[Preliminary information (Datasets, graphs) 3](#_Toc38712604)

[WikiText-2 3](#_Toc38712605)

[WikiText-2 Stats 3](#_Toc38712606)

[Graph 3](#_Toc38712607)

[SemCor 4](#_Toc38712608)

[SemCor stats 4](#_Toc38712609)

[Graph 4](#_Toc38712610)

[Globals - RNN 5](#_Toc38712611)

[Architecture 5](#_Toc38712612)

[Experiments 5](#_Toc38712613)

[Mini-experiment – overfit on fragment 5](#_Toc38712614)

[Experiment – RNN on WT2 7](#_Toc38712615)

[Globals – GRU 8](#_Toc38712616)

[Architecture 8](#_Toc38712617)

[Experiments 8](#_Toc38712618)

[Mini-experiment – overfit GRU on fragment of WT2 8](#_Toc38712619)

[Experiment – GRU on WT2 10](#_Toc38712620)

[Including the graph input 12](#_Toc38712621)

[Review: Graph Attention Networks 12](#_Toc38712622)

[Review: RGAT: Relational Graph-Attention Networks 13](#_Toc38712623)

[Globals & Graph Input – Graph Attention Networks 14](#_Toc38712624)

[GAT, input signals, 4-heads 14](#_Toc38712625)

[Architecture 14](#_Toc38712626)

[Mini-Experiment, GRU-GAT4 on fragment of WT2 15](#_Toc38712627)

[Experiment – GRU-GAT4 on WT2 15](#_Toc38712628)

[Experiment – GAT with 1 head 17](#_Toc38712629)

[The Sense task 19](#_Toc38712630)

[Reflections 19](#_Toc38712631)

[Baseline: Simple GRU on SemCor 19](#_Toc38712632)

[Mini-experiment: overfitting on a fragment of SemCor 19](#_Toc38712633)

[Experiment – GRU on SemCor 21](#_Toc38712634)

[Baseline 2: GRU + GAT4 (current global’s node state) on SemCor 22](#_Toc38712635)

[Mini-Experiment – Overfit on a fragment of SemCor 22](#_Toc38712636)

[Experiment – GRU+GAT4 on SemCor 22](#_Toc38712637)

[Senses: Photo-concat 24](#_Toc38712638)

[Architecture 24](#_Toc38712639)

[Experiment – Globals and senses (GRU + 2xGAT4) on SemCor 25](#_Toc38712640)

[Senses: Shared GRU layer 26](#_Toc38712641)

[Architecture 26](#_Toc38712642)

[Experiment 1 – Senses: Shared GRU Layer 27](#_Toc38712643)

[Senses: Self-attention scores on the senses of the k most likely globals 28](#_Toc38712644)

[Design 28](#_Toc38712645)

[Reviewing self-attention 29](#_Toc38712646)

[Implementation 29](#_Toc38712647)

[Experiments 31](#_Toc38712648)

[Architecture – IV 33](#_Toc38712649)

[Design 33](#_Toc38712650)

[Meetings 34](#_Toc38712651)

[IA 34](#_Toc38712652)

[IA, PSK, KF 34](#_Toc38712653)

[After the meetings 35](#_Toc38712654)

[Senses Baseline: Operate on the k most likely globals 35](#_Toc38712655)

[Hypothesis 1: FF-NN – from all logits to all logits 36](#_Toc38712656)

[Hypothesis 2: Project embeddings of the first k globals + concat + FF-NN 36](#_Toc38712657)

[Senses: self-attention on the k most likely globals 36](#_Toc38712658)

[Design 37](#_Toc38712659)

[Implementation 37](#_Toc38712660)

[Overfit on mini-dataset 37](#_Toc38712661)

[Globals – LSTM 38](#_Toc38712662)

[Mini-experiment – overfit LSTM on fragment of WT2 38](#_Toc38712663)

[Experiment – LSTM on WT2 39](#_Toc38712664)

# Preliminary information (Datasets, graphs)

## WikiText-2

### WikiText-2 Stats

Number of tokens in the splits, counted using nltk’s tokenizer:

Training: 2,207,934 tokens (1,774,387 without punctuation)

Validation: 242,478 (186,258)

Test: 277,846 (209,228)

Mini-Dataset – training: 632 tokens (530 without punctuation)

### Graph

After processing WikiText-2 (reading in, getting FastText single-prototype embeddings, retrieving WordNet informations and egdes), the resulting graph will be:

(From the log at DefineGraph.log)

Constructing X, matrix of node features

**X\_definitions**.shape=torch.Size([**28070**, 300])

**X\_examples**.shape=torch.Size([**26544**, 300])

**X\_senses**.shape=torch.Size([**28070**, 300])

**X\_globals**.shape=torch.Size([**31640**, 300])

Defining the edges: def, exs

**def\_edges**\_se.\_\_len\_\_()=**28070**

**exs\_edges**\_se.\_\_len\_\_()=**26544**

Defining the edges: sc

**sc\_edges**.\_\_len\_\_()=**28070**

**#** Currently, since WordNet is our only source, we have the correspondence 1sense-1definition, so the number of SenseChildren edges and the number of Definition edges coincide.

**sc\_edges\_with\_selfloops**.\_\_len\_\_()=**49276**

**#** The Relational Graph Convolutional Network and Graph Attention Network both require that all nodes have at least 1 edge – to satisfy this requirement, we add a self-loop to all the globals that do not have a sense

(example: the stopwords, like ‘for’, ‘and’, ‘of’, etc.)

This way, we also determine that there were 49276 – 28070 **= 21206** globals with no dictionary information, over a total of **31640 (67%).**

syn\_edges.\_\_len\_\_()=**40016 #** synonyms

ant\_edges.\_\_len\_\_()=**3938** **#** antonyms

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

Data(edge\_index=[2, 147844], edge\_type=[**147844**], node\_types=[**114324**], num\_relations=[1], **x=[114324, 300]**)

## 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: 180 tokens.

### Graph

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

Constructing X, matrix of node features

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

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

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

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

# Globals - RNN

## Architecture

The first baseline is a simple Recurrent Neural Network.

* X= input embeddings.
  + w(t)= the current global word
* x(t) = w(t)++s(t-1).   
  input\_x = torch.cat([currentword\_embedding, self.memory\_context])
* New state: use the matrix W. E.g.: (300+512) > 512. s(t) = W\*x(t)
* Save s(t) in the memory buffer
* Obtain the logits, using the matrix linear2global: 1024 -> |Vocab|
* Apply softmax

## Experiments

### Mini-experiment – overfit on fragment

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Simple RNN | 1) The word embedding of the current global | batch\_size=4 |
| 2 layers: 600,300 | TBPTT length=16 |
|  | learning rate=0.001 |
|  |  |

{len(train\_dataloader)=9 (tokens>=~9\*4\*16>=576, we know they are 632)}

|  |  |
| --- | --- |
| **Epoch** | **Training perplexity** |
| 1 | 79684.59 |
| 2 | 23370.42 |
| 3 | 11611.48 |
| 4 | 616.67 |
| 5 | 381.21 |
| 10 | 91.15 |
| 30 | 2.86 |
| 50 | 1.24 |
| 100 | 1.12 |

Doubt: why do I have 79K perplexity after the 1st epoch, when my vocabulary of globals is only **31640**?

In the 1st epoch, while 7 out of 9 batches have loss around 9 or 10, the last 2 have loss 26 and 65 respectively. Same for the 1st batch of the 2nd epoch (25.6).

Probably due to extreme values (0 or inf) in the network.

Nevertheless, this artefact disappears during the rest of the training.

