue:**

RuntimeError: one of the variables needed for gradient computation has been modified by an inplace operation: [CUDABoolType [25986]] is at version 21; expected version 20 instead.

Warning: Error detected in torch::autograd::CopySlices. Traceback of forward call that caused the error:

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

File "/home/andrealk3/ALMA\_PhD/Code/TASK1\_Gbwe/GNN/Models/Senses.py", line 234, in forward

senses\_softmax[s].masked\_scatter\_(mask=i\_senseneighbours\_mask[s], source=softmax\_selected\_senses)

(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/GNN/Training.py", line 195, in training\_loop

loss.backward()

## Mini-experiment 3

### Losses

Hyperparameters: \_batchPerSeqlen32\_area32\_lr0.0005\_epochs300

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (d=1150) | 1) The word embedding of the current global (d=300) | batch\_size=4  TBPTT length=8 |
| linear2Globals **FF-NN** | grapharea=32, hops=2 |
| **Senses GRU** (L=3, d=575) into linear2Senses **FF-NN**, apply softmax on the logits of the senses of the most likely ***k*** globals | learning rate=5\* 10^(-4) |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |
| 1 | 9741.0 | 1000000.32 |
| 2 | 175.6 | 1000000.46 |
| 3 | 79.5 | 1000000.32 |
| 4 | 62.28 | 1000000.32 |
| 5 | 63.95 | 1000000.32 |
| 10 | 61.28 | 1000000.32 |
| 35 | 60.88 | 1000000.32 |
| 50 | 52.69 | 1000000.32 |
| 75 | 31.13 | 1000000.32 |
| 100 | 18.38 | 421930.79 |
| 125 | 10.65 | 355068.55 |
| 150 | 7.32 | 251424.08 |
| 200 | 3.26 | 3997.55 |
| 225 | 2.31 | 424.04 |
| 250 | 1.81 | 252.63 |
| 300 | 1.39 | 252.63 |

### Mini-experiment 3 – Analysis of the predictions

|  |  |  |
| --- | --- | --- |
| Label: the next global is: jury(from 19)  INFO : Label: the next sense is: jury.n.01(from 13063) | INFO : Word: jury ; p=70.15%  INFO : Word: September ; p=21.89%  INFO : Word: investigation ; p=1.17%  INFO : Word: <unk> ; p=1.11%  INFO : Word: any ; p=0.71% | INFO : Sense: jury.n.01 ; p = 98.7%  *This time we manage to pick jury.n.01, instead of being undecided between jury.n.01 and jury.n.02* |
| Label: the next global is: find(from 67)  INFO : Label: the next sense is: rule.v.04(from 19931) | INFO : Word: find ; p=80.47%  INFO : Word: irregularities ; p=4.93%  INFO : Word: or ; p=3.77%  INFO : Word: that ; p=3.33%  INFO : Word: said ; p=2.67% | INFO : Sense: find.v.10 ; p = 21.37%  INFO : Sense: find.v.05 ; p = 20.19%  INFO : Sense: find.v.15 ; p = 16.35%  INFO : Sense: find.v.01 ; p = 14.71%  INFO : Sense: find.v.03 ; p = 12.83%  *We select the senses of the global ‘find’, but we* *are unable to find rule.v.04. That issue must still be addressed* |
| Label: the next global is: registration(from 70)  INFO : Label: the next sense is: registration.n.01(from 19059) | INFO : Word: registration ; p=66.08%  INFO : Word: received ; p=6.73%  INFO : Word: to ; p=6.45%  INFO : Word: , ; p=3.85%  INFO : Word: The ; p=3.11% | INFO : Sense: registration.n.01 ; p = 99.35%  *Instead of being split at 25% x4 between n.01,…,n.04* |
| … |  |  |
| Label: the next global is: legislators(from 80)  INFO : Label: the next sense is: legislator.n.01(from 13543) | INFO : Word: legislators ; p=87.93%  INFO : Word: been ; p=1.96%  INFO : Word: of ; p=1.93%  INFO : Word: and ; p=1.78%  INFO : Word: <unk> ; p=1.55% | INFO : Sense: legislator.n.01 ; p = 97.4% |
| Label: the next global is: act(from 81)  INFO : Label: the next sense is: act.v.01(from 260) | INFO : Word: act ; p=73.69%  INFO : Word: deserves ; p=4.82%  INFO : Word: that ; p=3.08%  INFO : Word: no ; p=2.64%  INFO : Word: . ; p=2.36% | INFO : The top- 5 predicted senses are:  *We have lost the senses, the previous version managed to pick them giving them a p=~7.5%* |

## Adding global-to-sense edges from the SLC

We have seen that sometimes the correct sense attributed to a word is not among those immediately associated with that word in the dictionary.

e.g. find -> rule.v.04

When we create the graph, it is opportune to read the training split of the senses’ corpus and add the senseChildren edges that derive from all occurrences of   
global\_w1 -> sense\_from\_w2

\* done when the global word is *not* in the sense denomination, e.g. say->state.v.04

Checking that the new senseChildren edges that we added are correct:

This time the adjacent nodes for ‘say’ (relative global 28113) are:

(remember: Senses: (0, 25986). Globals: (25986, 47974). Definitions: (47974, 73960). Examples: (73960, end) )

[28113, global 2127, ‘say’

22232, sense ‘state.v.01’

20173, sense ‘say.n.01’

20178, say.v.11

20175, say.v.08

18711, read.v.02

20177, say.v.10

20176, say.v.09

524, ﻿aforesaid.s.01

20174, ﻿say.v.07

16073, order.v.02

22929, ﻿suppose.v.01

26631, global 645, ‘order’

43985, global 17999 ’aforesaid’

26158, global 172 ‘state’

29596, global 3610 ‘suppose’

26599] global 3613 ‘read

The connections are all pertinent. We have the old senses of ‘say’ plus the external senseChildren connections that we have read from the training split of SemCor.

The larger the SenseLabeledCorpus, the more of these connections we will include.

### Analysis of predictions – why do we have no senses for ‘act’, SensePerplexity=nan, etc.

At Epoch 200, using the GPUs:

In the model’s forward(), k\_globals\_words=

<unk>, many of, The,

jury:

softmax\_selected\_senses = tensor([9.9997e-01, 3.2930e-05], device='cuda:0', grad\_fn=<SoftmaxBackward>)

said:

softmax\_selected\_senses = tensor([5.1017e-06, 7.6327e-06, 1.3423e-05, 5.6478e-06, 5.9111e-06, 6.8113e-06,

8.5246e-06, 8.5297e-06, 1.0177e-05, 9.9992e-01, 9.2005e-06],

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

ok, found the error.

When the probability of not-correct senses is < ε and we subtract ε, we eventually get something like

softmax\_selected\_senses = tensor([-0.0024, -0.0024, -0.0023, -0.0024, -0.0024, -0.0024, -0.0024, -0.0024,

-0.0024, 0.9976, -0.0024], device='cuda:0', grad\_fn=<SubBackward0>)

Which breaks the computation.

Let us change the epsilon (the small probability mass given to all the non-selected senses) from 10^(-6) to 10^(-8).

Moreover, as a safety measure, I could either:

1. set the epsilon to be the minimum between [10^(-8)] and [the smallest probability accorded to one of the softmax\_selected\_senses + 10^(-9)]
2. do not modify the selected\_senses\_softmax values < 10^(-8)

## Mini-experiment 4

Note: to have faster experiments while I debug the numerical computation on the senses\_softmax, I now use an even smaller fragment of SemCor (**MiniDataset 2**, with 63 tokens instead of 180).

### Losses

INFO : Hyperparameters: \_batchPerSeqlen32\_area32\_lr0.0001\_epochs300

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (d=1150) | 1) The word embedding of the current global (d=300) | batch\_size=4  TBPTT length=8 |
| linear2Globals **FF-NN** | grapharea=32, hops=2 |
| **Senses GRU** (L=3, d=575) into linear2Senses **FF-NN**, apply softmax on the logits of the senses of the most likely ***k*** globals | learning rate=10^(-4) |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |
| 1 | 21472.66 | 100000216.32 |
| 2 | 20215.56 | 100000216.32 |
| 3 | 18591.51 | 100000216.32 |
| 10 | 178.51 | 100000216.32 |
| 50 | 23.89 | 100000216.32 |
| 100 | 23.79 | 100000216.32 |
| 150 | 18.44 | 100000216.32 |
| 175 | 3.4 | 105.54 |
| 200 | 1.07 | 13.92 |
| 250 | 1.01 | 13.91 |
| 300 | 1.01 | 13.9 |

### Mini-experiment 4 – Analysis of the predictions

(standard correct predictions for globals and senses, with >99% probability for the label, are not reported)

|  |  |  |
| --- | --- | --- |
| Label: the next global is: investigation(from 4) |  | INFO : Label: the next sense is: probe.n.01(from 17882) |
| Label: the next global is: recent(from 8) |  | INFO : Label: the next sense is: late.s.03(from 13363) |
| Label: the next global is: primary election(from 9) |  | INFO : Label: the next sense is: primary.n.01(from 17809) |
| Label: the next global is: produced(from 10) |  | INFO : Label: the next sense is: produce.v.04(from 17913) |
| Label: the next global is: evidence(from 13) |  | INFO : Label: the next sense is: evidence.n.01(from 8606) |
| Label: the next global is: took place(from 17) |  | INFO : Label: the next sense is: happen.v.01(from 11119) |
| Label: the next global is: jury(from 19) |  | INFO : Label: the next sense is: jury.n.01(from 13063) |
| Label: the next global is: further(from 20) |  | INFO : Label: the next sense is: far.r.02(from 9122) |
| Label: the next global is: said(from 1) |  | INFO : Label: the next sense is: state.v.01(from 22232) |
| Label: the next global is: term(from 22) |  | INFO : Label: the next sense is: term.n.02(from 23451) |
| Label: the next global is: end(from 23) |  | INFO : Label: the next sense is: end.n.02(from 8298) |
| Label: the next global is: <unk>(from 21987) | *Probably the global is* Fulton\_County\_Grand\_Jury  *reported as <unk>* | INFO : Label: the next sense is: group.n.01(from 10898) |
| Label: the next global is: <unk>(from 21987) | *Probably the global is* City\_Executive\_Committee  *reported as <unk>* | INFO : Label: the next sense is: group.n.01(from 10898) |
| Label: the next global is: said(from 1) |  | INFO : Label: the next sense is: state.v.01(from 22232) |
| Label: the next global is: investigation(from 4) |  | INFO : Label: the next sense is: probe.n.01(from 17882) |
| … etc (all the others are correct) |  |  |

Some globals that appear only once in the training corpus are set to <unk>, and when we overfit on a fragment we have <unk> as the most likely global, that has no senses. So we can not guess the senses correctly with SelectK=1

Moreover, on a rerun I have:

Label: the next global is: produced(from 10)

INFO : Label: the next sense is: produce.v.04(from 17913)

INFO : The top- 5 predicted senses are:

INFO :

Why do I not manage to predict the senses here? Must debug.

This time

“Label: the next global is: said(from 1)

INFO : Label: the next sense is: state.v.01(from 22232)”

does not work…

k\_globals\_words = ‘said’

k\_globals\_lemmatized = ‘say’

lemmatized\_indices=[28113]

sense\_neighbours\_t = tensor([22232, 20173, 20178, 20175, 18711, 20177, 20176, 524, 20174, 16073, 22929], device='cuda:0')

softmax\_selected\_senses= tensor([1.1597e-05, 1.1795e-05, 1.4018e-05, 8.8305e-06, 1.0876e-05, 1.1523e-05, 1.0502e-05, 8.8967e-06, 1.1274e-05, 9.9989e-01, 1.3601e-05], device='cuda:0', grad\_fn=<SoftmaxBackward>)

quantity\_to\_subtract\_from\_selected = 2.3613636363636363e-05

**softmax\_selected\_senses** = tensor([-1.2016e-05, -1.1818e-05, -9.5958e-06, -1.4783e-05, -1.2737e-05, -1.2091e-05, -1.3112e-05, -1.4717e-05, -1.2340e-05, 9.9986e-01,

-1.0013e-05], device='cuda:0', grad\_fn=<SubBackward0>)

The newest experiment, after the adjustments, gets:

Perplexity: Globals perplexity=1.2 Sense perplexity=4.27

<unk>, said and produced still do not work.

When I try to apply the fix, I get:

RuntimeError: one of the variables needed for gradient computation has been modified by an inplace operation: [torch.cuda.FloatTensor [1]], which is output 0 of SoftmaxBackward, is at version 1; expected version 0 instead. Hint: enable anomaly detection to find the operation that failed to compute its gradient, with torch.autograd.set\_detect\_anomaly(True).

And what if I just didn’t subtract anything? Let the nll\_loss sort it out, even if the sum of the softmax values is \*slightly\* more than one (order of magnitude: epsilon x |vocab\_senses|, ~10^-8 x 10^5 =~ 10^-3).

Perplexity: Globals perplexity=1.01 Sense perplexity=4.27

Label: the next global is: said(from 1)

INFO : Label: the next sense is: state.v.01(from 22232)

INFO : Sense: state.v.01 ; p = 99.98%, now it works.

Now the only places where we do not manage to predict the senses are those where the global is <unk>.

## Mini-experiment 5

As the last modification to test before trying out the model on SemCor, we increase the number of globals we select the senses from, **k**, from 1 to 5.