**Reviewing the predictions:**

Label: the next global is: **less**(from 1575)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: less ; probability = 99.9%**

INFO : Word: the ; probability = 0.1%

INFO : Word: this ; probability = 0.01%

INFO : Word: during ; probability = 0.0%

INFO : Word: followed ; probability = 0.0%

INFO :

Label: the next global is: **than**(from 167)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: than ; probability = 100.0%**

INFO : Word: Kurfürst ; probability = 0.0%

INFO : Word: a ; probability = 0.0%

INFO : Word: were ; probability = 0.0%

INFO : Word: period ; probability = 0.0%

INFO :

Label: the next global is: **half**(from 904)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: half ; probability = 99.99%**

INFO : Word: a ; probability = 0.0%

INFO : Word: Cruiser ; probability = 0.0%

INFO : Word: draw ; probability = 0.0%

INFO : Word: at ; probability = 0.0%

INFO :

Label: the next global is: **an**(from 158)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: an ; probability = 100.0%**

INFO : Word: sisters ; probability = 0.0%

INFO : Word: began ; probability = 0.0%

INFO : Word: a ; probability = 0.0%

INFO : Word: resulted ; probability = 0.0%

INFO :

Label: the next global is: **hour**(from 2780)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: hour ; probability = 99.97%**

INFO : Word: fired ; probability = 0.03%

INFO : Word: in ; probability = 0.0%

INFO : Word: on ; probability = 0.0%

INFO : Word: Beatty ; probability = 0.0%

INFO :

Label: the next global is: later(from 678)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: later ; probability = 99.93%**

INFO : Word: their ; probability = 0.05%

INFO : Word: fired ; probability = 0.01%

INFO : Word: crew ; probability = 0.0%

INFO : Word: duel ; probability = 0.0%

INFO:

The predictions appear to be correct – a simple RNN manages to overfit on a fragment of WikiText-2.

### Experiment – RNN on WT2

(from the previous series)

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| Simple RNN | 1) The word embedding of the current global | batch\_size=40 |
| 2 layers: 512,256 | TBPTT length=32 |
|  | learning rate=10^(-4) |
|  |  |

Number of trainable parameters=43.042M

INFO : Parameters:

('module.X', torch.Size([114324, 300]), True

('module.memory\_h1', torch.Size([1, 512]), False), ('module.memory\_h2', torch.Size([1, 256]), False),

('module.W1.weight', torch.Size([512, 812]), True), ('module.W1.bias', torch.Size([512]), True),

('module.W2.weight', torch.Size([256, 768]), True), ('module.W2.bias', torch.Size([256]), True), ('module.linear2global.weight', torch.Size([31640, 256]), True), ('module.linear2global.bias', torch.Size([31640]), True)]

34.30M embeddings, 8.1M softmax, 0.62M core

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity* | *Validation perplexity* |
| 1 | 1034.0 | 472.72 |
| 2 | 548.1 | 385.2 |
| 3 | 438.11 | 345.09 |
| 4 | 374.13 | 318.25 |
| 5 | 329.48 | 302.44 |
| 6 | 296.01 | 293.35 |
| 7 | 269.14 | 288.09 |
| 8 | 247.38 | 285.5 |
| 9 | 228.91 | 283.75 |
| 10 | 213.34 | 283.78 |
| 11 | 199.86 | 283.23 |
| 12 | 188.19 | 284.34 |
| 13 | 177.92 | 283.85 |
| 14 | 168.97 | 286.75 |
| 15 | 161.08 | 283.29 |
| 16 | 154.02 | 280.45 |
| 17 | 147.79 | 280.46 |
| 18 | 141.92 | 285.7 |

In the “Restricted Recurrent Neural Networks” paper, that we are using as point of comparison, the validation performance of an RNN model on WT-2 is in the range (320,250), with the best value they obtained being 253.1 comparable to our value of 280.45.

# Globals – GRU

## Architecture

Reviewing the formula for the GRU:

* + Update gate:
  + Reset gate:
  + Proposed new-state:
  + Final updated state:

## Experiments

### Mini-experiment – overfit GRU on fragment of WT2

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| GRU RNN  (see the full GRU formula) | 1) The word embedding of the current global | batch\_size=4 |
| 2 layers: 300,300 | TBPTT length=16 |
|  | learning rate=0.001 |
|  |  |

{len(train\_dataloader)=9 (tokens>=~9\*4\*16>=576, we know they are 632)}

|  |  |
| --- | --- |
| **Epoch** | **Training perplexity** |
| 1 | 20311.33 |
| 2 | 514.01 |
| 3 | 134.27 |
| 4 | 109.9 |
| 5 | 101.61 |
| 10 | 96.02 |
| 30 | 88.64 |
| 50 | 56.54 |
| 75 | 11.12 |
| 100 | 2.37 |

**Reviewing the predictions:**

Label: the next global is: **s**(from 1829)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: s ; probability = 42.57%**

INFO : Word: two ; probability = 7.06%

INFO : Word: ordered ; probability = 6.17%

INFO : Word: were ; probability = 4.18%

INFO : Word: and ; probability = 3.71%

INFO :

Label: the next global is: **crew**(from 3618)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: crew ; probability = 34.44%**

INFO : Word: the ; probability = 12.77%

INFO : Word: British ; probability = 3.65%

INFO : Word: time ; probability = 2.89%

INFO : Word: by ; probability = 2.64%

INFO :

Label: the next global is: **spotted**(from 3845)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: spotted ; probability = 55.95%**

INFO : Word: Tiger ; probability = 3.76%

INFO : Word: his ; probability = 3.61%

INFO : Word: torpedo ; probability = 2.35%

INFO : Word: of ; probability = 2.16%

INFO :

Label: the next global is: **both**(from 81)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: both ; probability = 48.73%**

INFO : Word: that ; probability = 5.94%

INFO : Word: from ; probability = 4.72%

INFO : Word: was ; probability = 4.53%

INFO : Word: due ; probability = 3.38%

INFO :

Label: the next global is: **the**(from 31)

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

INFO : The top- 5 predicted globals are:

INFO : **Word: the ; probability = 94.16%**

INFO : Word: by ; probability = 0.78%

INFO : Word: steaming ; probability = 0.41%

INFO : Word: i ; probability = 0.21%

INFO : Word: followed ; probability = 0.2%

INFO :

In the overfitting experiment, the predictions are ok.

### Experiment – GRU on WT2

(from the previous series)

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| GRU RNN  (see the full GRU formula) | 1) The word embedding of the current global | batch\_size=80 |
| 2 layers: 300,300 | TBPTT length=35 |
| Pretrained embeddings from: FastText (d=300) | learning rate=10^(-5) |
|  |  |

[('module.X', torch.Size([114324, 300]), True), ('module.select\_first\_node', torch.Size([1]), False),

('module.memory\_h1', torch.Size([1, 300]), False), ('module.memory\_h2', torch.Size([1, 300]), False), ('module.U\_z\_1.weight', torch.Size([300, 300]), True), ('module.W\_z\_1.weight', torch.Size([300, 300]), True), ('module.U\_r\_1.weight', torch.Size([300, 300]), True), ('module.W\_r\_1.weight', torch.Size([300, 300]), True), ('module.U\_1.weight', torch.Size([300, 300]), True), ('module.W\_1.weight', torch.Size([300, 300]), True), ('module.U\_z\_2.weight', torch.Size([300, 300]), True), ('module.W\_z\_2.weight', torch.Size([300, 300]), True), ('module.U\_r\_2.weight', torch.Size([300, 300]), True), ('module.W\_r\_2.weight', torch.Size([300, 300]), True), ('module.U\_2.weight', torch.Size([300, 300]), True), ('module.W\_2.weight', torch.Size([300, 300]), True), ('module.linear2global.weight', torch.Size([31640, 300]), True), ('module.linear2global.bias', torch.Size([31640]), True)]

Number of trainable parameters=44.90M

34.30M embeddings, 9.52M softmax, 1.08M core

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity* | *Validation perplexity* |
| 1 | 1226.69 | 681.65 |
| 2 | 694.85 | 444.03 |
| 3 | 522.39 | 377.5 |
| 4 | 450.1 | 341.81 |
| 5 | 406.0 | 319.07 |
| 6 | 374.83 | 302.5 |
| 7 | 350.41 | 289.09 |
| 8 | 330.49 | 278.24 |
| 9 | 313.65 | 269.1 |
| 10 | 298.89 | 261.25 |
| 11 | 286.02 | 254.58 |
| 12 | 274.56 | 248.86 |
| 13 | 264.47 | 243.84 |
| 14 | 255.26 | 239.41 |
| 15 | 246.88 | 235.47 |
| 16 | 239.29 | 231.94 |
| 17 | 232.18 | 228.78 |
| 18 | 225.76 | 226.07 |
| 19 | 219.9 | 223.64 |
| 20 | 214.4 | 221.48 |
| 21 | 209.27 | 219.62 |
| 22 | 204.49 | 217.89 |
| 23 | 200.09 | 216.31 |
| 24 | 195.9 | 214.96 |
| 25 | 191.87 | 213.62 |
| 26 | 188.1 | 212.35 |
| 27 | 184.61 | 211.15 |
| 28 | 181.25 | 210.06 |
| 29 | 178.03 | 208.99 |
| 30 | 174.86 | 208.09 |
| 31 | 171.93 | 207.27 |
| 32 | 169.15 | 206.45 |
| 33 | 166.5 | 205.68 |
| 34 | 163.81 | 205.07 |
| 35 | 161.3 | 204.35 |
| 36 | 158.91 | 203.68 |
| 37 | 156.63 | 203.03 |
| 38 | 154.5 | 202.49 |
| 39 | 152.32 | 201.94 |
| 40 | 150.23 | 201.48 |
| 41 | 148.17 | 201.03 |
| 42 | 146.31 | 200.55 |
| 43 | 144.47 | 200.21 |
| 44 | 142.68 | 200.0 |
| 45 | 140.97 | 199.49 |
| 46 | 139.21 | 199.32 |

Operating with a (complete) Gated Recurrent Unit, we obtain a Validation Perplexity of 199.3 on WikiText-2.

# Including the graph input

The graph:



(note: the figure also higlights the problem of dealing with inflected forms. We ignore that for now. We could create a new edge type.)

The idea is to Add **in parallel** the input from the graph, as well as the one from the senses:  
current global’s word embedding || global’s node-state || sense’s node state.

What could we use to obtain node states from the graph?

## Review: Graph Attention Networks

The steps of Graph Attention Networks, originally proposed by Velickovic et al. (2017), are the following:

* Shared linear transformation on every node,   
  This can be modified, using a different weight matrix for each node type: definitions, examples, senses, synonyms, antonyms (and globals?)
* Non-normalized attention coefficients for neighbours:   
  The attention mechanism uses a 1-layer FF-NN:
* Get the normalized coefficients using softmax over the neighbourhood:
* Finally, compute the new state of the node:

Multi-head attention (either through averaging or through concatenating) was found beneficial to extend and stabilize the method.

## Review: RGAT: Relational Graph-Attention Networks

* Starting point: a graph with |R| relation types and N nodes.
* We operate with |R|=5 intermediate representations for the nodes, 1 for each relation.  
  In practice, they are summed up to obtain the node rep at the next layer, .
* Attention mechanism, part 1:  
  We must compute the logit scores , from node *i* to *j* under relation *r.* Self-attention is used.
  + We obtain queries and keys of dimension d=1 as follows:  
     and
  + Then, we create either Additive logits: or Multiplicative
* Attention mechanism, part 2:  
  Either WithIn-Relation G.AT. (independent probablity distribution for each relation r)  
  or Across-Relation G.AT. = 1 probability distribution. The sum to 1 across all the neighbouring nodes and relations
* Propagation rule:

Found to be comparable with RGCNs, but not significantly better.

For inductive tasks, ARGAT with multiplicative logits fared slightly better

# Globals & Graph Input – Graph Attention Networks

Note: Originally, I tried replacing the global’s word embedding with the *node state* of the global node from the graph, obtained with a GNN.

It seems that *replacing* the word embedding with the graph signal from a 2-hops neighbourhood introduces noise. Validation perplexity obtained on WikiText-2: 225.8 > 199.

No improvement on GRU\_GNN.

2 modifications must be applied:

* Using (concatenated) **multi-head attention** in the GAT
* Changing altogether the inclusion of the graph-influenced input: instead of **replacing** the pretrained word embedding, it should be **added in parallel**.

## GAT, input signals, 4-heads

### Architecture

Input signals:

1. the current global word
2. the node-state of the word in the KB-graph, obtained applying the GNN
3. the node-state of the current sense (if present)

We concatenate the input signals and send them as input to the 1st GRU layer*.*

Operating with no senses, total dimensions of the concatenated input: 600.

The hidden state at GRU\_h1 will also have increased dimensionality, 600 (from 300).

Parameters:

[('module.X', torch.Size([114324, 300]), True), ('module.select\_first\_node', torch.Size([1]), False), ('module.nodestate\_zeros', torch.Size([1, 300]), False),

('module.memory\_h1', torch.Size([1, 600]), False), ('module.memory\_h2', torch.Size([1, 300]), False), ('module.gat.weight', torch.Size([300, 300]), True), ('module.gat.att', torch.Size([1, 4, 150]), True), ('module.gat.bias', torch.Size([300]), True),

('module.U\_z\_1.weight', torch.Size([600, 600]), True), ('module.W\_z\_1.weight', torch.Size([600, 600]), True), ('module.U\_r\_1.weight', torch.Size([600, 600]), True), ('module.W\_r\_1.weight', torch.Size([600, 600]), True), ('module.U\_1.weight', torch.Size([600, 600]), True), ('module.U\_1.bias', torch.Size([600]), True), ('module.W\_1.weight', torch.Size([600, 600]), True), ('module.W\_1.bias', torch.Size([600]), True), ('module.U\_z\_2.weight', torch.Size([300, 300]), True), ('module.W\_z\_2.weight', torch.Size([300, 600]), True), ('module.U\_r\_2.weight', torch.Size([300, 300]), True), ('module.W\_r\_2.weight', torch.Size([300, 600]), True), ('module.U\_2.weight', torch.Size([300, 300]), True), ('module.U\_2.bias', torch.Size([300]), True), ('module.W\_2.weight', torch.Size([300, 600]), True), ('module.W\_2.bias', torch.Size([300]), True), ('module.linear2global.weight', torch.Size([31640, 300]), True), ('module.linear2global.bias', torch.Size([31640]), True)]

Trainable parameters=46.883M ; 34.30M embeddings, 9.52M softmax, 3.06M core

### Mini-Experiment, GRU-GAT4 on fragment of WT2

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| GRU + GAT with 4 attention heads | 1) Current global’s word embedding  2) current global’s node-state | batch\_size=4 |
| The GRU has 2 layers: 300, 300 | TBPTT length=8 |
| Pretrained embeddings from: FastText | learning rate=10^(-3) |
|  | graph\_area=32 |

|  |  |
| --- | --- |
| **Epoch** | **Training perplexity** |
| 1 | 16303.75 |
| 2 | 437.81 |
| 3 | 130.15 |
| 4 | 109.63 |
| 5 | 101.48 |
| 10 | 95.21 |
| 30 | 61.0 |
| 50 | 7.62 |
| 75 | 1.45 |
| 100 | 1.11 |