### Losses

INFO : Hyperparameters: \_batchPerSeqlen32\_area32\_lr0.0001\_epochs300

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (d=1150) | 1) The word embedding of the current global (d=300) | batch\_size=4  TBPTT length=8 |
| linear2Globals **FF-NN** | grapharea=32, hops=2 |
| **Senses GRU** (L=3, d=575) into linear2Senses **FF-NN**, apply softmax on the logits of the senses of the most likely ***k=5*** globals | learning rate=10^(-4) |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |
| 1 | 21796.59 | 100000216.32 |
| 2 | 20597.76 | 5375446.1 |
| 3 | 19078.43 | 3381108.91 |
| 10 | 153.35 | 28035284.03 |
| 50 | 23.89 | 3075497.52 |
| 100 | 23.77 | 2489515.49 |
| 150 | 23.33 | 114444.77 |
| 175 | 18.07 | 14368.72 |
| 200 | 8.21 | 52.94 |
| 250 | 1.25 | 16.76 |
| 300 | 1.03 | 17.94 |

### Mini-experiment 5 – Analysis of the predictions

Mostly correct, apart from <unk>s as expected, with one exception:

Label: the next global is: investigation(from 4)

INFO : Label: the next sense is: probe.n.01(from 17882)

INFO : The top- 5 predicted globals are:

**INFO : Word: investigation ; p=96.87%**

INFO : Word: <unk> ; p=0.85%

***INFO : Word: jury ; p=0.52%***

INFO : Word: Atlanta ; p=0.39%

INFO : Word: term ; p=0.28%

INFO : The top- 5 predicted senses are:

INFO : Sense: jury.n.02 ; p = 93.08%

INFO : Sense: probe.n.01 ; p = 6.22%

And one partial exception:

Label: the next global is: produced(from 10)

INFO : Label: the next sense is: produce.v.04(from 17913)

INFO : The top- 5 predicted globals are:

**INFO : Word: produced ; p=96.67%**

INFO : Word: " ; p=0.86%

INFO : Word: any ; p=0.64%

INFO : Word: recent ; p=0.38%

INFO : Word: primary election ; p=0.38%

INFO : The top- 5 predicted senses are:

**INFO : Sense: produce.v.04 ; p = 64.3%**

INFO : Sense: any.r.01 ; p = 21.21%

INFO : Sense: recent.s.01 ; p = 6.0%

INFO : Sense: produce.v.03 ; p = 3.15%

INFO : Sense: produce.v.02 ; p = 1.75%

**Observations**: increasing k moves the “burden” of prediction from the globals, that have an easier task (as long as they manage to include the correct global among the k most likely ones) to the senses (since we have to discriminate among more senses).

It is probably worth to explore k=1(as a baseline – the senses of the most likely predicted global), k=3, k=5.

Moreover, alternative architectures for the senses’logits may help. We noted previously that the senses’part “activates” only when there is a sense label, and thus does not have a continuous view of the text. Maybe instead of 3layers with hd=575 we should have 1layer with 1150?

## Experiment 1 – SelectK1 on SemCor

INFO : Model:

INFO : DataParallel(

(module): SelectK(

(main\_gru): GRU(300, 1150, num\_layers=3)

(gru\_senses): GRU(300, 575, num\_layers=3)

(linear2global): Linear(in\_features=1150, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=575, out\_features=25986, bias=True)

)

)

INFO : Hyperparameters: \_batchPerSeqlen1400\_area32\_lr5e-05\_epochs100

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (d=1150) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| linear2Globals **FF-NN** | grapharea=32, hops=2 |
| **Senses GRU** (L=3, d=575) into linear2Senses **FF-NN**, apply softmax on the logits of the senses of the most likely ***k=1*** global | learning rate=5\*10^(-5) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* | *Validation PPL on globals* | *Validation PPL on senses* |
| 1 | 918.46 | 28057.44 | 434.44 | 25986.01 |
| 2 | 490.94 | 26994.04 | 314.98 | **25549.72** |
| 3 | 367.5 | 31471.46 | 268.2 | 47641.5 |
| 4 | 309.25 | 44004.55 | 243.77 | 66291.41 |
| 5 | 272.81 | 46505.83 | 228.62 | 74292.94 |
| 6 | 246.53 | 49694.05 | 219.81 | 78444.04 |
| 7 | 225.84 | 52122.73 | 212.56 | 81960.8 |
| 8 | 208.45 | 54768.2 | 206.16 | 84532.23 |
| 9 | 193.63 | 57472.16 | 203.44 | 86204.86 |
| 10 | 180.89 | 59758.27 | 199.58 | 88047.98 |
| 11 | 169.68 | 61785.0 | 201.01 | 87741.85 |
| 12 | 159.69 | 63496.94 | 197.92 | 90574.67 |
| 13 | 150.52 | 65106.49 | 193.72 | 93140.19 |
| 14 | 142.27 | 66584.56 | 193.75 | 94614.72 |
| 15 | 134.68 | 68381.98 | 194.04 | 99078.3 |
| 16 | 127.65 | 70024.39 | 192.0 | 102214.42 |
| 17 | 121.23 | 71601.27 | 193.27 | 105976.18 |
| 18 | 115.08 | 72829.32 | 192.83 | 112505.01 |
| 19 | 109.42 | 74349.62 | 192.25 | 119116.06 |
| 20 | 104.15 | 75866.42 | 192.37 | 127979.96 |
| 21 | 99.13 | 76963.22 | 192.07 | 131361.44 |
| 22 | 94.44 | 77852.04 | 191.46 | 139729.8 |
| 23 | 90.09 | 78860.22 | **190.51** | 141888.52 |
| 24 | 85.91 | 79598.94 | 192.02 | 150453.28 |
| 25 | 81.98 | 80649.1 | 192.48 | 156545.26 |

Validation perplexity on globals = 190.5

Validation perplexity on senses = 25549.7

Which is extremely high.

**Hypothesis:**

We only try to predict the senses from the most likely predicted global, k=1.

If the globals’ prediction is unable to choose the correct global, we are only adjusting the weights (and softmax probabilities) to choose among a set of senses that does not overlap with the correct solution.

**Follow-up:**

Set k=5 and verify what happens.

# Multi-sense LM. Part 2

## Checking & Debugging

Before doing anything else, trying architectures, experiments etc. it is necessary to verify that the node indices and retrieval are correct.

Let us examine the variables when training on the fragment of SemCor (Mini-dataset 2: 63 tokens)

Setting batch\_size=1 and seq\_len=1 for simplicity.

At step 1,

(input) = (global\_index, sense\_index)=(0, -1), corresponding to : (The, no\_sense)

(labels) = (global\_index, sense\_index)=(21987, 10898) (<unk>, group.n.01)

Regarding the batch input tensor and how it’s unpacked:

For example, operating with batch\_size=1 and seq\_len=2, we have

batch\_input.shape=torch.Size([1, 2, 1150])

Only 1 element in the batch size, so we have

sequences\_in\_the\_batch\_ls = [batchinput\_tensor]

Then,

padded\_sequence = padded\_sequence.squeeze() # so we obtain [2,1150]. We should probably only squeeze 1 dimension here.

Back on track:

x\_indices\_g = tensor([25986]) (corresponding to global n.0 + number of senses)

batch\_input\_signals.shape= torch.Size([1, 1, 300]) # no other input signals here

Then,

senses\_gru\_out, hidden\_s =self.gru\_senses(batch\_input\_signals, self.memory\_hn\_senses)

followed by logits\_sense = self.linear2senses(senses\_gru\_out), etc.

The most likely global is: n.19876, “collects”

It is a consequence of the random initialization of the GRU.

lemmatized indices=[33154], corresponding to the 7168th global ‘collect’

sense\_neighbours\_t = tensor([ 4593, 4590, 4591, 4594, 4595, 10373, 4592]), ‘collect.n.01’, … , ‘collect.n.05’, ‘gather.v.01’

It seems that now the retrieval of sense nodes is correct.

The next steps & experiments will be:

* **Baseline GRU** for the senses, no methods
* **SelectK** with **k**=5 (k=1 was executed after the modifications to the retrieval mechanism that gave us the current, correct version of the models)
* SelectK with **k**=10, or alternatively
* The **Self-attention** mechanism using the sense embeddings

## Baseline 0: 2 GRUs

### Model

1 GRU for the globals, 1 for the Senses. No specific mechanism or refinement for the Senses, and no connection between the 2 predictions (apart from the moveable embeddings for globals, senses, defs and examples).

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)

)

(rnn\_senses): GRU(300, 1024, num\_layers=2)

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

(linear2senses): Linear(in\_features=1024, out\_features=25986, bias=True)))

INFO : Parameters:

INFO : ('module.X', torch.Size([99963, 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([2, 4, 1024]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([2, 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.rnn\_senses.weight\_ih\_l0', torch.Size([3072, 300]), torch.float32, True)

('module.rnn\_senses.weight\_hh\_l0', torch.Size([3072, 1024]), torch.float32, True)

('module.rnn\_senses.bias\_ih\_l0', torch.Size([3072]), torch.float32, True)

('module.rnn\_senses.bias\_hh\_l0', torch.Size([3072]), torch.float32, True)

('module.rnn\_senses.weight\_ih\_l1', torch.Size([3072, 1024]), torch.float32, True)

('module.rnn\_senses.weight\_hh\_l1', torch.Size([3072, 1024]), torch.float32, True)

('module.rnn\_senses.bias\_ih\_l1', torch.Size([3072]), torch.float32, True)

('module.rnn\_senses.bias\_hh\_l1', torch.Size([3072]), torch.float32, True)

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

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

('module.linear2senses.weight', torch.Size([25986, 1024]), torch.float32, True)

('module.linear2senses.bias', torch.Size([25986]), torch.float32, True)

INFO : Number of trainable parameters=91.008M

### Mini-experiment: overfit on fragment of SemCor

|  |  |  |
| --- | --- | --- |
| **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 |
| **Senses GRU** with 2 layers (1024 > 1024) | learning rate=10^(-3) |

INFO : Hyperparameters: \_batchPerSeqlen32\_area32\_lr0.0005\_epochs300

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |
| 1 | 19936.75 | 22948.16 |
| 2 | 2651.57 | 3397.39 |
| 3 | 288.8 | 20.79 |
| 10 | 25.17 | 11.21 |
| 50 | 23.61 | 1.54 |
| 100 | 20.34 | 1.05 |
| 150 | 17.46 | 1.02 |
| 200 | 15.21 | 1.01 |
| 250 | 14.22 | 1.01 |
| 300 | 13.64 | 1.01 |

This GRU architecture for the globals appears to be slower to learn to overfit than other alternatives.

Regardless, I will try mini-experiment hyperparameters again, but in the meantime I will start the standard experiment on the full SemCor

### Experiment: GRUs on SemCor

|  |  |  |
| --- | --- | --- |
| **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 |
| **Senses GRU** with 2 layers (1024 > 1024) | learning rate=10^(-4) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* | *Validation perplexity on globals* | *Validation Perplexity on senses* |
| 1 | 960.38 | 4852.58 | 487.37 | **793.37** |
| 2 | 608.2 | 3104.06 | 372.77 | 805.0 |
| 3 | 437.28 | 2688.92 | 297.47 | 862.89 |
| 4 | 338.82 | 2532.54 | 250.72 | 1183.94 |
| 5 | 281.69 | 2409.66 | 226.53 | 1274.36 |
| 6 | 247.24 | 2358.39 | 214.41 | 891.52 |
| 7 | 222.84 | 2341.0 | 205.94 | 1434.78 |
| 8 | 203.59 | 2177.51 | 199.12 | 1464.13 |
| 9 | 187.78 | 2071.02 | 195.98 | 1649.94 |
| 10 | 174.29 | 1980.39 | 192.04 | 1799.48 |
| 11 | 162.52 | 1980.49 | 193.21 | 2270.78 |
| 12 | 152.06 | 1788.65 | 189.99 | 2408.59 |
| 13 | 142.42 | 1743.1 | 185.98 | 1681.45 |
| 14 | 133.8 | 1565.92 | **185.95** | 2171.4 |
| 15 | 125.8 | 1476.37 | 186.36 | 2240.74 |
| 16 | 118.53 | 1352.14 | 184.78 | 3159.46 |
| 17 | 111.91 | 1264.11 | 186.69 | 2708.47 |
| 18 | 105.64 | 1164.57 | 187.41 | 2474.85 |
| 19 | 100.05 | 1063.79 | 187.73 | 3647.71 |
| 20 | 95.04 | 983.45 | 189.34 | 4330.14 |
| … |  |  |  |  |
| 40 | 34.88 | 138.33 | 256.96 | 5288.73 |
| … |  |  |  |  |
| 60 | 12.98 | 19.67 | 463.47 | 9549.37 |

The GRUs eventually overfit on SemCor.

Best values for the Valid-PPL: on globals=185.9. on senses=793.4, reached immediately in Epoch 1.

### Experiment: GRUs on SemCor – version 2

This time we use the architecture 1024>1024>**1024**, that in the SelectK5 mini-experiment actually manages to overfit on a fragment for the globals.

Lower learning rate.

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (1024 > 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 |
| **Senses GRU** with 2 layers (1024 > 1024) | learning rate=5\*10^(-5) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* | *Validation perplexity on globals* | *Validation Perplexity on senses* |
| 1 | 972.76 | 5214.45 | 466.09 | 842.91 |
| 2 | 527.05 | 3437.78 | 332.01 | **811.1** |
| 3 | 389.62 | 3110.54 | 279.08 | 817.96 |
| 4 | 324.65 | 2892.64 | 250.44 | 842.01 |
| 5 | 284.52 | 2780.03 | 233.17 | 856.07 |
| 6 | 256.73 | 2705.51 | 223.32 | 886.45 |
| 7 | 235.48 | 2650.44 | 215.0 | 1649.36 |
| 8 | 217.94 | 2793.9 | 207.86 | 978.71 |
| 9 | 203.1 | 2633.58 | 204.51 | 2335.16 |
| 10 | 190.3 | 2464.52 | 200.2 | 1864.83 |
| 11 | 178.99 | 2525.9 | 201.09 | 1082.45 |
| 12 | 168.89 | 2447.28 | 197.38 | 1165.9 |
| 13 | 159.62 | 2387.41 | 192.88 | 1689.99 |
| 14 | 151.26 | 2240.95 | 192.46 | 1418.81 |
| 15 | 143.58 | 2188.04 | 192.3 | 1972.79 |
| 16 | 136.46 | 2081.01 | 190.04 | 1957.07 |
| 17 | 129.97 | 2102.42 | 191.05 | 1755.35 |
| 18 | 123.73 | 1965.57 | 190.37 | 2243.94 |
| 19 | 118.02 | 1942.21 | 189.86 | 1798.67 |
| 20 | 112.7 | 1886.14 | 190.0 | 2460.83 |
| 21 | 107.6 | 1840.18 | 189.92 | 2157.65 |
| 22 | 102.81 | 1780.39 | 189.58 | 2446.24 |
| 23 | 98.37 | 1726.86 | **188.67** | 1950.77 |
| 24 | 94.12 | 1777.95 | 190.4 | 2151.06 |
| 25 | 90.06 | 1648.05 | 190.87 | 2376.13 |
| … |  |  |  |  |
| 40 | 48.05 | 876.12 | 212.67 | 3407.63 |
| … |  |  |  |  |
| 60 | 24.65 | 447.34 | 251.02 | 4499.36 |

## SelectK5

### Model

Idea: make the senses’ task dependent on the globals’ standard language model task, choosing among the senses of the most likely *k* globals.