### Experiment – GRU-GAT4 on WT2

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| GRU + GAT with 4 attention heads | 1) Current global’s word embedding  2) current global’s node-state | batch\_size=40 |
| The GRU has 2 layers: 600, 300 | TBPTT length=35 |
| Pretrained embeddings from: FastText | learning rate=0.5\* 10^(-4) |
|  | graph\_area=32 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Run 1* | | *Run 2* | |
|  | *Train PPL* | *Valid PPL* | *Train PPL* | *Valid PPL* |
| 1 | 1158.57 | 666.68 |  |  |
| 2 | 675.32 | 438.89 |  |  |
| 3 | 514.35 | 376.3 |  |  |
| 4 | 450.62 | 344.82 |  |  |
| 5 | 411.91 | 323.98 |  |  |
| 6 | 384.14 | 308.58 |  |  |
| 7 | 362.51 | 296.37 |  |  |
| 8 | 344.84 | 286.66 |  |  |
| 9 | 329.53 | 278.12 |  |  |
| 10 | 316.27 | 270.79 |  |  |
| 11 | 304.32 | 264.27 |  |  |
| 12 | 293.48 | 258.2 |  |  |
| 13 | 283.52 | 252.67 |  |  |
| 14 | 274.34 | 247.66 |  |  |
| 15 | 265.88 | 243.3 |  |  |
| 16 | 258.1 | 239.37 |  |  |
| 17 | 250.94 | 235.71 |  |  |
| 18 | 244.27 | 232.26 |  |  |
| 19 | 238.02 | 229.14 |  |  |
| 20 | 232.13 | 226.21 |  |  |
| 21 | 226.74 | 223.62 |  |  |
| 22 | 221.61 | 221.24 |  |  |
| 23 | 216.83 | 218.96 |  |  |
| 24 | 212.31 | 216.91 |  |  |
| 25 | 208.05 | 215.11 |  |  |
| 26 | 204.0 | 213.33 |  |  |
| 27 | 200.18 | 211.71 |  |  |
| 28 | 196.53 | 210.42 |  |  |
| 29 | 193.09 | 209.1 |  |  |
| 30 | 189.71 | 207.78 |  |  |
| 31 | 186.61 | 206.7 |  |  |
| 32 | 183.59 | 205.57 |  |  |
| 33 | 180.7 | 204.68 |  |  |
| 34 | 177.88 | 203.91 |  |  |
| 35 | 175.27 | 203.04 |  |  |
| 36 | 172.72 | 202.27 |  |  |
| 37 | 170.28 | 201.66 |  |  |
| 38 | 167.89 | 200.95 |  |  |
| 39 | 165.57 | 200.27 |  |  |
| 40 | 163.42 | 199.83 |  |  |
| 41 | 161.19 | 199.24 |  |  |
| 41 | 159.17 | 198.72 |  |  |
| 43 | 157.1 | 198.31 |  |  |
| 44 | 155.16 | 197.9 |  |  |
| 45 | 153.26 | 197.44 |  |  |
| 46 | 151.4 | 197.06 |  |  |
| 47 | 149.6 | 196.7 |  |  |
| 48 | 147.92 | 196.36 |  |  |
| 49 | 146.17 | 195.99 |  |  |
| 50 | 144.51 | 195.46 |  |  |
| 51 | 142.86 | 194.95 |  |  |
| 52 | 141.34 | 194.48 |  |  |
| 53 | 139.82 | 194.06 |  |  |
| 54 | 138.34 | 193.6 |  |  |
| 55 | 136.91 | 193.16 |  |  |
| 56 | 135.46 | 192.92 |  |  |
| 57 | 134.1 | 192.5 |  |  |
| 58 | 132.79 | 192.33 |  |  |
| 59 | 131.49 | 192.01 |  |  |
| 60 | 130.16 | 191.86 |  |  |

191.8 Validation Perplexity on WikiText-2, better than the 199 obtained with a GRU that used only the Word Embedding as an Input Signal.

### Experiment – GAT with 1 head

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| GRU + GAT | 1) Current global’s word embedding  2) current global’s node-state | batch\_size=40 |
| GRU: 2 layers: 600, 300 | TBPTT length=35 |
| GAT: 1 attention head | learning rate=0.5\* 10^(-4) |
|  | graph\_area=32 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Run 1* | | *Run 2* | |
|  | Train PPL | Valid PPL | Train PPL | Valid PPL |
| 1 | 1121.7 | 585.29 |  |  |
| 2 | 645.82 | 442.49 |  |  |
| 3 | 531.05 | 395.01 |  |  |
| 4 | 470.75 | 359.85 |  |  |
| 5 | 425.67 (411.91) | 333.01 (323.98) |  |  |
| 6 | 391.9 | 313.56 |  |  |
| 7 | 365.68 | 299.06 |  |  |
| 8 | 344.83 | 288.12 |  |  |
| 9 | 327.42 | 278.83 |  |  |
| 10 | 312.78 (316.27) | 271.01 (270.79) |  |  |
| 11 | 299.8 | 264.19 |  |  |
| 12 | 288.27 | 258.05 |  |  |
| 13 | 277.72 | 252.5 |  |  |
| 14 | 268.3 | 247.34 |  |  |
| 15 | 259.63 | 243.05 |  |  |
| 16 | 251.68 | 238.9 |  |  |
| 17 | 244.33 | 235.22 |  |  |
| 18 | 237.52 | 231.76 |  |  |
| 19 | 231.29 | 228.76 |  |  |
| 20 | 225.35 (232.13) | 225.83 (226.21) |  |  |
| 21 | 219.98 | 223.45 |  |  |
| 22 | 214.9 | 221.34 |  |  |
| 23 | 209.97 | 219.27 |  |  |
| 24 | 205.46 | 217.36 |  |  |
| 25 | 201.1 | 215.71 |  |  |
| 26 | 196.97 | 214.14 |  |  |
| 27 | 193.09 | 212.92 |  |  |
| 28 | 189.32 | 211.9 |  |  |
| 29 | 185.78 | 210.78 |  |  |
| 30 | 182.33 (189.71) | 209.9 (207.78) |  |  |
| 31 | 179.06 | 208.99 |  |  |
| 32 | 175.96 | 208.21 |  |  |
| 33 | 172.89 | 207.44 |  |  |
| 34 | 169.98 | 206.91 |  |  |
| 35 | 167.2 | 206.42 |  |  |
| 36 | 164.49 | 205.98 |  |  |
| 37 | 161.94(170.28) | 205.39(201.66) |  |  |

Conclusion: operating with only 1 head is not as effective as having multiple attention heads in the GAT.  
From the descent, we can reasonably expect that (GAT-1head > simple GRU). However, (GAT-4heads > GAT-1head).

# The Sense task

## Reflections

The starting point is that the original idea was suboptimal:

It was:   
(1st part, GRU, GNNs etc.) > representation > in parallel, 1 FF-NN to the globals’ logits and   
 || 1 FF-NN to the senses’ logits

Preliminary experiments on SemCor indicated that when the perplexity on globals was decreasing, the PPL on senses ended up increasing.

The reason was that I was trying to adjust 1 encoding (the representation built with GNNs & co.) to do 2 tasks: predictions on globals and predictions on senses.

## Baseline: Simple GRU on SemCor

**Parameters:**

[('module.X', torch.Size([99963, 300]), True), ('module.select\_first\_node', torch.Size([1]), False), ('module.embedding\_zeros', torch.Size([1, 300]), False), ('module.memory\_h1', torch.Size([1, 300]), False), ('module.memory\_h2', torch.Size([1, 300]), False), ('module.U\_z\_1.weight', torch.Size([300, 300]), True), ('module.W\_z\_1.weight', torch.Size([300, 300]), True), ('module.U\_r\_1.weight', torch.Size([300, 300]), True), ('module.W\_r\_1.weight', torch.Size([300, 300]), True), ('module.U\_1.weight', torch.Size([300, 300]), True), ('module.U\_1.bias', torch.Size([300]), True), ('module.W\_1.weight', torch.Size([300, 300]), True), ('module.W\_1.bias', torch.Size([300]), True), ('module.U\_z\_2.weight', torch.Size([300, 300]), True), ('module.W\_z\_2.weight', torch.Size([300, 300]), True), ('module.U\_r\_2.weight', torch.Size([300, 300]), True), ('module.W\_r\_2.weight', torch.Size([300, 300]), True), ('module.U\_2.weight', torch.Size([300, 300]), True), ('module.U\_2.bias', torch.Size([300]), True), ('module.W\_2.weight', torch.Size([300, 300]), True), ('module.W\_2.bias', torch.Size([300]), True), ('module.linear2global.weight', torch.Size([21988, 300]), True), ('module.linear2global.bias', torch.Size([21988]), True)]

Number of trainable parameters=37.69M, where 6.62M softmax, 29.99M embeddings, ~1.08M core

### Mini-experiment: overfitting on a fragment of SemCor

(Mini-Dataset – training: 180 tokens)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Training hyperparameters** | | **Input signals & GNN** | | **Architecture** | |
| batch\_size | 4 | grapharea | I: 32n | II: 181e | GRU layers | 2 |
| learning rate | 0.001 | Input signals | 1) current word embedding | hidden dim.s | 300,300 |
| TBPTT length | 16 |  |  |  |

|  |  |
| --- | --- |
| **Epoch** | **Training perplexity** |
| 1 | 20725.76 |
| 5 | 96.59 |
| 10 | 46.49 |
| 20 | 45.48 |
| 30 | 45.26 |
| 50 | 39.78 |
| 100 | 24.66 |
| 150 | 15.71 |
| 200 | 3.16 |

**Reviewing the predictions:**

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 = 16.48%**

INFO : Word: produced ; probability = 13.0%

INFO : Word: . ; probability = 10.6%

INFO : Word: relative ; probability = 6.38%

INFO : Word: s ; probability = 5.68%

INFO :

Label: the next global is: **primary election**(from 9)

INFO : Label: the next sense is: primary.n.01(from 17809)

INFO : The top- 5 predicted globals are:

INFO : **Word: primary election ; probability = 20.99%**

INFO : Word: was ; probability = 14.71%

INFO : Word: " ; probability = 10.42%

INFO : Word: reports ; probability = 4.05%

INFO : Word: of ; probability = 3.89%

INFO :

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 ; probability = 17.9%**

INFO : Word: recent ; probability = 10.71%

INFO : Word: . ; probability = 9.93%

INFO : Word: relative ; probability = 6.46%

INFO : Word: s ; probability = 5.18%

INFO :

Overfitting the GRU on fragment of SemCor: confirmed.

### Experiment – GRU on SemCor

(Run 1 is from the previous series)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Training hyperparameters** | | **Input signals & GNN** | | **Architecture** | |
| batch\_size | 40 | grapharea | I: 32n | II: 181e | GRU layers | 2 |
| learning rate | 1\* 10^(-4) | Input signals | 1) current word embedding | hidden dim.s | 300,300 |
| TBPTT length | 35 |  | Dropout | 0.1 on GRU W1, W2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Run 1* | | *Run 2* | |
|  | Train PPL | Valid PPL | Train PPL | Valid PPL |
| 1 | 1160.8 | 512.56 | 1152.3 | 500.47 |
| 2 | 592.36 | 371.88 | 576.01 | 362.94 |
| 3 | 447.17 | 318.74 | 437.35 | 311.46 |
| 4 | 389.24 | 293.26 | 378.46 | 288.02 |
| 5 | 352.05 | 277.62 | 344.2 | 274.89 |
| 6 | 325.85 | 266.19 | 321.18 | 264.72 |
| 7 | 306.34 | 256.95 | 303.29 | 255.9 |
| 8 | 290.63 | 249.21 | 287.73 | 247.86 |
| 9 | 276.94 | 242.02 | 274.08 | 240.81 |
| 10 | 264.88 | 235.99 | 262.45 | 235.38 |
| 11 | 254.07 | 230.65 | 252.32 | 230.51 |
| 12 | 244.18 | 226.36 | 243.29 | 226.65 |
| 13 | 235.07 | 222.76 | 234.99 | 223.46 |
| 14 | 226.92 | 219.69 | 227.56 | 220.68 |
| 15 | 219.65 | 217.09 | 220.81 | 218.17 |
| 16 | 212.96 | 214.94 | 214.5 | 216.15 |
| 17 | 207.14 | 212.94 | 208.8 | 214.22 |
| 18 | 201.55 | 211.29 | 203.23 | 212.76 |
| 19 | 196.51 | 209.76 | 198.2 | 211.32 |
| 20 | 192.0 | 208.54 | 193.51 | 210.15 |
| 21 | 187.67 | 207.39 | 189.17 | 208.9 |
| 22 | 183.68 | 206.49 | 184.99 | 208.06 |
| 23 | 179.96 | 205.57 | 181.3 | 207.21 |
| 24 | 176.36 | 204.91 | 177.64 | 206.52 |
| 25 | 172.92 | 204.41 | 174.17 | 206.15 |
| 26 | 169.72 | 203.82 | 171.0 | 205.55 |
| 27 | 166.71 | 203.58 | 167.96 | 205.45 |
| 28 | 163.69 | 203.23 | 164.88 | 205.22 |
| 29 | 160.78 | 203.03 | 162.02 | 204.95 |
| 30 | 157.97 | 202.63 | 159.26 | 204.86 |
| 31 | 155.36 | 202.61 |  |  |
| 32 | 152.77 | 202.4 |  |  |
| 33 | 150.34 | 202.49 |  |  |
| 34 | 147.87 | 202.42 |  |  |
| 35 | 145.6 | 202.34 |  |  |
| 36 | 143.37 | 202.53 |  |  |
| 37 | 141.23 | 202.76 |  |  |

With the given hyperparameters (that are not necessarily optimal – we are running a comparison with the inclusion of the graph input and the addition of senses) we obtain a min. Valid-PPL of 202 on SemCor, with a vocabulary of globals of |V|=21,988

## Baseline 2: GRU + GAT4 (current global’s node state) on SemCor

We examine the performance of the same model that brough us improvements on WikiText-2 compared to the simple GRU – i.e. the model that includes the node-state of the current global, obtained from the graph by applying a GAT.

**Parameters:**

[('module.X', torch.Size([99963, 300]), True), ('module.select\_first\_node', torch.Size([1]), False), ('module.nodestate\_zeros', torch.Size([1, 300]), False), ('module.memory\_h1', torch.Size([1, 600]), False), ('module.memory\_h2', torch.Size([1, 300]), False), ('module.gat.weight', torch.Size([300, 300]), True), ('module.gat.att', torch.Size([1, 4, 150]), True), ('module.gat.bias', torch.Size([300]), True), ('module.U\_z\_1.weight', torch.Size([600, 600]), True), ('module.W\_z\_1.weight', torch.Size([600, 600]), True), ('module.U\_r\_1.weight', torch.Size([600, 600]), True), ('module.W\_r\_1.weight', torch.Size([600, 600]), True), ('module.U\_1.weight', torch.Size([600, 600]), True), ('module.U\_1.bias', torch.Size([600]), True), ('module.W\_1.weight', torch.Size([600, 600]), True), ('module.W\_1.bias', torch.Size([600]), True), ('module.U\_z\_2.weight', torch.Size([300, 300]), True), ('module.W\_z\_2.weight', torch.Size([300, 600]), True), ('module.U\_r\_2.weight', torch.Size([300, 300]), True), ('module.W\_r\_2.weight', torch.Size([300, 600]), True), ('module.U\_2.weight', torch.Size([300, 300]), True), ('module.U\_2.bias', torch.Size([300]), True), ('module.W\_2.weight', torch.Size([300, 600]), True), ('module.W\_2.bias', torch.Size([300]), True), ('module.linear2global.weight', torch.Size([21988, 300]), True), ('module.linear2global.bias', torch.Size([21988]), True)]

Number of trainable parameters=39.67M, where 6.62M softmax, 29.99M embeddings, ~3.06M core