We consider the globals’ logits (main GRU > FF-NN, standard language model), and the senses’ logits (senses’ GRU > FF-NN):

logits\_global = self.linear2global(main\_gru\_out)

logits\_sense = self.linear2senses(senses\_gru\_out)

Sort the logits of the globals, to see which globals are most likely in our LM prediction:

k\_globals\_indices = logits\_global.sort(descending=**True**).indices[:, 0:self.k]

sample\_k\_indices\_lls\_relative = k\_globals\_indices.tolist()

(’relative’ means ’in-vocabulary’, without considering X and the last sense idx)

For every sample *s*: (we are handling batch\_size x seq\_len samples)

k\_globals\_relative\_indices = sample\_k\_indices\_lls\_relative[s]

k\_globals\_words > k\_globals\_lemmatized

Take the most likely *k* globals for the current sample. We lemmatize them, so we turn ‘said’ into ‘say’, ‘collects’ into ‘collect’, ‘are’ into ‘be’, etc.

lemmatized\_indices

sense\_neighbours\_t = get\_senseneighbours\_of\_k\_globals(self, lemmatized\_indices)

Using the grapharea\_matrix for speed, we retrieve the neighbours of every most-likely global, and collect them in a tensor, keeping only the nodes with the index in the range of senses.

Next: from the sensesGRU>linear2Senses, we select the logits of the senses of the most likely k globals:

logits\_selected\_senses = sample\_logits\_senses.index\_select(dim=0, index=sense\_neighbours\_t)

and apply a softmax only over those:

softmax\_selected\_senses = tfunc.softmax(input=logits\_selected\_senses, dim=0)

Then, having defined a probability distribution over the whole vocabulary of senses with a fixed ε value for every sense (epsilon = 10 \*\* (-8)), we copy-paste the values from the softmax over the selected senses.

senses\_softmax[s].masked\_scatter\_(mask=i\_senseneighbours\_mask[s].data.clone(), source=softmax\_selected\_senses)

As a side effect, this means that the sum of the softmax over the whole vocabulary will be slightly more than 1, sum=1+(10^(-8)\*num\_senses) = 1+ 0.00256986.

Currently, the sum>1 does not seem to produce a significant error. It could be possible to subtract this probability mass from the most likely entry (or entries) in the selected senses. In this case, one needs to be careful not to push very unlikely selected entries to have a probability < 0, because this breaks the nll\_loss numerically.

Eventually, INFO : Model:

INFO : DataParallel(

(module): SelectK(

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

(main\_rnn\_ls): ModuleList(

(0): GRU(300, 1024)

(1): GRU(1024, 1024)

(2): GRU(1024, 1024))

(gru\_senses): GRU(300, 1024, num\_layers=2)

(linear2global): Linear(in\_features=1024, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=1024, out\_features=25986, bias=True))

)

### Mini-experiment: overfit on fragment of SemCor

**Observation**:

The 1024 > 1024 > 512 architecture for the globals DOES NOT manage to overfit on a fragment. Instead, an architecture 1024 > 1024 > 1024 does.

Thus, we change the main\_gru – and above we repeat the experiments regarding the GRU-baseline.

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (d=1024) | 1) The word embedding of the current global (d=300) | batch\_size=4  TBPTT length=8 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
| **Senses GRU** (**L=2, d=1024**) into linear2Senses FF-NN, apply softmax on the logits of the senses of the most likely ***k=5*** globals | learning rate=10^(-3) |

INFO : Hyperparameters: \_batchPerSeqlen32\_area32\_lr0.001\_epochs400

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* |
| 1 | 19691.81 | 18904264.16 |
| 2 | 734.52 | 28635572.41 |
| 3 | 45.67 | 10122294.55 |
| 10 | 24.34 | 3791925.79 |
| 50 | 22.56 | 1295454.06 |
| 100 | 12.69 | 50190.08 |
| 150 | 9.52 | 83.24 |
| 200 | 6.2 | 7.28 |
| 250 | 4.06 | 4.85 |
| 300 | 2.83 | 8.53 |
| 350 | 1.78 | 1.13 |

### Experiment: SelectK5 on SemCor

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (d=1024) | 1) The word embedding of the current global (d=300) | batch\_size=40  TBPTT length=35 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
| **Senses GRU** (**L=2, d=**1024) into linear2Senses FF-NN, apply softmax on the logits of the senses of the most likely ***k=5*** globals | learning rate=5\*10^(-5) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity on globals* | *Training perplexity on senses* | *Validation perplexity on globals* | *Validation Perplexity on senses* |
| 1 | 966.81 | 31392.45 | 459.5 | **25986.01** |
| 2 | 534.55 | 112360.31 | 340.65 | 405029.77 |
| 3 | 406.77 | 2021427.64 | 288.92 | 1602356.61 |
| 4 | 335.86 | 5084236.43 | 255.78 | 1754059.22 |
| 5 | 291.11 | 4701992.56 | 236.08 | 1808811.46 |
| 6 | 260.5 | 4655555.52 | 225.17 | 1831024.41 |
| 7 | 237.87 | 4717672.25 | 216.66 | 1894365.78 |
| 8 | 219.72 | 4688206.98 | 209.8 | 2094054.77 |
| 9 | 204.64 | 4577951.25 | 206.46 | 2036662.93 |
| 10 | 191.8 | 4410303.11 | 202.24 | 2034214.03 |
| 11 | 180.6 | 4346401.35 | 203.27 | 1948670.31 |
| 12 | 170.73 | 4297053.32 | 199.64 | 1902468.18 |
| 13 | 161.65 | 4206292.45 | 194.88 | 1985912.05 |
| 14 | 153.5 | 4111808.46 | 194.5 | 1905634.53 |
| 15 | 146.01 | 4078699.91 | 194.63 | 1869208.05 |
| 16 | 139.06 | 4016920.82 | **192.49** | 1820520.83 |
| 17 | 132.74 | 3976029.51 | 193.47 | 1852512.54 |
| 18 | 126.63 | 3931912.18 | 192.79 | 1883425.9 |
| 19 | 121.06 | 3850101.57 | 192.45 | 1953383.04 |
| 20 | 115.85 | 3793525.22 | 192.54 | 1988254.07 |

Observations: the senses’ prediction is consistently incorrect, and given that we add an epsilon of 10^-8 to every sense that does not belong to the top k most likely globals, this means that the nll\_loss in those samples is ~10^8, and thus brings up the PPL on senses.

Idea: do not use the nll\_loss / PPL to evaluate the senses’ prediction when operating with structured prediction / SelectK.

Add the evaluation measures: “correct label” and “correct sense in the top-5 senses”.

After epoch 10, the Training and Valid PPL on senses decreases, so the model is learning something. However, we do not know how relevant this is until we consider the number of correct labels, given the problem with computing the Senses’ PPL in this architecture.

## Recording the number of correct predictions

We check at epoch 300, when a mini-experiment of SelectK has reached overfitting.

In Training.py, at compute\_model\_loss():

batch\_labels\_globals =

tensor([21987, 1, 2, 3, 4, 5, 6, 5197, 8, 9,

10, 11, 12, 13, 11, 14, 15, 16, 17, 18,

21987, 0, 19, 20, 1, 21, 22, 23, 21987, 14,

24, 21987], device='cuda:0')

batch\_labels\_senses =

tensor([10898, 22232, -1, -1, 17882, -1, -1, -1, 13363, 17809,

17913, -1, -1, 8606, -1, -1, -1, -1, 11119, -1,

-1, -1, 13063, 9122, 22232, -1, 23451, 8298, -1, -1,

-1, 10898], device='cuda:0')

indices\_g[:, 0] =

tensor([ 1, 21, 24, 18, 27, 19, 28, 0, 20, 31, 14, 17, 9, 5, 3, 26, 7, 29,

15, 11, 10, 12, 25, 2, 22, 30, 23, 13, 6, 4, 16, 8],

device='cuda:0')

torch.sum(indices\_g[:, 0] == batch\_labels\_globals) = tensor(0)

### Batches bug while reading sense-labeled corpus

In the verbose batch logging, when operating on the fragment of SemCor (i.e. on the SLC corpus):

the next batch is only switched by 1 word forward compared to the previous one. While this is a possible way to handle BPTT, it is *not* the intended way.

On the fragment\_semcor.xml, we get:

Original corpus:

<paragraph>

<sentence>

<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::"/>

<word surface\_form="**of**" pos="IN"/>

<word surface\_form="**Atlanta**" lemma="atlanta" pos="NNP" wn16\_key="atlanta%1:15:00::" wn30\_key="atlanta%1:15:00::"/>

<word surface\_form="'**s**" pos="POS"/>

<word surface\_form="**recent**" lemma="recent" pos="JJ" wn16\_key="recent%5:00:00:past:00" wn30\_key="recent%3:00:00:past:00"/>

<word surface\_form="**primary\_election**" lemma="primary\_election" pos="NN" wn16\_key="primary\_election%1:04:00::" wn30\_key="primary\_election%1:04:00::"/>

<word surface\_form="**produced**" lemma="produce" pos="VBD" wn16\_key="produce%2:39:01::" wn30\_key="produce%2:39:01::"/>

<word surface\_form="&**quot**;" pos="''"/>

<word surface\_form="**no**" pos="DT"/>

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

<word surface\_form="&**quot**;" pos="''"/>

<word surface\_form="**that**" pos="IN"/>

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

<word surface\_form="**irregularities**" lemma="irregularity" pos="NNS" wn16\_key="irregularity%1:04:00::" wn30\_key="irregularity%1:04:00::"/>

<word surface\_form="**took\_place**" lemma="take\_place" pos="VB" wn16\_key="take\_place%2:30:00::" wn30\_key="take\_place%2:30:00::"/>

<word surface\_form="**.**" pos="."/>

</sentence>

</paragraph>

<paragraph>

<sentence>

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

<word surface\_form="**jury**" lemma="jury" pos="NN" wn16\_key="jury%1:14:00::" wn30\_key="jury%1:14:00::"/>

<word surface\_form="**further**" lemma="far" pos="RB" wn16\_key="far%4:02:00::" wn30\_key="far%4:02:00::"/>

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

<word surface\_form="**in**" pos="IN"/>

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

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

<word surface\_form="**presentments**" lemma="presentment" pos="NNS" wn16\_key="presentment%1:04:00::" wn30\_key="presentment%1:04:00::"/>

<word surface\_form="**that**" pos="IN"/>

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

<word surface\_form="**City\_Executive\_Committee**" lemma="group" pos="NN" wn16\_key="group%1:03:00::" wn30\_key="group%1:03:00::"/>

<word surface\_form="**,**" pos=","/>

<word surface\_form="**which**" pos="WDT"/>

<word surface\_form="**had**" lemma="have" pos="VBD" wn16\_key="have%2:40:04::" wn30\_key="have%2:40:04::"/>

<word surface\_form="**over-all**" lemma="overall" pos="JJ" wn16\_key="overall%5:00:00:gross:00" wn30\_key="overall%3:00:00:gross:00"/>

<word surface\_form="**charge**" lemma="charge" pos="NN" wn16\_key="charge%1:04:03::" wn30\_key="charge%1:04:03::"/>

<word surface\_form="**of**" pos="IN"/>

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

<word surface\_form="**election**" lemma="election" pos="NN" wn16\_key="election%1:04:01::" wn30\_key="election%1:04:01::"/>

<word surface\_form="**,**" pos=","/>

<word surface\_form="&**quot**;" pos="''"/>

<word surface\_form="**deserves**" lemma="deserve" pos="VBZ" wn16\_key="deserve%2:42:00::" wn30\_key="deserve%2:42:00::"/>

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

<word surface\_form="**praise**" lemma="praise" pos="NN" wn16\_key="praise%1:10:00::" wn30\_key="praise%1:10:00::"/>

<word surface\_form="**and**" pos="CC"/>

<word surface\_form="**thanks**" lemma="thanks" pos="NNS" wn16\_key="thanks%1:10:00::" wn30\_key="thanks%1:10:00::"/>

<word surface\_form="**of**" pos="IN"/>

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

<word surface\_form="**City\_of\_Atlanta**" lemma="location" pos="NN" wn16\_key="location%1:03:00::" wn30\_key="location%1:03:00::"/>

<word surface\_form="&**quot**;" pos="''"/>

<word surface\_form="**for**" pos="IN"/>

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

<word surface\_form="**manner**" lemma="manner" pos="NN" wn16\_key="manner%1:07:02::" wn30\_key="manner%1:07:02::"/>

<word surface\_form="**in**" lemma="in" pos="RB"/>

<word surface\_form="**which**" pos="RB"/>

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

<word surface\_form="**election**" lemma="election" pos="NN" wn16\_key="election%1:04:01::" wn30\_key="election%1:04:01::"/>

<word surface\_form="**was**" lemma="be" pos="VBD"/>

<word surface\_form="**conducted**" lemma="conduct" pos="VBN" wn16\_key="conduct%2:41:00::" wn30\_key="conduct%2:41:00::"/>

<word surface\_form="**.**" pos="."/>

</sentence>

batch 0:

-----

Label: the next global is: <**unk**>(from 21987)

INFO : Label: the next sense is: group.n.01(from 10898)

Label: the next global is: **said**(from 1)

INFO : Label: the next sense is: state.v.01(from 22232)

Label: the next global is: **Friday**(from 2)

Label: the next global is: **an**(from 3)

Label: the next global is: **investigation**(from 4)

INFO : Label: the next sense is: probe.n.01(from 17882)

Label: the next global is: **of**(from 5)

Label: the next global is: **Atlanta**(from 6)