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

### Experiment – GRU+GAT4 on SemCor

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Training hyperparameters** | | **Input signals & GNN** | | **Architecture** | |
| batch\_size | 40 | grapharea | I: 32n | II: 181e | GRU layers | 2 |
| learning rate | 1\* 10^(-4) | Input signals | 1) current word embedding | hidden dim.s | 600,300 |
| TBPTT length | 35 | 2) current word node-state | Dropout | 0.1 on GRU W1, W2 |
|  |  | GNN | GAT, 4 heads |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Run 1* | | *Run 2* | |
|  | *Training PPL* | *Validation PPL* | *Training PPL* | *Validation PPL* |
| 1 | 1097.37 | 515.14 |  |  |
| 2 | 604.26 | 363.72 |  |  |
| 3 | 442.33 | 314.53 |  |  |
| 4 | 384.1 | 290.12 |  |  |
| 5 | 348.03 | 274.99 |  |  |
| 6 | 321.38 | 262.19 |  |  |
| 7 | 299.82 | 251.3 |  |  |
| 8 | 281.92 | 243.1 |  |  |
| 9 | 266.89 | 236.23 |  |  |
| 10 | 254.27 | 230.96 |  |  |
| 11 | 243.59 | 226.13 |  |  |
| 12 | 234.27 | 222.1 |  |  |
| 13 | 225.63 | 218.67 |  |  |
| 14 | 217.94 | 215.57 |  |  |
| 15 | 210.91 | 212.6 |  |  |
| 16 | 204.41 | 210.59 |  |  |
| 17 | 198.69 | 208.53 |  |  |
| 18 | 193.23 | 207.21 |  |  |
| 19 | 188.31 | 205.75 |  |  |
| 20 | 183.83 | 204.55 |  |  |
| 21 | 179.44 | 203.63 |  |  |
| 22 | 175.5 | 202.84 |  |  |
| 23 | 171.81 | 202.28 |  |  |
| 24 | 168.29 | 201.88 |  |  |
| 25 | 164.92 | 201.69 |  |  |
| 26 | 161.75 | 201.46 |  |  |
| 27 | 158.71 | 201.5 |  |  |
| 28 | 155.78 | 201.52 |  |  |
| 29 | 152.78 | 201.72 |  |  |
| 30 | 150.03 (157.97) | 201.55 (202.63) |  |  |

Training on the SemCor dataset, and evaluating the perplexity on the global words, we observe that including the KB input only gives a marginal improvement (201.5 vs 202.3).

## Senses: Photo-concat

### Architecture

Consideration: It would be useful to   
Idea: take a “photo” of the encoding produced by the 2-layer GRU. We copy the value, with no gradient.

The input from the current word can be given as the concatenation of (global) Word Embedding + (global) Node State + (sense) Node State.

Context data + current word data are then passed on to a FF-NN.



Parameters:

[('module.X', torch.Size([99963, 300]), True), ('module.select\_first\_node', torch.Size([1]), False), ('module.nodestate\_zeros', torch.Size([1, 300]), False), ('module.memory\_h1', torch.Size([1, 600]), False), ('module.memory\_h2', torch.Size([1, 300]), False), ('module.gat\_globals.weight', torch.Size([300, 300]), True), ('module.gat\_globals.att', torch.Size([1, 4, 150]), True), ('module.gat\_globals.bias', torch.Size([300]), True), ('module.gat\_senses.weight', torch.Size([300, 300]), True), ('module.gat\_senses.att', torch.Size([1, 4, 150]), True), ('module.gat\_senses.bias', torch.Size([300]), True), ('module.U\_z\_1.weight', torch.Size([600, 600]), True), ('module.W\_z\_1.weight', torch.Size([600, 900]), True), ('module.U\_r\_1.weight', torch.Size([600, 600]), True), ('module.W\_r\_1.weight', torch.Size([600, 900]), True), ('module.U\_1.weight', torch.Size([600, 600]), True), ('module.U\_1.bias', torch.Size([600]), True), ('module.W\_1.weight', torch.Size([600, 900]), True), ('module.W\_1.bias', torch.Size([600]), True), ('module.U\_z\_2.weight', torch.Size([300, 300]), True), ('module.W\_z\_2.weight', torch.Size([300, 600]), True), ('module.U\_r\_2.weight', torch.Size([300, 300]), True), ('module.W\_r\_2.weight', torch.Size([300, 600]), True), ('module.U\_2.weight', torch.Size([300, 300]), True), ('module.U\_2.bias', torch.Size([300]), True), ('module.W\_2.weight', torch.Size([300, 600]), True), ('module.W\_2.bias', torch.Size([300]), True), ('module.linear2global.weight', torch.Size([21988, 300]), True), ('module.linear2global.bias', torch.Size([21988]), True), ('module.linear2sense.weight', torch.Size([25986 , 1200]), True), ('module.linear2sense.bias', torch.Size([25986]), True)]

Number of trainable parameters=71.51M, where Softmax=6.62M+31.21M, Embeddings=29.99M, core=3.69M (from the 3.06M with 1 GAT)

### Experiment – Globals and senses (GRU + 2xGAT4) on SemCor

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Training hyperparameters** | | **Input signals & GNN** | | **Architecture** | |
| batch\_size | 40 | grapharea | I: 32n | II: 181e | GRU layers | 2 |
| learning rate | 1\* 10^(-4) | Input signals | 1) current word embedding | hidden dim.s | 600,300 |
| TBPTT length | 35 | 2) current word node-state | Dropout | 0.1 on GRU W1, W2 |
| GNN | 2 GATs, 4 heads |  | 3) current *sense* node-state |  |  |

We also have sense prediction, that works using just a FF-NN on the concatenated input: copied h2 from the GRU || word embedding || global word node-state || sense node-state ||

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity* | | *Validation perplexity* | |
|  | Globals | Senses | Globals | Senses |
| 1 | 1104.34 | 5368.16 | 510.79 | 849.74 |
| 2 | 587.02 | 2973.06 | 353.61 | 758.61 |
| 3 | 417.42 | 2212.39 | 289.25 | 693.2 |
| 4 | 342.46 | 1815.9 | 261.3 | 675.57 |
| 5 | 301.03 | 1591.29 | 245.63 | 672.9 |
| 6 | 274.34 | 1432.27 | 235.33 | 675.28 |
| 7 | 255.2 | 1305.72 | 227.81 | 681.36 |
| 8 | 240.04 | 1199.06 | 222.46 | 687.76 |
| 9 | 227.35 | 1107.27 | 216.99 | 699.28 |
| 10 | 216.69 | 1025.38 | 213.57 | 712.9 |
| 11 | 207.53 | 951.97 | 210.25 | 726.85 |

It caused early stopping. The senses use a FF-NN that takes in as input: h2 ++ (word ++ global node ++ sense node). They depend on h2, and as h2 is tuned and modified only depending on the globals’ loss.

Maybe if h1 was shared (with gradient, not only as a copy) between the two tasks, and then we had a 2nd separate GRU layer?