Label: the next global is: **s**(from 5197)

Label: the next global is: **recent**(from 8)

INFO : Label: the next sense is: late.s.03(from 13363)

Label: the next global is: **primary** **election**(from 9)

INFO : Label: the next sense is: primary.n.01(from 17809)

Label: the next global is: **produced**(from 10)

INFO : Label: the next sense is: produce.v.04(from 17913)

Label: the next global is: **"**(from 11)

Label: the next global is: **no**(from 12)

Label: the next global is: **evidence**(from 13)

INFO : Label: the next sense is: evidence.n.01(from 8606)

Label: the next global is: **"**(from 11)

Label: the next global is: **that**(from 14)

Label: the next global is: **any**(from 15)

Label: the next global is: **irregularities**(from 16)

Label: the next global is: **took** **place**(from 17)

INFO : Label: the next sense is: happen.v.01(from 11119)

Label: the next global is: <**unk**>(from 21987)

Label: the next global is: **The**(from 0)

Label: the next global is: **jury**(from 19)

INFO : Label: the next sense is: **jury**.n.01(from 13063)

Label: the next global is: **further**(from 20)

INFO : Label: the next sense is: far.r.02(from 9122)

Label: the next global is: **said**(from 1)

INFO : Label: the next sense is: state.v.01(from 22232)

Label: the next global is: **in**(from 21)

Label: the next global is: **term**(from 22)

INFO : Label: the next sense is: term.n.02(from 23451)

Label: the next global is: **end**(from 23)

INFO : Label: the next sense is: end.n.02(from 8298)

Label: the next global is: <**unk**>(from 21987)

Label: the next global is: **that**(from 14)

Label: the next global is: **the**(from 24)

Label: the next global is: <**unk**>(from 21987)

INFO : Label: the next sense is: group.n.01(from 10898)

batch n.1:

compute\_model\_loss > verbose logging of batch

Label: the next global is: <**unk**>(from 21987)

INFO : Label: the next sense is: group.n.01(from 10898)

Label: the next global is: **said**(from 1)

INFO : Label: the next sense is: state.v.01(from 22232)

Label: the next global is: **Friday**(from 2)

Label: the next global is: **an**(from 3)

Label: the next global is: **investigation**(from 4)

INFO : Label: the next sense is: probe.n.01(from 17882)

…

Label: the next global is: **in**(from 21)

Label: the next global is: **term**(from 22)

INFO : Label: the next sense is: term.n.02(from 23451)

Label: the next global is: **end**(from 23)

INFO : Label: the next sense is: end.n.02(from 8298)

Label: the next global is: <**unk**>(from 21987)

Label: the next global is: **that**(from 14)

Label: the next global is: **the**(from 24)

Label: the next global is: <**unk**>(from 21987)

INFO : Label: the next sense is: group.n.01(from 10898)

And how does it work on the standard text corpus?

Fragment selected from wiki.train.tokens for WikiText-2:

“Markgraf was present during the fleet operation that resulted in the Battle of Jutland which took place on 31 May and 1 June 1916 . The German fleet again sought to draw out and isolate a portion of the Grand Fleet and destroy it before the main British fleet could retaliate . Markgraf was the third ship in the German line , behind her sisters König and Grosser Kurfürst and followed by Kronprinz …”

Batch 0:

compute\_model\_loss > verbose logging of batch

Label: the next global is: was(from 37)

Label: the next global is: present(from 324)

Label: the next global is: during(from 1674)

Label: the next global is: the(from 24)

Label: the next global is: fleet(from 18419)

Label: the next global is: operation(from 279)

Label: the next global is: that(from 14)

Label: the next global is: resulted(from 1911)

batch 1:

compute\_model\_loss > verbose logging of batch

Label: the next global is: in(from 21)

Label: the next global is: the(from 24)

Label: the next global is: Battle(from 19836)

Label: the next global is: of(from 5)

Label: the next global is: <unk>(from 21987)

Label: the next global is: which(from 26)

Label: the next global is: took(from 1326)

Label: the next global is: place(from 199)

compute\_model\_loss > verbose logging of batch

…

So the standard text corpus is OK, but the generator for the Sense-Labeled Corpus is not.

Time to review DataLoading.py

The generator seems to be correct, both the standard text and the SLC.

When we try again the SLC, with batch\_size=2 and seq\_len=4, it behaves correctly.

compute\_model\_loss > verbose logging of batch

Label: the next global is: <unk>(from 21987)

INFO : Label: the next sense is: group.n.01(from 10898)

Label: the next global is: said(from 1)

INFO : Label: the next sense is: state.v.01(from 22232)

Label: the next global is: Friday(from 2)

Label: the next global is: an(from 3)

Label: the next global is: investigation(from 4)

INFO : Label: the next sense is: probe.n.01(from 17882)

Label: the next global is: of(from 5)

Label: the next global is: Atlanta(from 6)

Label: the next global is: s(from 5197)

compute\_model\_loss > verbose logging of batch

Label: the next global is: recent(from 8)

INFO : Label: the next sense is: late.s.03(from 13363)

Label: the next global is: primary election(from 9)

INFO : Label: the next sense is: primary.n.01(from 17809)

Label: the next global is: produced(from 10)

INFO : Label: the next sense is: produce.v.04(from 17913)

Label: the next global is: "(from 11)

INFO : Label: the next sense is: None(from -1)

Label: the next global is: no(from 12)

Label: the next global is: evidence(from 13)

INFO : Label: the next sense is: evidence.n.01(from 8606)

Label: the next global is: "(from 11)

Label: the next global is: that(from 14)

INFO : updated\_predictions\_history\_dict = {'correct\_g': 0, 'top\_k\_g': 0, 'tot\_g': 16, 'correct\_s': 0, 'top\_k\_s': 0, 'tot\_s': 7}

INFO : \*\*\*\*\*\*\*

compute\_model\_loss > verbose logging of batch

Label: the next global is: any(from 15)

Label: the next global is: irregularities(from 16)

Label: the next global is: took place(from 17)

…

With batch\_size=3, seq\_len=6, it repeats the first batch at the last place. Why?

The same phenomenon is found in the text corpus: the first batch is repeated at the last place.

Is this a consequence of itertools(cycle(train\_dataloader)))?

(In any case, it is opportune to decrease the hyperparameters when operating on a fragment, from 4x8 to 2x4)

Considerations:

Operating with large text corpuses, the fact that the first batch is repeated in place of the last is ultimately irrelevant.

When handling a fragment, we should decrease batch\_size and seq\_len to decrease the impact of this bug.

### Again on the number of correct predictions

Thanks to moving the sorting on dimension 1, this time, when we overfit on a fragment, the count of correct predictions will be for instance:

INFO : updated\_predictions\_history\_dict = {'correct\_g': 47, 'top\_k\_g': 0, 'tot\_g': 48, 'correct\_s': 17, 'top\_k\_s': 0, 'tot\_s': 20}

(we proceed now to implement the top\_k count, where we increment the counter if label \in first\_k\_predictions).

## SelectK – with number of correct predictions

This time, we use the new architecture for SelectK (1024x3), and more notably we register the number of correct predictions, and the number of times that the solution is among the first 10 most likely predictions.

We start the experiments on the SelectK with k=1, as a baseline.

### SelectK1 on SemCor

INFO : Model:

INFO : DataParallel(

(module): SelectK(

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

(main\_rnn\_ls): ModuleList(

(0): GRU(300, 1024)

(1): GRU(1024, 1024)

(2): GRU(1024, 1024))

(gru\_senses): GRU(300, 1024, num\_layers=2)

(linear2global): Linear(in\_features=1024, out\_features=21988, bias=True)

(linear2senses): Linear(in\_features=1024, out\_features=25986, bias=True)

))

INFO : Parameters:

INFO : ('module.X', torch.Size([99963, 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\_hn\_senses', torch.Size([2, 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([3072, 1024]), torch.float32, True)

('module.main\_rnn\_ls.2.weight\_hh\_l0', torch.Size([3072, 1024]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_ih\_l0', torch.Size([3072]), torch.float32, True)

('module.main\_rnn\_ls.2.bias\_hh\_l0', torch.Size([3072]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l0', torch.Size([3072, 300]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l0', torch.Size([3072, 1024]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l0', torch.Size([3072]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l0', torch.Size([3072]), torch.float32, True)

('module.gru\_senses.weight\_ih\_l1', torch.Size([3072, 1024]), torch.float32, True)

('module.gru\_senses.weight\_hh\_l1', torch.Size([3072, 1024]), torch.float32, True)

('module.gru\_senses.bias\_ih\_l1', torch.Size([3072]), torch.float32, True)

('module.gru\_senses.bias\_hh\_l1', torch.Size([3072]), torch.float32, True)

('module.linear2global.weight', torch.Size([21988, 1024]), torch.float32, True)

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

('module.linear2senses.weight', torch.Size([25986, 1024]), torch.float32, True)

('module.linear2senses.bias', torch.Size([25986]), torch.float32, True)

INFO : Number of trainable parameters=106.202M, where embeddings=29.989M, softmax = 22.538 + 26.636 = 49.174M, core=27.039Ms

INFO : Hyperparameters: \_batchPerSeqlen1120\_area32\_lr5e-05\_epochs100

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main GRU** with 3 layers (d=1024) | 1) The word embedding of the current global (d=300) | batch\_size=32  TBPTT length=35 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
| **Senses GRU** (L=2, d=1024) into linear2Senses FF-NN, apply softmax on the logits of the senses of the most likely ***k=1*** global | learning rate=5\*10^(-5) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training* | | | | | | *Validation* | | | | | |
|  | *Globals* | | | *Senses* | | | *Globals* | | | *Senses* | | |
|  | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* |
| 1 | 903.73 | 59772  / **646240** | 224401 / **646240** | 26180.4 | 0 / **180563** | 41/ **180563** | 433.69 | 11472 / **81760** | 33291 / **81760** | 25986.02 | 0 /  **7474** | 6 /  **7474** |
| 2 | 475.85 | 98720 | 248222 | 27218.12 | 194 | 373 | 311.84 | 14568 | 35170 | 25217.6 | 63 | 109 |
| 3 | 359.81 | 114825 | 265749 | 31127.7 | 1197 | 1765 | 265.2 | 15689 | 36842 | 55855.13 | 284 | 469 |
| 4 | 299.06 | 123502 | 277655 | 45109.98 | 2794 | 4302 | 238.93 | 16740 | 37820 | 93341.75 | 375 | 624 |
| 5 | 261.83 | 129245 | 284746 | 50247.58 | 3314 / 180586 | 5255 / 180586 | 224.26 | 17142 | 38374 | 92995.75 | 403 /  7454 | 672 / 7454 |
| 6 | 235.46 | 133224 | 290071 | 51278.77 | 3790 | 5934 | 214.15 | 17404 | 38729 | 96175.28 | 433 | 716 |
| 7 | 215.37 | 136100 | 294308 | 54112.81 | 4076 | 6358 | 208.01 | 17573 | 38971 | 103965.09 | 435 | 725 |
| 8 | 199.23 | 138338 | 297917 | 57368.62 | 4322 | 6662 | 204.05 | 17692 | 39175 | 108124.96 | 439 | 735 |
| 9 | 185.47 | 140445 | 301014 | 59854.02 | 4527 | 6945 | 200.08 | 17872 | 39325 | 108995.3 | 443 | 747 |
| 10 | 173.67 | 142375 | 303754 | 61897.7 | 4718 | 7210 | 197.27 | 18004 | 39482 | 112898.01 | 443 | 746 |
| 11 | 163.18 | 144085 | 306337 | 63771.16 | 4901 | 7430 | 196.62 | 18066 | 39553 | 109368.96 | **448** | **757** |
| 12 | 153.85 |  |  | 65518.66 |  |  | 194.54 | 18166 | 39669 | 107929.46 | 440 | 753 |
| 13 | 145.43 |  |  | 67256.21 |  |  | 195.32 | 18170 | 39696 | 102600.67 | 421 | 750 |
| 14 | 137.82 |  |  | 69214.7 |  |  | 196.27 | 18179 | 39703 | 102582.49 | 401 | 744 |
| 15 | 130.79 |  |  | 70635.5 |  |  | 194.08 | 18224 | 39801 | 104169.24 | 377 | 743 |
| 16 | 124.27 |  |  | 72048.91 |  |  | 191.68 | 18287 | 39948 | 106219.68 | 378 | 749 |
| 17 | 118.28 | 152980 | 319607 | 73187.93 | 5969 | 8556 | 191.49 | **18315** | **39999** | 109088.27 | 366 | 755 |
| 18 | 112.7 |  |  | 74112.9 |  |  | 191.89 | 18273 | 39986 | 111854.1 | 352 | **772** |
| 19 | 107.42 |  |  | 75209.5 |  |  | 192.81 | 18243 | 39961 | 114972.12 | 350 | 763 |
| 20 | 102.53 |  |  | 76188.19 |  |  | **191.17** | 18283 | 40119 | 121182.75 | 341 | 770 |
| 21 | 97.97 |  |  | 77272.23 |  |  | 192.99 | 18277 | 40052 | 124499.92 | 327 | 772 |
| 22 | 93.62 | 159480 | 330780 | 78406.33 | 7140 | 9372 | 193.31 | 18275 | 40038 | 124587.04 | 332 | **789** |

# Multi-sense LM, part 3

## Reviewing the Vocabulary for SemCor

*For the globals*, to make a comparison with the official results under the same conditions, it is necessary to modify the creation of the vocabulary, using the same method employed in AWD-LSTM, which is found in data.py: no pre-processing, just tokenization on whitespaces.

**…**

Thanks to implementing this, we have vocabulary\_df = 33,278, which is exactly the official vocabulary size for WikiText-2.

We can also explore what happens if we do not exclude the long tail in the vocabulary of SemCor.

With a min\_count=2, we have only 21988globals**. (based only on the Training split. I have changed this to be based on train+validation+test now, following AWD-LSTM. ~~Even if it would be more reasonable to be based only on train+validation.~~ can just follow the same procedure ~~)~~**

Without a barrier on the minimum frequency, and extracting words from all the 3 splits (training, validation, test) we get:

53,138 globals

where some phrases are included and separated by \_. For instance, from\_that\_time\_on, build\_on, executive\_officer, drying\_out.