## Senses: Shared GRU layer

### Architecture



**Parameters:**

[('module.X', torch.Size([99963, 300]), True), ('module.select\_first\_node', torch.Size([1]), False), ('module.nodestate\_zeros', torch.Size([1, 300]), False), ('module.memory\_h1', torch.Size([1, 600]), False), ('module.memory\_h2', torch.Size([1, 300]), False), ('module.memory\_h2b', torch.Size([1, 300]), False), ('module.gat\_globals.weight', torch.Size([300, 300]), True), ('module.gat\_globals.att', torch.Size([1, 4, 150]), True), ('module.gat\_globals.bias', torch.Size([300]), True), ('module.gat\_senses.weight', torch.Size([300, 300]), True), ('module.gat\_senses.att', torch.Size([1, 4, 150]), True), ('module.gat\_senses.bias', torch.Size([300]), True), ('module.U\_z\_1.weight', torch.Size([600, 600]), True), ('module.W\_z\_1.weight', torch.Size([600, 900]), True), ('module.U\_r\_1.weight', torch.Size([600, 600]), True), ('module.W\_r\_1.weight', torch.Size([600, 900]), True), ('module.U\_1.weight', torch.Size([600, 600]), True), ('module.U\_1.bias', torch.Size([600]), True), ('module.W\_1.weight', torch.Size([600, 900]), True), ('module.W\_1.bias', torch.Size([600]), True), ('module.U\_z\_2.weight', torch.Size([300, 300]), True), ('module.W\_z\_2.weight', torch.Size([300, 600]), True), ('module.U\_r\_2.weight', torch.Size([300, 300]), True), ('module.W\_r\_2.weight', torch.Size([300, 600]), True), ('module.U\_2.weight', torch.Size([300, 300]), True), ('module.U\_2.bias', torch.Size([300]), True), ('module.W\_2.weight', torch.Size([300, 600]), True), ('module.W\_2.bias', torch.Size([300]), True), ('module.U\_z\_2b.weight', torch.Size([300, 300]), True), ('module.W\_z\_2b.weight', torch.Size([300, 600]), True), ('module.U\_r\_2b.weight', torch.Size([300, 300]), True), ('module.W\_r\_2b.weight', torch.Size([300, 600]), True), ('module.U\_2b.weight', torch.Size([300, 300]), True), ('module.U\_2b.bias', torch.Size([300]), True), ('module.W\_2b.weight', torch.Size([300, 600]), True), ('module.W\_2b.bias', torch.Size([300]), True), ('module.linear2global.weight', torch.Size([21988, 300]), True), ('module.linear2global.bias', torch.Size([21988]), True), ('module.linear2sense.weight', torch.Size([25986, 300]), True), ('module.linear2sense.bias', torch.Size([25986]), True)]

Number of trainable parameters=48.93M, where Softmax = 6.62M + 7.82M, Embeddings=29.99M, core=4.5M   
(2 GATs, and 2 second layers of the GRU for the 2 different tasks of globals and senses)

### Experiment 1 – Senses: Shared GRU Layer

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Training hyperparameters** | | **Input signals & GNN** | | **Architecture** | |
| batch\_size | 40 | grapharea | I: 32n | II: 181e | GRU layers | 2 |
| learning rate | 1\* 10^(-4) | Input signals | 1) current word embedding | hidden dim.s | 600,300 |
| TBPTT length | 35 | 2) current word node-state | Dropout | 0.1 on GRU W1, W2 |
| GNN | 2 GATs, 4 heads |  | 3) current *sense* node-state |  |  |

**Run n. 1:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Epoch* | *Training perplexity* | | *Validation perplexity* | |
|  | Globals | Senses | Globals | Senses |
| 1 | 1106.28 | 5481.8 | 497.85 | 906.28 |
| 2 | 543.93 | 3233.42 | 346.43 | 792.47 |
| 3 | 413.44 | 2691.96 | 297.21 | 746.68 |
| 4 | 352.45 | 2291.23 | 272.99 | 708.21 |
| 5 | 315.81 | 2028.78 | 258.67 | 683.04 |
| 6 | 291.14 | 1839.55 | 249.37 | 662.51 |
| 7 | 272.99 | 1693.47 | 242.19 | 651.76 |
| 8 | 258.16 | 1572.48 | 237.35 | 643.23 |
| 9 | 245.72 | 1469.71 | 233.13 | 641.04 |
| 10 | 235.15 | 1378.23 | 229.57 | 642.25 |
| 11 | 225.58 | 1298.84 | 225.68 | 644.98 |
| 12 | 217.04 | 1229.62 | 222.35 | 647.24 |
| 13 | 209.12 | 1163.16 | 219.15 | 650.73 |
| 14 | 202.15 | 1103.04 | 216.29 | 654.98 |
| 15 | 195.66 | 1049.54 | 214.35 | 662.43 |
| 16 | 189.7 | 998.4 | 212.92 | 667.79 |

The perplexity on globals with all 3 input signals was interesting, and it looked on track to overtake previous results before the early stopping was triggered.

It seems that even the current architecture, that shares 1 layer of the GRU between the Global and the Sense tasks and then uses GRU Layer 2 > FF-NN to logits for each, is not good at dealing with senses.

Alternatives:

* 1. Separate the 2 tasks entirely: use 2 GRUs, 1 for globals and 1 for senses
  2. Use the prediction of the next global to influence or restrict the prediction of the next sense.

Of these, **(b)** is the one that warrants exploration the most.

## Senses: Self-attention scores on the senses of the k most likely globals

### Design

How could I make it so that the prediction of the next global influences or restricts the prediction of the next sense?

Hypothesis: I consider the max first k=100 logits of the globals. The predicted sense must be found among the senses of these words.

What do I need to implement this hypothesis:

retrieve the sense indices of the senses of the k likely globals.

Then, either:

1. use the senses architecture as normal; assign to 0 all the logits of senses that are not of the likely globals; softmax&predict
2. consider the first… n=e.g.5\*k senses. Select their embeddings. This would give us a matrix of n (e.g.500) x 300. We have a soft-classification task, where we need to choose one of the rows as the correct one, with a probability distribution over them.  
   To do this, I need several pieces of information:
   * The embeddings of the first n senses, as stated
   * The preceding context, after which we are making the prediction. I could copy h2 from the globals’GRU again?
   * The current token. I may bring the whole input (global word embedding || global node state || sense node state), or only part of it.

The main change of (B) is that, instead of having to create a probability distribution over the whole set of 25K+ senses, the distribution p would be over a small number of candidates, determined by the globals I am currently predicting.

Given: n embeddings of d=300; context information; current token;

how to obtain a probability distribution p over the n sense embeddings, to determine the next sense?

Possibility: Use self-attention.

### Reviewing self-attention

We start with N inputs.

Every input must have three representations: key, query, and value. They do not necessarily have the same number of dimensions as the input (e.g. from 512 to 64, or from 300 to 100, thus allowing for multi-headed self-attention if needed).

We need to use 3 projection matrices: Wq, Wk, Wv.

Their operate on: [n x (d x dq)], [n x (d x dk)], [n x (d x dv)].

The self-attention logit score of word2 from the point of view of word1, we proceed as follows:   
q1\*k2, the dot product of the point of view’s query vector with the key vector of the word we’re scoring.

We obtain: s11= q1\*k1, s12= q1\*k2, s13= q1\*k3, s14= q1\*k4, …

Then, we divide: s11/sqrt(dk) , by the square root of the dimension of the key vectors, and we apply the softmax over the window to obtain the self-attention score.

The next step is to multiply the self-attention score per the Value vector at that position.

The final step is to sum up the value vectors in the window, weighted by the scores from position 1.

I see how self-attention can be useful for the senses’ architecture:

The query q1 will always be the context.

The key will be projected from the candidate senses that we have selected. Likewise the value.

I could also use a simpler alternative, i.e. only the first part of the self-attention mechanism:

s11= q1\*k1, s12= q1\*k2,… etc. where q<- context and k <- sense embedding,  
followed by /sqrt(dk) and the softmax.

### Implementation

**1)** Given the logits of the globals for the current prediction, select the indices of those globals with the greatest k=100 logits.

**2)** The GNN also has a reference to the graph object. We want to retrieve the indices of the senses of the k=100 most likely globals.  
We need the indices of the immediate neighbours. So, n\_hops=1.

We do not need to check the edge\_index and edge\_type. Instead: does the index of the neighbour falls into the range of the senses in the embeddings matrix X, [0, last\_idx\_senses]?

**3)** Once wehave the indices of the senses of the most-likely-globals, we call X.index\_select(…) to retrieve their embeddings.