Incidentally, all of these have frequency=1.

Since I do not have to match the vocabulary of any official result/paper, I can decide how to treat the SemCor vocabulary. Whereas WikiText-2 has a no-preprocessing policy and an official number of |Vocabulary|=33,278…

… let’s start with following the same procedure as AWD-LSTM (no pre-processing, all tokens from the 3 splits). If the distortion is too great, I may consider frequency-cut on the phrases (the tokens that contain ‘\_’)

Meeting with IA, 23/06:

1. Add a dummy sense label. This will be used by the GRU\_senses.
2. As a consequence, I must compute 2 perplexities on the GRU\_senses: one for all the words in the document, and another only for the words that have multiple senses.
3. For the AWD-LSTM, I can try using 2 models in parallel, one from the d400 embeddings and and one from the d300 FastText embeddings. Each one has tied weights, identical hyperparameters etc.  
   Then, as a transfer learning method: use the weighted, learned average of the softmax (or logits, depending on how it works numerically) coming from the 2 models.  
   This weight can be 1 number, influenced by… yet another AWD-LSTM? With which input? the 400 or the 300? Possibly the concatenation of the last encoding 400+300, with a 1-layer LSTM…

# Modifications: dummy sense label, pre-lemmatization, PPL

### Dummy sense label

In order to allow for the GRU\_senses to read through the entire text word by word, and not just perceive ‘snapshots’ of the locations that have a sense label, we introduce a ‘dummy’ sense label for all the words that do not have one (the most immediate example is stopwords: ‘for’, ‘and’, ‘of’ etc.)

Let us examine the creation of the nodes of the graph in the pipeline:

DefineGraph.py > get\_graph\_dataobject(…) > create\_graph(method, slc\_corpus)

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

1)

if method == Method.FASTTEXT:

single\_prototypes\_file = F.SPVs\_FASTTEXT\_FILE

elif method == Method.DISTILBERT:

single\_prototypes\_file = F.SPVs\_DISTILBERT\_FILE

2)

X\_definitions = load\_senses\_elements(method, Utils.DEFINITIONS)

X\_examples = load\_senses\_elements(method, Utils.EXAMPLES)

X\_senses = initialize\_senses(X\_definitions, X\_examples, average\_or\_random=True)

X\_globals = torch.tensor(np.load(os.path.join(F.FOLDER\_INPUT, single\_prototypes\_file))).to(torch.float32)

In initialize\_senses(X\_defs, X\_examples, average\_or\_random):

db\_filepath = os.path.join(F.FOLDER\_INPUT, Utils.INDICES\_TABLE\_DB)

indicesTable\_db\_c.execute(**"SELECT \* FROM indices\_table"**)

**while** (**True**):  
 db\_row = indicesTable\_db\_c.fetchone()

…

X\_senses\_ls.extend([sense\_vector])

This means we have to go back to the creation of the indices\_table.sql.

In PrepareKBInput.py > create\_senses\_indices\_table(vocabulary\_words\_ls) …

Let us examine the result of operating on a subset of the vocabulary

– using vocabulary\_wordList = vocabulary\_df['word'].to\_list().copy()[0:50] –

The start of the senses’ table follows the same trajectory we had previously:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| word\_sense | vocab\_index | start\_defs | end\_defs | start\_examples | end\_examples |
| by.r.01 | 0 | 0 | 1 | 0 | 1 |
| developed.a.01 | 1 | 1 | 2 | 1 | 3 |
| developed.s.02 | 2 | 2 | 3 | 3 | 4 |

followed by:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| word\_sense | vocab\_index | start\_defs | end\_defs | start\_examples | end\_examples |
| for Global.01. | 53 | 53 | 53 | 72 | 72 |
| Sega.Global.01 | 54 | 53 | 53 | 72 | 72 |
| Released.Global.01 | 55 | 53 | 53 | 72 | 72 |
| Japanese.Global.01 | 56 | 53 | 53 | 72 | 72 |
| ).Global.01 | 57 | 53 | 53 | 72 | 72 |
| of.Global.01 | 58 | 53 | 53 | 72 | 72 |

# note: must replace “.Global” with “.dummySense” to avoid confusion.

Let us return to DefineGraph. initialize\_senses(X\_defs, X\_examples, average\_or\_random) and its iteration cycle over

indicesTable\_db\_c.execute(**"SELECT \* FROM indices\_table"**)

We have to adjust the code to face the problem of global words containing a dot, like

‘..Global.01’ or ‘Sr..Global.01’.

### Pre-lemmatization

To avoid using the lemmatizer during the iterations of the NN models, it is opportune to insert an additional column in the vocabulary:

(index), word, frequency, **lemmatized\_form**.

## Checks

### Creating the graph and the grapharea\_matrix

graph\_dataobj =Data(edge\_index=[2, 216881], edge\_type=[216881], node\_types=[268447], num\_relations=[1], x=[268447, 300])

m = edges\_added\_per\_area = 181

k=32

There is a bug:

After node\_index=164000

adj\_edge\_index = tensor([], device='cuda:0', dtype=torch.int64)

adj\_nodes\_ls = [164624]

and all the nodes after this one, too.

Where could we be in the graph?

Senses, Globals, Definitions, Examples:

0-73,626 ; 73,626-126,765 ; 126,765-202,483 ; 202,483-268,447

At the 164624-126765=37859th definition.

In indices\_table.sql, we have

|  |  |
| --- | --- |
| word\_sense | vocab\_index |
| zoologist.n.01 | 37858 |
| Draft.Global.01 | 37859 |

# note: must replace “.Global” with “.dummySense” to avoid confusion.

This means that the dummy-sense nodes we added have no edges, and thus cause an error.

Just like the global nodes without senses (“for”, “of”, etc.), the dummy-sense nodes should have a self-loop edge, so they do not cause a bug; the self-edge is ignored by pytorch-geometric’s GNNs.

**Attempt n. 2:**

We get:

INFO : Constructing X, matrix of node features

INFO : X\_definitions.shape=torch.Size([75718, 300])

INFO : X\_examples.shape=torch.Size([65964, 300])

INFO : X\_senses.shape=torch.Size([73626, 300])

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

INFO : Defining the edges: def, exs

INFO : def\_edges\_se.\_\_len\_\_()=37859

INFO : exs\_edges\_se.\_\_len\_\_()=32982

INFO : Defining the edges: sc

INFO : sc\_edges.\_\_len\_\_()=73626

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\_\_()=16504

INFO : Globals with no edges, needing self-loops: 70

INFO : [73638, 73645, 73654, 73685, 73905, 74070, 74155, 74187, 74308, 74309, 74334, 74535, 75393, 75466, 75734, 76011, 76419, 76948, 77420, 78249, 78250, 80599, 80600, 80873, 84837, 85006, 85068, 85199, 85338, 85370, 85403, 85404, 85472, 85474, 85476, 85503, 85525, 85559, 85686, 85687, 86231, 86280, 86369, 86702, 86735, 86762, 86770, 87717, 88356, 88361, 88373, 88379, 88465, 88494, 88501, 88519, 88525, 95558, 98940, 100468, 102396, 102738, 106124, 115013, 120159, 120569, 122787, 122793, 124354, 124368]

*corresponding to*: [12, 19, 28, 59, 279, 444, 529, 561, 682, 683, 708, 909, 1767, 1840, 2108, 2385, 2793, 3322, 3794, 4623, 4624, 6973, 6974, 7247, 11211, 11380, 11442, 11573, 11712, 11744, 11777, 11778, 11846, 11848, 11850, 11877, 11899, 11933, 12060, 12061, 12605, 12654, 12743, 13076, 13109, 13136, 13144, 14091, 14730, 14735, 14747, 14753, 14839, 14868, 14875, 14893, 14899, 21932, 25314, 26842, 28770, 29112, 32498, 41387, 46533, 46943, 49161, 49167, 50728, 50742]

*corresponding to*

['"', '.', ',', 'a', ':', '-', '$', 'A', '(', ')', 'I', "'", 'D', ';', '0', '!', 'et\_al', '?', 'C', '[', ']', 'S', 'P', '/', 'b', 'K', 'c', '%', 'V', 'x', '\*', 'f', 'H', 'h', 'T', 's', 'B', 'q', 'M', 'g', 'm', 'F', 'n', 'N', 'R', 'E', 'L', 'u', 'i', 'p', 'U', 'r', 'X', 'Q', 't', 'v', 'e', '&', '^', 'o', 'k', 'd', 'y', '".', 'W', 'Y', 'J', 'Z', '\\', 'G']

**Question**: Why are they so few now, and almost only characters and punctuation signs?

**Answer**: This may be due to the fact that I have added dummySenses. Stopwords & co., like “for”, “of”, etc. now are not globals with no connections anymore: they have their for.dummySense.01, etc.

-> New Question: Does it mean that we can encounter these characters and punctuation signs in the vocabulary, and they will not have a dummy sense?

The indices\_table.sql has “et\_al.”, apparently we are unable to recover it.

The dollar symbol, $, is in the vocabulary (global n. 529). It does not have a dummy sense… but it should, if our aim is to read every step of the text in a continuous manner.

Is ‘$’ in the indices\_table.sql? No, it’s not.

In PrepareKBInput.py,

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

**INSERT INTO indices\_table VALUES …**

and

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

and

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

This is a difference between the punctuation sign and “et\_al.”. The indices\_table.sql has et\_al..r.01, et\_al..r.02, and et\_al..dummySense.01 (that actually should not be there)

The vocabulary of globals contains both et\_al and et\_al.

I probably should modify the indices\_table. It contains already for.dummySense.01 and **et\_al.**.dummySense.01, but it should also have $.dummySense.01 and **et\_al**.dummySense.01.

'$' is part of the words\_without\_senses\_set. And so is ‘et\_al.’

Now, we find $.dummySense.01, [.dummySense.01 and so on, which is correct.

However, we have et\_al..dummySense.01, when there are already et\_al..r.01 and et\_al..r.02.

Probably because ‘et\_al.’ does not have a definition?

The definitions for .r.01 and .r.02 are both there… (adjusted)

INFO : Dummy senses, needing self-loops: 34089

**Attempt n. 3:**

INFO : Constructing X, matrix of node features

INFO : X\_definitions.shape=torch.Size([75718, 300])

INFO : X\_examples.shape=torch.Size([65964, 300])

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

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

INFO : Defining the edges: def, exs

INFO : def\_edges\_se.\_\_len\_\_()=37859

INFO : exs\_edges\_se.\_\_len\_\_()=32982

INFO : Defining the edges: sc

INFO : sc\_edges.\_\_len\_\_()=73706

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\_\_()=16504

INFO : Globals with no edges, needing self-loops: 0

INFO : []

INFO : Dummy senses, needing self-loops: 34170

INFO : [39536, 39537,…, 73704, 73705]

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=124380

INFO : Defining the edges: syn, ant

INFO : Inserted 5000 synonyms edges

INFO : Inserted 10000 synonyms edges

INFO : Inserted 15000 synonyms edges

INFO : Inserted 20000 synonyms edges

INFO : Inserted 25000 synonyms edges

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

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

Then, we create the grapharea\_matrix.

Until

INFO : node\_index=164000

everything is ok. Then,

INFO : node\_index=**164704**

INFO : adj\_nodes\_ls=[164704]

INFO : adj\_edge\_index=tensor([], device='cuda:0', dtype=torch.int64)

INFO : adj\_edge\_type=tensor([], device='cuda:0', dtype=torch.int64)

…

INFO : node\_index=268526

INFO : adj\_nodes\_ls=[268526]

INFO : adj\_edge\_index=tensor([], device='cuda:0', dtype=torch.int64)

INFO : adj\_edge\_type=tensor([], device='cuda:0', dtype=torch.int64)

Data(edge\_index=[2, 251061], edge\_type=[251061], node\_types=[268527], num\_relations=[1], x=[268527, 300])

Node intervals: 0, 73706, 73706+53139=126845, 126845+75718=202563, 202563+65964=268527.

Must review the process of graph and edge creation.

(or maybe it was because the indices\_table.sql was filled twice?)

**Attempt n.4:**

…

INFO : node\_index=164000

…

INFO : node\_index=164704

INFO : adj\_nodes\_ls=[164704]

INFO : adj\_edge\_index=tensor([], device='cuda:0', dtype=torch.int64)

…

INFO : node\_index=268526

INFO : adj\_nodes\_ls=[268526]

INFO : adj\_edge\_index=tensor([], device='cuda:0', dtype=torch.int64)

INFO : adj\_edge\_type=tensor([], device='cuda:0', dtype=torch.int64)

### Reviewing graph & edge creation

* globals\_vocabulary\_ls = globals\_vocabulary\_df[**'word'**].to\_list().copy()
* 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\_definitions.shape=torch.Size([75718, 300])  
  X\_examples.shape=torch.Size([65964, 300])  
  X\_globals.shape=torch.Size([53139, 300])   
  X\_senses.shape=torch.Size([73706, 300])
* initialize\_senses() iterates over the rows of the indicesTable\_db  
  if pos == 'dummySense':  
  # no definitions and examples, this is a dummy sense. It gets initialized with the global vector  
  else:  
  # average of definitions and examples
* X = torch.cat([X\_senses, X\_globals, X\_definitions, X\_examples])
* get\_edges\_elements() # definitions -> senses : [se+sp, se+sp+d) -> [0,se)  
   # examples --> senses : [se+sp+d, e==num\_nodes) -> [0,se)  
  Iterating over the db\_rows, as usual…
* if elements\_name==Utils.DEFINITIONS:  
   start\_sources = db\_row[2] + elements\_start\_index\_toadd  
   end\_sources = db\_row[3] + elements\_start\_index\_toadd
* edges\_toadd\_counter = edges\_toadd\_counter + (end\_sources-start\_sources)  
   for source in range(start\_sources, end\_sources):  
   edges\_ls.append((source, target\_idx))  
   indicesTable\_db.close()  
  return edges\_ls
* sc\_edges = get\_edges\_sensechildren(globals\_vocabulary\_df, X\_senses.shape[0])  
  iterating over the rows of the indices\_table.sql again  
  word = Utils.get\_word\_from\_sense(word\_sense)   
  sourceglobal\_raw\_idx = globals\_voc\_df.loc[globals\_voc\_df['word'] == word].index[0]  
  sourceglobal\_idx = globals\_start\_index\_toadd + sourceglobal\_raw\_idx  
  targetsense\_idx = db\_row[1]  
  edges\_ls.append((sourceglobal\_idx, targetsense\_idx))
* *# If operating on a sense-labeled corpus, we need to connect globals & their senses that do not belong to them*sc\_external\_edges = get\_additional\_edges\_sensechildren\_from\_slc(globals\_vocabulary\_df, globals\_start\_index\_toadd=X\_senses.shape[0])  
  sc\_edges.extend(sc\_external\_edges)
  + token\_dict = slc\_train\_corpus\_gen.\_\_next\_\_()  
    *# 1) Get the sense (and its index) specified in the SLC for the current token*  
    wordnet\_sense = try\_to\_get\_wordnet\_sense(wn30\_key)  
    *# 2) Get the global word of this token*lemmatized\_word = lemmatize\_term(word, lemmatizer)
* get\_edges\_selfloops(sc\_edges, num\_globals=X\_globals.shape[0], num\_dummysenses=num\_dummysenses)

These are the steps.

Now, what are the node intervals when we add a specific kind of edges, and are they correct compared to the graph?

The graph’s node intervals, again:

Node intervals: 0 – **senses** – 73706, 73706 – **globals** – 126845,   
126845 – **definitions** – 202563, 202563 – **examples** – 268527.

INFO : X\_definitions.shape=torch.Size([75718, 300])

INFO : X\_examples.shape=torch.Size([65964, 300])

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

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

INFO : Defining the edges: def, exs

# Definitions to senses

INFO : Min source node in edges-definitions = 126845

INFO : Max source node in edges-definitions = 164703

# 164703 – 126845 = 37858. I expected a domain [126845, 202563].

Possible error: the last row of that part is:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| word\_sense | vocab\_index | start\_defs | end\_defs | start\_examples | end\_examples |
| zoologist.n.01 | 37858 | 37858 | 37859 | 32982 | 32982 |

*Must review the creation of the indices\_table.sql.*

INFO : Min target node in edges-definitions = 0

INFO : Max target node in edges-definitions = 37858

# the 1st dummySense is found at 37859, so this is ok. Range. -> [0, senses that aren’t dummySenses]

INFO : def\_edges\_se.\_\_len\_\_()=37859

INFO : Min source node in edges-examples = 202563

INFO : Max source node in edges-examples = 235544

# Another mismatch. I expected a domain of [202563, 268527](**examples**)

INFO : Min target node in edges-examples = 1

INFO : Max target node in edges-examples = 37849

#This is ok, because we do not expect *all* the senses (senses with data: 0-37858, as we recall) to have examples.

INFO : exs\_edges\_se.\_\_len\_\_()=32982

INFO : Defining the edges: sc

INFO : Min source node in edges-get\_edges\_sensechildren = 73706

INFO : Max source node in edges-get\_edges\_sensechildren = 126844

# 73706 – **globals** – 126845, ok

INFO : Min target node in edges-get\_edges\_sensechildren = 0

INFO : Max target node in edges-get\_edges\_sensechildren = 73705

# 0 – **senses** – 73706

INFO : sc\_edges.\_\_len\_\_()=73706

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

# Before reviewing the process of graph creation, I will just re-execute the pipeline from the start. Based on the semcor.xml dataset? It’s more appropriate to use a **part** of semcor.xml for debugging, and then restart on the whole dataset.

### Operating on a part of SemCor

INFO : The corpus at TextCorpuses/My Sense-Labeled Corpus/semcor.xml has 26 documents

INFO : Training dataset will contain: 20 documents , Validation dataset will contain: 3 documents , Test dataset will contain: 3 documents

INFO : \*\*\* The vocabulary was created. Number of words= 10301\*\*\*

INFO : CreateGraphInput.exe() > number of ords included in the vocabulary chunk, to be prepared: 10268

**The Graph**

INFO : X\_**senses**.shape=torch.Size([20916, 300])

INFO : X\_**globals**.shape=torch.Size([10301, 300])

INFO : X\_**definitions**.shape=torch.Size([15409, 300])

INFO : X\_**examples**.shape=torch.Size([17689, 300])

Intervals: 0, 20916, 31217, 46626, 64315.

Where we have 15409 senses with data, and the remaining 5507 are dummySenses

INFO : Defining the edges: def, exs

# Edges-definitions: from definitions to senses: [31217, 46625] -> [0, 15408]

INFO : def\_edges\_se.\_\_len\_\_()=15409

From the definitions’ interval to the senses-with-data.

Edges-examples: from examples to senses:

[46626, 64314] -> [0, 15408]

INFO : Min target node in edges-examples = 0

INFO : Max target node in edges-examples = 15408

INFO : exs\_edges\_se.\_\_len\_\_()=17689

Edges-senseChildren, from globals to senses:

[20916, 31216] -> [0, 20915]

INFO : sc\_edges.\_\_len\_\_()=20916

From the globals’ interval to the senses (all of them, including dummySenses).

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

get\_additional\_edges\_sensechildren\_from\_slc : [20917, 31206] -> [0, 15381]

INFO : sc\_edges\_with\_external.\_\_len\_\_()=2610

INFO : get\_edges\_selfloops>max\_sense=20915

[20928, 28163] -> [20928, 28163]

INFO : [20928, 20935, 20944, 20975, 21177, 21195, 21360, 21445, 21477, 21598, 21599, 21624, 21825, 22683, 22756, 23024, 23301, 24238, 24710, 25539, 25540, 27889, 27890, 28136, 28163]

corresponding to the globals:

['"', '.', ',', 'a', '1', ':', '-', '$', 'A', '(', ')', 'I', "'", 'D', ';', '0', '!', '?', 'C', '[', ']', 'S', 'P', 'O', '/']

INFO : sc\_edges\_with\_selfloops.\_\_len\_\_()=23551

Apparently, single characters did not manage to have a dummySense associated with it.

We have the dollar sign ‘$’ in the vocabulary of globals. As well as ‘[‘, etc.

This time, we have $.dummySense.01 in the indices\_table.sql

Let us try recreating the graph.

We attempt the insertion

INFO : dummy\_wn\_id=$.dummySense.01 , and also

A.dummySense.01, etc.

We have A.dummySense.01 in the indices\_table, and also !.dummySense.01, $.dummySense.01, etc.

When creating the graph, we attempt to establish:

INFO : get\_edges\_sensechildren>word=$

INFO : get\_edges\_sensechildren>word=!

Finally, we get:

INFO : get\_edges\_selfloops>max\_sense=20948

INFO : []

INFO : len(edges\_ls)==0

Therefore, it is now possible to run the pipeline on the whole of SemCor.

### On the entirety of SemCor, new statistics

The fault of the single characters not having a dummySense associated with them may be due to the vocabulary\_ls passed on manually to the create\_senses\_indices\_table() function.

For now, we reload the vocabulary and re-create the graph.

The list from get\_edges\_selfloops should be empty.

Maybe because in the RetrieveInputData when I access WordNet, I have

*# do not retrieve dictionary data for punctuation symbols, e.g. '==', '(' etc.*

This can be fixed by re-loading the vocabulary when creating the SQL table.

When computing the embeddings of the elements (definitions and examples, in order):

INFO : ComputeEmbeddings > embds\_nparray.shape=(37859, 300)

INFO : ComputeEmbeddings > embds\_nparray.shape=(32982, 300)

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

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

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

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

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

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

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

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

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

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

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

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

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

get\_edges\_selfloops:

INFO : []

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

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

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

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

### Graph retrieval

Node n: 73707.

Adjacent nodes:

|  |  |  |
| --- | --- | --- |
| 73707 | globals (2nd) |  |
| 15606 | senses | group.n.01 |
| 38358 | senses | Fulton\_County\_Grand\_Jury.dummySense.01 |

The rest of the check of the graph retrieval is ok, but this highlights a problem:

I should not add dummySenses if I am adding senses that are based on the corpus.

The root of the problem is \*not\* in the creation of the graph, but in the fact that we first create the dummySenses for those globals without WordNet senses, and then read the Sense-Labeled Corpus to add senses that are associated there.

*However*, is this really a problem?

It is not guaranteed that a given global (e.g. ‘'Oxford'’ , ‘l'Osservatore\_Romano’ , etc.) will always be associated with that particular sense (e.g. ‘group.n.01’). What if we encounter an occurrence of the word – word which has no data in WordNet – that does not have that particular sense?

Therefore, it is opportune to leave it as it is.

Checking other nodes:

Node n. 50.

|  |  |  |  |
| --- | --- | --- | --- |
| 50 | senses | ablaze.s.03 |  |
| 126895 | definitions | 50th def: ablaze.s.03 | resembling flame in brilliance or color |
| 164762 | examples | 58th example (ablaze.s.03) | maple trees ablaze in autumn |
| 107209 | globals | 33503rd global | ablaze |

Node n. 107209

|  |  |  |  |
| --- | --- | --- | --- |
| 107209 | globals | 33503rd global | ablaze |
| 48 | senses | ablaze.s.01 |  |
| 49 | senses | ablaze.s.02 |  |
| 50 | senses | ablaze.s.03 |  |
| 51 | senses | ablaze.s.04 |  |
| 94982 | globals | 21276th global | aflame |
| 109031 | globals | 35325 | arouse |
| 115216 | globals | 41510 | alight |
| 122637 | globals | 48931 | inflame |
| 126073 | globals | 52367 | on\_fire |

Node n. 15606

|  |  |  |  |
| --- | --- | --- | --- |
| 15606 | senses |  |  |
| 74017 | globals (lower boundary at 73706) | 1038 |  |
| 142451 | definition, lower boundary 126845 | 15606th : group.n.01 | any number of entities (members) considered as a unit |

Inside the function, I find also 73707 (Fulton\_County\_Grand\_Jury), the 2nd global (index 1 if we start from 0)…

I lose it because the list of neighbouring nodes would have len>32 otherwise, and we stated that 32 is the maximum area size.

Globals:

(311, 'Grady\_Hospital'),

(1038, 'State\_Health\_Department'),

(1946, 'Knights\_of\_Columbus'),

(2107, 'Anson'),

(2209, 'Rice\_University'),

(3120, 'Continental'),

(3597, 'Tipoff\_Club'),

(7182, 'Oxford'),

(9741, 'BBB'),

(9879, 'Division\_of\_Public\_Information'),

(11229, 'Small\_Business\_Administration\_Regional\_Offices'),

(11305, 'Banks\_for\_Cooperatives'),

(11331, 'U.S.\_Department\_of\_the\_Interior'),

(13506, 'Brookfield\_Zoo'),

(15325, 'Council\_of\_Ministers'),

(17030, 'Polaroid'),

(17115, 'Central'),

(18046, 'Kent\_House'),

(18091, 'Trustee\_Board'),

(18121, 'Old\_Clubhouse'),

(20139, 'Santo\_Spirito'),

(23174, "l'Osservatore\_Romano"),

(24635, 'Bronx\_Zoo'),

(25844, 'A\_+\_S'),

(25847, 'Joseph\_Horne'),

(28007, 'Smith-Hughes'),

(29253, 'American\_Assembly\_of\_Columbia\_University'),

(29433, 'Spanish\_Inquisition'),

(30465, 'C.'),

(30686, 'Structural\_Clay\_Products\_Institute')]

## Perplexity

It is our objective to compute 2 perplexities for the GRU\_senses: one for all the words in the document, and another only for the words that have multiple senses.

I must keep a list of a) indices of words with >1 sense b) indices of words with only 1 sense.

This can be created at the time of the training\_setup()

### Globals with =/>1 senses

**for** idx **in** range(last\_idx\_senses, last\_idx\_senses+last\_idx\_globals)

# idx=73796, …, etc.

ith\_global\_row = grapharea\_matrix[idx]

I should operate on

ith\_global\_row.toarray()[0][grapharea\_size+2\*max\_edges:]

…

With a first version,

INFO : len(words\_1\_sense)=53139

INFO : len(words\_multiple\_senses)=0

Note:

As we saw above, some words have both a dummySense and a sense from the SLCorpus. Should they be counted as multi-sense? The reasonable answer is no.

How to find out?

If a number of the adjacent nodes (see the first part, 0:grapharea\_size of the row in the grapharea\_matrix) are in the range of the dummySenses.

How to determine the range of the dummySenses?

By reading the indices\_table.sql . It could be added in Utils.

If I counted them as multi-sense, I would be choosing between the dummySense and the SLCorpus-specified sense.

Note:

Remember that when extracting data from the grapharea\_matrix, in

**def** get\_node\_data(grapharea\_matrix, i, grapharea\_size, features\_mask=(**True**,**True**,**True**))

we do:

list(map( **lambda** value: value-1,…

operating with: filter-on-nonzero > subtract 1.

This is also due to the fact that numpy sparse matrices only work with 0 as base value.

Therefore, when we operate on the grapharea\_matrix rows now, with the aim of determining the globals with =1 / >1 sense, it is necessary to subtract 1 from the values, after a numpy.nonzero filter. This is highlighted by the fact that, without modifications, the edge\_type-s are 1,…,5, whereas they should be 0,…,4.

Note: When determining the number of senses of a global, we must lemmatize the token first.

I can exploit

get\_node\_data(grapharea\_matrix, i, grapharea\_size, features\_mask=(**True**,**True**,**True**))

that is already found in Adjacencies.py

The presence of the dummySenses means we have to change one of the checks in the lemmatization. Namely,

*# if a word has edges that are not all self-loops, do not lemmatize it (to avoid turning 'as' into 'a')***if not**(all([src\_dest\_tpl[0]==src\_dest\_tpl[1] **for** src\_dest\_tpl **in** edge\_index.t()])):  
 logging.debug(**"word has edges that are not all self-loops"**)  
 **return** x\_indices, edge\_index, edge\_type

now every node has edges that are not self-loops (the connection to the dummySense).

The check must become:

*if the neighbouring nodes are* ***not*** *all* ***dummySenses****, do not lemmatize it*

This introduces the need for the startpoint of the dummySenses as a parameter of the lemmatization function.

Eventually, we find:

INFO : len(words\_1\_sense)=38297

INFO : len(words\_multiple\_senses)=14842

Later on, in order to save time in the training\_setup, we could add a column tot\_num\_senses in the vocabulary’s H5 archive, to contain the information ‘num\_senses’.

A separate getter function then reads the vocabulary,

multisense\_globals\_indices = AD.get\_multisense\_globals\_indices()

obtaining

len(multisense\_globals\_indices)= 14842 (as expected)

### Loss computation and perplexity

We are currently using a dictionary for the number of correct and top-k labels:

correct\_predictions\_dict = {**'correct\_g'**:0,  
 **'top\_k\_g'**:0,  
 **'tot\_g'**:0,  
 **'correct\_s'**:0,  
 **'top\_k\_s'**:0,  
 **'tot\_s'**:0}

And lists-of-tuples of loss values for each epoch:

training\_losses\_lts = [] *# mutated into a lts, with (global\_loss, sense\_loss)*validation\_losses\_lts = []

We have to use 2 sense losses & statistics: for **'correct\_all\_s'** and for **'correct\_multi\_s'**

# Multi-Sense LM, part 4

## Architectures

We review the architecture of the GRU (and later, of the LSTM), based on past papers as sources.

(Unrelated note: I haven’t even tried the DistilBERT-derived embeddings. Every experiment so far has used the FastText embeddings. Wouldn’t it be good to have the other embeddings as a comparison/reference point?)

|  |  |  |  |
| --- | --- | --- | --- |
| Source | Dataset | Architecture | PPL |
| Restricted Recurrent Neural Networks by E.Diao et al., 2019 | PTB, WikiText-2 | Batch size = 80 BPTT seq\_len = 35  3 layers, with hidden\_units=200 and  embedding\_size=200 | RNN (Restricted RNN with s=0, so standard RNN):  Test=230.8, Valid=253.1  (R)GRU: Test=167.5, Valid=181.8  (R)LSTM: Test=**154.5**, Valid=**167.6** |
| An Analysis of the Utility of Explicit Negative Examples  to Improve the Syntactic Abilities of Neural Language Models by Noji & Takamura, 2020 | *From English Wikipedia*, 80M/10M/10M  Vocabulary trimmed to 50K with <unk> | 1) 3 layers LSTM,  hidden\_units=1150, d\_embeddings=400, input and output embeddings are tied. 2) From Marvin&Linzen(2018), 2 layers with d\_emb=d\_hidden=650 and  Dropout on word embeddings and the output of every layer. | M&L18: 78.6  LSTM-LM: 49.5 |
| Regularizing and Optimizing LSTM Language Models by S.Merity et al., 2017 | PTB, WikiText-2 | **AWD-LSTM**  d\_emb=400, layers=1150>1150>400(tied)  Variable-length BPTT: p=0.95\*N(70,s=5) <+> (0.05)\*N(35,5)  DropConnect: w=0.4, 1L,2L=0.3, 3L=0.4, e=0.1  (Temporal) Activation Regularization  Averaged SGD | WT-2,  Validation=**68.6**, Test=**65.8** |
| **~~Improving Neural Language Models with a Continuous Cache~~**, by E.Grave et al., 2016 | WikiText-2, et al. | The LSTM has: hidden\_units=1024 Dropout=0.65 (very high)  batch\_size=20 BPTT len=30 steps  AdaGrad (…),  with clipped gradient norm at <=0.1 | Validation=**104.2**  Test=**99.3** |
| On the State of the Art of Evaluation in Neural Language Models by G.Melis et al. 2017 | WikiText-2, et al. | batch size=64  BPTT len=35  Adam optimizer  *No specification of hidden units,* uses parameter budget of 10M or 24M | (24M)  best: 2 layers with 24M,  Validation=69.1, Test=65.9 |

## LSTM, version 0

Let us apply the LSTM on the SemCor dataset. 2 LSTMs, one for globals, one for senses.

The main modification compared to previous experiments is that now we operate with dummySenses, that allow for a continuous reading. The hyperparameters are also modified.

Later on, we also examine the impact of adding the node-state of the current global from the dictionary graph.

The point of comparison for this is [Baseline 0: 2 GRUs](#_Baseline_0:_2), that had a Validation Perplexity on SemCor globals in the range 185-188 (although it operated on the reduced vocabulary of 21988 globals).

### Debug

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

Debugging the double Perplexity and applying a Time Analysis on the compute\_model\_loss(…) function…

t1 - t0 = 0.09125

t2 - t1 = 1.15813

t3 - t2 = 0.00012

t4 - t3 = 0.0022

t5 - t4 = 0.00013

-> After turning the list multisense\_globals\_ls into a set, for faster membership check:

t1 - t0 = 0.05659

t2 - t1 = 9e-05

t3 - t2 = 0.00014

t4 - t3 = 0.06355

t5 - t4 = 5e-05

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

batch\_labels\_all\_senses=tensor([15606, 32203, -1, -1, 25908, -1])

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

multisense\_globals\_set=<class 'set'>: {2, 5, 32774, 8, 9, 11,...

Adjusted bug:

batch\_labels\_multi\_senses\_ls = list(map(  
 **lambda** i : batch\_labels\_all\_senses[i] **if** batch\_labels\_globals[i]**.item()** **in** multisense\_globals\_set  
 **else** -1, range(len(batch\_labels\_all\_senses))))

### 2LSTMs: Model

INFO : Model:

INFO : DataParallel(

(module): RNN(

(dropout): Dropout(p=0.65, inplace=False)

(main\_rnn\_ls): ModuleList(

(0): LSTM(300, 1024) )

(senses\_rnn\_ls): ModuleList(

(0): LSTM(300, 1024) )

(linear2global): Linear(in\_features=1024, out\_features=53139, bias=True)

(linear2senses): Linear(in\_features=1024, out\_features=73706, bias=True) ))

INFO : Parameters:

INFO : ('module.X', torch.Size([197686, 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([1, 2, 1024]), torch.float32, False)

('module.memory\_cn', torch.Size([1, 2, 1024]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([1, 2, 1024]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([1, 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.senses\_rnn\_ls.0.weight\_ih\_l0',torch.Size([4096, 300]), torch.float32,True)

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

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

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

('module.linear2global.weight', torch.Size([53139, 1024]), torch.float32, True)

('module.linear2global.bias', torch.Size([53139]), torch.float32, True)

('module.linear2senses.weight', torch.Size([73706, 1024]), torch.float32, True)

('module.linear2senses.bias', torch.Size([73706]), torch.float32, True)

INFO : Number of trainable parameters=200184517, i.e. 200.18M, where:

embeddings=59305800= 59.30M

softmax= 54.47 (globals) + 75.55 (senses) =… 130.01M. embeddings+softmax= 189.32M

core=10.86M (other networks had numbers like 27, 28, 33, 36M core parameters

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training* | | | | | | | | |
|  | *Globals* | | | *Senses* | | | *Senses (of words with multiple senses)* | | |
|  | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* |
| 1 | 3131.35 | 33210 /646200 | 170458 /646200 | 16011.02 | 7220 /196363 | 18602 /196363 | 1478.37? | 4050 /147706 | 10812 /147706 |
| 2 | 1240.06 | 53335 | 196110 | 5225.57 | 8539 /196353 | 23458 /196353 | 634.88? (must check computation) | 4942 /147678 | 13505 /147678 |
| 3 | 1106.41 | 64234 | 198815 | 4785.79 | 8667 | 23663 | (594.13) | 5141 | 13611 |
| 5 | 932.06 | 77098 | 208419 | 4528.98 | 8656 | 23814 | (572.7) | 5082 /147695 | 13629 /147695 |
| 10 | 705.96 | 95627 | 224126 | 4212.46 | 8929 | 24019 | (546.97) | 4795 | 13648 |
| 15 | 589.28 | 102861 | 237579 | 3880.75 | 8740 | 24716 | (518.03) | 4399 | 13775 |
| 53 | 280.81 |  |  | 2743.87 |  |  | 403.91 |  |  |

### 2LSTMs: Experiment 0 on SemCor

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with 1 layer (1024) | 1) The word embedding of the current global (d=300) | batch\_size=2  TBPTT length=3 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
| **Senses LSTM** (L=1, d=1024) into linear2Senses FF-NN | learning rate=10^(-5) |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Validation* | | | | | | | | |
|  | *Globals* | | | *Senses* | | | *Senses (of words with multiple senses)* | | |
|  | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* |
| 1 | 1423.89 | 7689 /81000 | 24732 /81000 | 2013.07 | 963 /8121 | 1882 /8121 | 1706.88 | 760 /6503 | 1574 /6503 |
| 2 | 1380.97 | 7678 | 24738 | 1827.22 | 961 /8117 | 1933 /8117 | 1548.84 | 759 /6499 | 1579 /6499 |
| 3 | 1326.75 |  |  | 1775.05 |  |  | 1503.36 |  |  |
| 5 | 1233.1 |  |  | 1719.95 |  |  | 1451.59 |  |  |
| 10 | 1146.58 |  |  | 1697.99 |  |  | 1425.78 |  |  |
| 15 | 1093.65 |  |  | 2265.61 |  |  | 1943.42 |  |  |
| 53 | 793.59 |  |  | 2907.89 |  |  | 2502.12 |  |  |

### Observations

* The chosen architecture (LSTM: 1024 -> Linear2Logits, w/ dropout=0.65) is unable to learn even the globals. In spite of having chosen the architecture from the Neural Cache paper by Grave et. al. (2016), it appears to be useless. There are several possible reasons for it.

1. I did not check the predictions on a mini-experiment before starting an experiment The sense labels are still the old ones, with -1, and that is wrong. We should be able to extract the dummySenses in the DataLoading.
2. The original paper uses:  
   “**Implementation details.**

We train recurrent neural network language models with 1024 LSTM units,

regularized with dropout (probability of dropping out units equals to 0.65).

We use the **Adagrad** algorithm, with a learning rate of **0.2**, a batchsize of 20 and initial weight uniformly sampled in the range [-0.05; 0.05].

We clip the norm of the gradient to **0.1** and unroll the network for 30 steps.”

## Check predictions & labels, Parameters & Debug

### Sense labels

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

batch\_labels\_all\_senses = tensor([15606, 32203, -1, -1, 25908, -1])

We have dummySenses, so the labels are wrong. We must review the process of DataLoading.

To get the label: NumericalIndices. convert\_tokendict\_to\_tpl():

* examines the token\_dict,
* from the token\_dict extracts the wordnet sense with try\_to\_get\_wordnet\_sense(wn30\_key),
* then: we query the database, with:  
  query = "SELECT vocab\_index FROM indices\_table " + "WHERE word\_sense='" + wordnet\_sense + "'" (e.g.: 'SELECT vocab\_index FROM indices\_table WHERE word\_sense=\'state.v.01\'')

Which means: if there is no specified label (e.g. “for”, “of” etc.) we are not able to find the dummySense.

We must modify:

**if 'wn30\_key' in** keys:

…

**if** sense\_index\_queryresult **is None**: *# there was no sense-key, or we   
 did not find the sense for the key* sense\_index = -1

We should:

use the graph. Check if the current global has a sense connection (if it did not have a sense/a valid sense was not found, it will be a dummySense). Use the dummySense node as the label…

After version 1.0 of the new retrieval has been done:

batch\_labels\_globals= tensor([1, 2, 3, 4, 5, 6]) (unchanged)

batch\_labels\_all\_senses= tensor([15606, 32203, 44429, 50085, 25908, 50376])

batch\_labels\_multi\_senses = tensor([ -1, 32203, -1, -1, 25908, -1])

Verifying the senses:

The

|  |  |  |
| --- | --- | --- |
| 15606 | group.n.01 | Fulton\_County\_Grand\_Jury |
| 32203 | **state.v.01** | **said** |
| 44429 | Friday.dummySense.01 | Friday |
| 50085 | an.dummySense.01 | an |
| 25908 | **probe.n.01** | **investigation** |
| 50376 | of.dummySense.01 | of |

…

This time it seems to be acceptable. The only potential point of contention is whether we should define the “multi”senses (senses of words that have multiple senses we have to choose from) from the global or from the sense. We can just keep using the global for that.

## LSTM, version 1

### 2LSTMs, Mini-Experiment 1, overfit on fragment of SemCor

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with 1 layer (1024) | 1) The word embedding of the current global (d=300) | batch\_size=2  TBPTT length=3 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
| **Senses LSTM** (L=1, d=1024) into linear2Senses FF-NN | learning rate=10^(-3) |
| optimizer: Adam. gradient clipping at 0.1 |

INFO : Hyperparameters: \_batchPerSeqlen6\_area32\_lr0.001\_epochs300

INFO : Model:

INFO : DataParallel(

(module): RNN(

(dropout): Dropout(p=0.65, inplace=False)

(main\_rnn\_ls): ModuleList(

(0): LSTM(300, 1024))

(senses\_rnn\_ls): ModuleList(

(0): LSTM(300, 1024))

(linear2global): Linear(in\_features=1024, out\_features=53139, bias=True)

(linear2senses): Linear(in\_features=1024, out\_features=73706, bias=True)))

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training* | | | | | | | | | |
|  | *Globals* | | | | *Senses* | | | | | |
|  | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | | *Correct / total* | *Top-10 / total* | *PPL (multi)* | *Correct / total (multi)* | *Top-10 / total (multi)* |
| 1 | 51680.9 | 2 / 66 | 6 / 66 | 70615.72 | | 2 / 66 | 10 / 66 | 96.57 | 1 / 27 | 3 / 27 |
| 2 | 4595.97 | 8 | 27 | 5027.58 | | 10 | 27 | 50.56 | 1 | 5 |
| 3 | 259.92 |  |  | 288.02 | |  |  | 24.52 |  |  |
| 5 | 54.71 |  |  | 54.09 | |  |  | 7.69 |  |  |
| 10 | 30.65 |  |  | 38.75 | |  |  | 5.83 |  |  |
| 20 | 23.96 |  |  | 28.8 | |  |  | 5.03 |  |  |
| 30 | 17.81 |  |  | 20.89 | |  |  | 3.98 |  |  |
| 50 | 2.36 |  |  | 2.17 | |  |  | 1.46 |  |  |
| 75 | 1.04 | 65 | 66 | 1.06 | | 66 | 66 | 1.04 | 28/29 | 29/29 |

### 2LSTMs: Experiment 1 on SemCor

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Globals’ prediction: **Main LSTM** with 1 layer (1024) | 1) The word embedding of the current global (d=300) | batch\_size=20  TBPTT length=30 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
| **Senses LSTM** (L=1, d=1024) into linear2Senses FF-NN | learning rate=~~0.2~~ 10^(-4)  optimizer: Adam.  ~~gradient clipping at 0.1.~~  Dropout, p=0.65 |

(The current insertion of gradient clipping causes mistakes in the loss computation. Scrapping, otherwise I would have to review it)

INFO : Model:

INFO : DataParallel(

(module): RNN(

(dropout): Dropout(p=0.65, inplace=False)

(main\_rnn\_ls): ModuleList(

(0): LSTM(300, 1024)

)

(senses\_rnn\_ls): ModuleList(

(0): LSTM(300, 1024)

)

(linear2global): Linear(in\_features=1024, out\_features=53139, bias=True)

(linear2senses): Linear(in\_features=1024, out\_features=73706, bias=True)

)

)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training* | | | | | | | | |
|  | *Globals* | | | *Senses* | | | *Senses (of words with multiple senses)* | | |
|  | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* |
| 1 | 1204.09 | 73656 /646200 | 205985 /646200 | 1733.67 | 33720 /646200 | 197234 /646200 | 35.16(?) | 5 /243482 | 10130 /243482 |
| 2 | 611.37 |  |  | 1152.74 |  |  | 28.65 |  |  |
| 3 | 483.92 |  |  | 1088.83 |  |  | 27.85 |  |  |
| 4 | 414.42 |  |  | 1046.86 |  |  | 27.31 |  |  |
| 5 | 366.72 |  |  | 1015.32 |  |  | 26.88 |  |  |
| 6 | 330.94 |  |  | 985.38 |  |  | 26.52 |  |  |
| 7 | 301.14 |  |  | 956.94 |  |  | 26.12 |  |  |
| 8 | 276.76 |  |  | 930.17 |  |  | 25.8 |  |  |
| … |  |  |  |  |  |  |  |  |  |
| 46 | 36.03 |  |  | 295.9 |  |  | 14.3 |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Validation* | | | | | | | | |
|  | *Globals* | | | *Senses* | | | *Senses (of words with multiple senses)* | | |
|  | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* |
| 1 | 898.15 | 11712 /81000 | 27428 /81000 | **1775.81** | 4259 /81000 | 24921 /81000 | 11219.03 | 0 /31151 | 1899 /31151 |
| 2 | 744.53 |  |  | 1822.77 |  |  | **11197.02** |  |  |
| 3 | 688.39 |  |  | 1906.73 |  |  | 11523.88 |  |  |
| 4 | 666.56 |  |  | 2032.61 |  |  | 12849.4 |  |  |
| 5 | 662.56 |  |  | 2188.04 |  |  | 14508.6 |  |  |
| 6 | **655.76** |  |  | 2340.68 |  |  | 16033.88 |  |  |
| 7 | 661.56 |  |  | 2488.49 |  |  | 17355.67 |  |  |
| 8 | 674.6 |  |  | 2647.02 |  |  | 18515.3 |  |  |
| … |  |  |  |  |  |  |  |  |  |
| 46 | 1699.21 |  |  | 8280.35 |  |  | 70343.86 |  |  |

## LSTM, version 2

### 2LSTMs, Mini-Experiment 2, overfit on fragment of SemCor

|  |  |  |
| --- | --- | --- |
| **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=2  TBPTT length=3 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
| **Senses LSTM** (L=3, d=1150>1150>400) into linear2Senses FF-NN | learning rate=10^(-3) |
| optimizer: Adam. |

INFO : Hyperparameters:

INFO : Model: INFO : 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)

)

(senses\_rnn\_ls): ModuleList(

(0): LSTM(300, 1150)

(1): LSTM(1150, 1150)

(2): LSTM(1150, 400)

)

(linear2global): Linear(in\_features=400, out\_features=53139, bias=True)

(linear2senses): Linear(in\_features=400, out\_features=73706, bias=True)

)

)

INFO : Parameters:

INFO : ('module.X', torch.Size([197686, 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, 40, 1150]), torch.float32, False)

('module.memory\_cn', torch.Size([3, 40, 1150]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([3, 40, 1150]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([3, 40, 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.senses\_rnn\_ls.0.weight\_ih\_l0', torch.Size([4600, 300]), torch.float32, True)

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

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

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

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

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

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

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

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

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

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

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

('module.linear2global.weight', torch.Size([53139, 400]), torch.float32, True)

('module.linear2global.bias', torch.Size([53139]), torch.float32, True)

('module.linear2senses.weight', torch.Size([73706, 400]), torch.float32, True)

('module.linear2senses.bias', torch.Size([73706]), torch.float32, True)

INFO : Number of trainable parameters=149.67M, where

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training* | | | | | | | | | |
|  | *Globals* | | | | | *Senses* | | | | |
| 1 | 38541.07 | 2 / 66 | 6 / 66 | 54460.17 | 2 / 66 | | 10 / 66 | 91.98 | 1 / 27 | 3 / 27 |
| 2 | 848.03 |  |  | 1393.31 |  | |  | 25.27 |  |  |
| 3 | 71.72 |  |  | 74.26 |  | |  | 8.69 |  |  |
| 5 | 36.79 |  |  | 37.09 |  | |  | 5.47 |  |  |
| 10 | 27.95 |  |  | 27.73 |  | |  | 5.0 |  |  |
| 30 | 19.8 |  |  | 20.03 |  | |  | 4.02 |  |  |
| 50 | 18.88 |  |  | 19.36 |  | |  | 3.99 |  |  |
| 75 | 16.57 |  |  | 16.61 |  | |  | 3.75 |  |  |
| 100 | 5.43 |  |  | 4.15 |  | |  | 2.13 |  |  |
| 125 | 1.1 |  |  | 1.1 |  | |  | 1.05 |  |  |

### 2LSTMs: Experiment 2 on SemCor

|  |  |  |
| --- | --- | --- |
| **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=5\*10^(-5) |
|  |

INFO : Hyperparameters: \_batchPerSeqlen1400\_area32\_lr5e-05\_epochs300

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training* | | | | | | | | | *Validation* | | | | | | | | |
|  | *Globals* | | | *Senses* | | | *Senses (of words with multiple senses)* | | | *Globals* | | | *Senses* | | | *Senses (of words with multiple senses)* | | |
|  | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* | *PPL* | *Correct / total* | *Top-10 / total* |
| 1 | 2384.19 |  |  | 2793.12 |  |  | 38.47 |  |  | 1490.12 |  |  | 1886.82 |  |  | 13569.56 |  |  |
| 2 | 1209.49 |  |  | 1234.63 |  |  | 31.15 |  |  | 1557.92 |  |  | **1856.19** | 4213 /81200 | 24737 /81200 | 11883.82 | 0 / | 1175 / |
| 3 | 1186.57 |  |  | 1198.5 |  |  | 30.66 |  |  | 1617.78 |  |  | 1898.59 |  |  | 11881.51 |  |  |
| 4 | 1178.66 |  |  | 1188.3 |  |  | 30.56 |  |  | 1660.72 |  |  | 1940.25 |  |  | 11871.85 |  |  |
| 5 | 1174.36 |  |  | 1182.62 |  |  | 30.52 |  |  | 1705.98 |  |  | 1995.05 |  |  | 12179.13 |  |  |
| 6 | 1167.33 |  |  | 1178.98 |  |  | 30.49 |  |  | 1650.05 |  |  | 2037.15 |  |  | 12373.67 |  |  |
| 7 | 931.49 |  |  | 1177.18 |  |  | 30.47 |  |  | 1204.33 |  |  | 2080.41 |  |  | 12471.89 |  |  |
| 8 | 763.28 |  |  | 1174.76 |  |  | 30.43 |  |  | 1109.38 |  |  | 2110.91 |  |  | 12614.54 |  |  |
| 9 | 689.09 |  |  | 1173.29 |  |  | 30.42 |  |  | **1067.93** | 11815/ 81200 | 27646 /81200 | 2146.89 |  |  | 12902.05 |  |  |
| 10 | 639.55 |  |  | 1170.56 |  |  | 30.37 |  |  | 1490.12 |  |  | 1886.82 |  |  | 13569.56 |  |  |

Given the disappointing experiment on SemCor, we return to the vocabulary of ~25K, obtained with min\_freq=2. Is it easier to handle? After all, WikiText-2 has <unk> tokens, so there is an amount of pre-processing, and the vocabulary is thus “trimmed” to a reasonable size of 33K.

## LSTM, version 3

This new experiment is influenced by the following observation:

While the *code* does not modify WikiText-2, appearing to take in all every word, the *text* itself is pre-processed. E.g.:

“Parachute competitions are held at club , regional , national and international levels , and include the disciplines of accuracy landings , <unk> <unk> , formation <unk> , canopy formation , freestyle and <unk> , and <unk> . British teams consistently win medals in canopy formation world championships , and a British team took the 2006 world championship in women 's 4 @-@ way formation <unk> .”

The presence of the <unk> tokens allows WT-2 to have a vocabulary of reasonable size, 33,278.

Therefore, since the last experiment on the full-SemCor vocabulary of 55+K had significantly worse perplexity results compared to the previous ones with min\_freq=2 and |V|≈21K, we return to having min\_freq=2, while still leaving out some other steps of pre-processing like the conversion into <NUM>s.

### 2LSTMs: Experiment 3 on SemCor

INFO : Model: INFO : 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) )

(senses\_rnn\_ls): ModuleList(

(0): LSTM(300, 1150)

(1): LSTM(1150, 1150)

(2): LSTM(1150, 400)

)

(linear2global): Linear(in\_features=400, out\_features=25693, bias=True)

(linear2senses): Linear(in\_features=400, out\_features=43559, bias=True) ))

INFO : Parameters:

INFO : ('module.X', torch.Size([126462, 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, 40, 1150]), torch.float32, False)

('module.memory\_cn', torch.Size([3, 40, 1150]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([3, 40, 1150]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([3, 40, 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.senses\_rnn\_ls.0.weight\_ih\_l0', torch.Size([4600, 300]), torch.float32, True)

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

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

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

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

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

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

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

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

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

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

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

('module.linear2global.weight', torch.Size([25693, 400]), torch.float32, True)

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

('module.linear2senses.weight', torch.Size([43559, 400]), torch.float32, True)

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

INFO : Number of trainable parameters=105.212M, where

embeddings = 37.939M

softmax = 10.303M (globals) + 17.467M (senses) = 27.77M

core = 39.503M

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Epoch* | *Training PPL, globals* | *Training PPL, senses* | *Training PPL,*  *multi-senses* | *Validation PPL, globals* | *Validation PPL, senses* | *Validation PPL, multi-senses* |
| 1 | 1387.84 | 1798.57 |  | 900.8 | 1140.57 |  |
| 2 | 801.29 | 904.14 |  | 917.52 | 1112.46 |  |
| 3 | 790.34 | 885.25 |  | 941.96 | 1135.62 |  |
| 4 | 787.45 | 880.07 |  | 952.23 | 1146.91 |  |
| 5 | 784.91 | 876.95 |  | 957.2 | 1157.18 |  |
| 6 |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |
| 17 |  |  |  |  |  |  |
| 18 |  |  |  |  |  |  |
| 19 |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

# 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 : 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=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([2, 4, 1024]), torch.float32, False)

('module.memory\_cn', torch.Size([2, 4, 1024]), torch.float32, False)

('module.memory\_hn\_senses', torch.Size([2, 4, 1024]), torch.float32, False)

('module.memory\_cn\_senses', torch.Size([2, 4, 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([33278, 1024]), torch.float32, True)

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

INFO : Number of trainable parameters=89.623M

where embeddings =41.685M, softmax=34.110M, core = 13.828M

### Mini-exp B

|  |  |  |
| --- | --- | --- |
| **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=4  TBPTT length=8 |
| linear2Globals FF-NN | grapharea=32, hops=1 |
|  | learning rate=0.0001 |
| dropout=none |

|  |  |
| --- | --- |
| Epoch | Training PPL on globals |
| 1 |  |
| 2 |  |
| 5 |  |
| 10 |  |
| 50 |  |
| 100 |  |
| 150 |  |
| 200 |  |
| 250 |  |
| 300 |  |
| 350 |  |

### 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.

|  |  |  |
| --- | --- | --- |
| **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* | *Training PPL, senses* | *Training PPL,*  *multi-senses* | *Validation PPL, globals* | *Validation PPL, senses* | *Validation PPL, multi-senses* |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |
| 17 |  |  |  |  |  |  |
| 18 |  |  |  |  |  |  |
| 19 |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |
| 21 |  |  |  |  |  |  |
| 22 |  |  |  |  |  |  |
| 23 |  |  |  |  |  |  |
| 24 |  |  |  |  |  |  |
| 25 |  |  |  |  |  |  |

# 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

# 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

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

# Freezing standard LM and then proceeding to Senses

## Notes & ideas

The idea is:

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

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

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

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

I could put the loss of senses artificially at 0…

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

In the initialization function:

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

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

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

In the forward() call:

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

In Training.py:

In training\_setup():

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

(taken care of above)

In train\_loop():

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

**else**:  
 loss = loss\_global

In compute\_model\_loss():

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

In evaluation():

including\_senses = model\_forParameters.predict\_senses

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

**…**

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

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

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

At that point,

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

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

## Implementation

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

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

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

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

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

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

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

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

'main\_rnn\_ls.1.weight\_ih\_l0', …,

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

We should set requires\_grad=False

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

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

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

Yes.

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

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

What could this be due to?

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

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

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

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

In Epoch 1:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In Epoch 4:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

INFO : Number of trainable parameters=48669959

Maybe its due to the momentum?

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

What happens if I do not recreate the optimizer?

Still the same.

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

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

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