**4)** Multiply self.memory\_h2 x self.Wq to obtain the query (in this version, the query of the self-attention mechanism is always the same, derived from the current context).

Multiply self.likely\_senses\_embs x self.Wk, to project the senses embeddings into the keys of the self-attention mechanism (currently, we also change the dimensionality from 300 to 150).

*note/issue*: A global has, at minimum, 0 senses (not even 1), e.g. ‘for’ ,’of’ and any other stopwords that we do not find in the KB/dictionary.

Thus, in the self-attention mechanism, we may have: keys.shape=torch.Size([92, 100])

To solve it: pad it with 0s? But some scores are <0… I can index manually.

*note/issue*: at start, the matrix Wk has values ranging from 10^-38 to 10^38, and thus the matrix multiplication gives NaN. It is necessary to adjust its initialization.

**5)** Multiply query x keys, and divide by sqrt(dk) in order to obtain the attention logits.

**6)** Apply softmax on the <= k attention logits. We have now a probability distribution over the senses of the most likely globals.

**7)** We have to assign the self-attention scores to the selected senses, using their indices. All other senses in the vocabulary must have probability =0.

*Issue*:

I am assigning manually 0 to the sample’s predicted senses… however, in the globals and in the previous senses architecture, this element is not obtained with a softmax but with a tf.nn.functional.log\_softmax:

From the docs of pytorch, on tfunc.log\_softmax:  
“While mathematically equivalent to log(softmax(x)), doing these two operations separately is slower, and numerically unstable.  This function uses an alternative formulation to compute the output and gradient correctly.”

From the docs of pytorch, on torch.nn.LogSoftmax:

“[This function] returns a Tensor of the same dimension and shape as the input with values in the range [-inf, 0)”  
Hypothesis: I can simply assign -inf to “zero out” the senses that do not come from the most likely predicted globals?

No. loss\_sense=inf

Among the options of tfunc.nll\_loss(…):

**ignore\_index** (*python: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**

### Experiments

The preliminary experiment on a fragment of SemCor does not even manage to overfit on a small training set:

Training, end of epoch 300. Global step n.1200. Time = 47826.22. ¨

The training losses are:

Losses: Globals loss=4.19 Sense loss=61.69 Total loss=65.876

Perplexity: Globals perplexity=66.0

Sense perplexity=6.1632970018788446**e+26**

It is necessary to find another architecture.

## Architecture – IV

### Design

I think that the idea of selecting / considering the most likely globals should still be kept – it is a way to make the senses’ prediction influenced by the current global prediction.

Hypothesis:

There are a number of possible input signals for the senses’ task. :

* the most likely k globals
* the input signal of the GRU (current global’s word embedding || current global’s node state || current sense node state)
* a recurrent state of a GRU for the senses

Element n. 3 (memory, recurrent NN) could be implemented by having a (2-layer) GRU on the input signals n.1 and 2.

However, the most likely globals can easily be a matrix (k=500 x d=300). I can not apply a FF-NN directly on them.

# Meetings

## IA

* Alternative: simple selection from best globals, no self-attention, also works as estimate
* 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)

# After the meetings

## Senses Baseline: Operate on the k most likely globals

After we have the logits from the globals’ prediction, we can sort them descendingly and obtain the indices of the ***k* most likely globals**.

Then… for each of the *k* globals we retrieve the indices of the neighbours, and filter out those that are not in the range of senses?

~~Question: do we have a guarantee on how many senses we retrieve this way?~~

~~A global may have multiple senses in the dictionary, or only one, or even 0 (see: all the stopwords: ‘for’, ‘and’, ‘of’ etc.).~~

~~We do not have a guarantee.~~

~~It would be opportune to retrieve, let’s say, at most~~ *~~3\*k~~* ~~senses and have a mechanism that also works if we have strictly less than~~ ***~~3\*k~~* ~~senses~~**~~, by ignoring all 0/empty values.~~

~~Possible answer: create the logits for 3k senses. If we got less than 3k senses from the k most likely globals, fill with random sense indices. Apply log-softmax. Assign the log-softmax values to the output – in nll\_loss we can ignore part of the values.~~

Idea:

Filter the most likely k globals.

Send the logits over k most likely globals. k can be fixed or dynamic.

* + - look up: structured prediction.

There is a network that takes in input from the globals’ prediction and opens up to the senses’ logits over the whole vocabulary.

Hypotheses:

* take in the entirety of the globals’ logits, and have a FF-NN lead to the senses’ logits.  
  The dimensions of the Linear network would be: 21.9K x 25.9K =~ 567M parameters -> no.
* take in the embeddings of the k most likely globals, and then lead to the senses’ logits. (k=50 x d=300) -> 25.9K = 388M parameters, no.
* Make a structure with a globals-to-senses correspondence. If 1 (input) neuron == 1 global and 1(output, logit) neuron == 1 sense… I would need to create this structure beforehand, from the graph\_dataobject… wouldn’t it just return the most likely sense?
* represent the indices of the globals in 1-hot encoding… No. This would be worse than the first alternative that uses the logits as they are.
* Forcefully project the globals’ embeddings to a lower dimensionality, let’s say 300 -> 50. Then, (d=50++logit)\*k > senses’ vocabulary, which means (51\*50)\*25.9K =~67M
* Maybe, if I chose fewer globals? (20x300) x 25.9K = 155..., it could be viable.
* Passing on to the senses’ network: the globals’ projected embeddings, and their logits. I can choose relatively few globals and add an intermediate FF-NN.  
  (300 x 100) + (30x(100+1) x 25.9K = 6000 + 77.9M = 78.5M.

### Hypothesis 1: FF-NN – from all logits to all logits

Take in the entirety of the globals’ logits, and have a FF-NN lead to the senses’ logits.  
The dimensions of the Linear network would be: 21.9K x 25.9K =~ 567M parameters

*However,* we wish to limit the update to the logits of the first most likely *k* globals.

The globals’ logits, sorted descendingly:

tensor([tensor([0.0969, 0.0945, 0.0936, 0.0922, 0.0922, 0.0914, 0.0893, 0.0891, 0.0891,

0.0864,…, 0.0855, 0.0851, 0.0851,x -0.0871, -0.0877, -0.0887],)

And if we executed tfunc.softmax(logits\_global)?

We get a Tensor of 21988, that currently (in iteration number 1, just after the initialization) ranges from 5.0074e-05 to 4.1589e-05.

I can zero out all the softmax values that do not come from the most likely globals.

RunTimeError: CUDA out of memory. We can not have 600+M parameters.

### Hypothesis 2: Project embeddings of the first k globals + concat logit + FF-NN

* Select the first k globals with the greatest logits
* Extract their embeddings that have d=300, and use a FF-NN P to project them down to d\_p=50
* For each global, concatenate (projected embedding ++ logit value). We obtain a matrix PL of 20x51
* Flatten PL into a vector of d=1020. Apply a FF-NN from it to the ~26K senses’ logits

If I operate with the globals alone, that use the GRU-GAT4, I manage to overfit on a fragment of SemCor.

If I include this mechanism for the senses, I do not:

Perplexity: Globals perplexity=45.32 Sense perplexity=37.46

Let us try without the projection, simply concatenating the embeddings and the logits.

… More or less no improvement.

And what if I detached and clone-copied properly the embeddings of the globals, so that the sense task can not modify them?

I still need to project, or I run into CUDAoutOfMemory.

If I project with a higher dimension, like to d=100?

Considerably more interesting, to the point of being valid:

After 200 epochs at lr=0.001, with k=10 and dp=100,

Perplexity: Globals perplexity=1.39 Sense perplexity=4.59

After 500 epochs at lr=0.0001, with k=10 and dp=150?

It does NOT work: Perplexity: Globals perplexity=45.3 Sense perplexity=36.59

After 300 epochs at lr=0.001, with k=10 and dp=150,

Perplexity: Globals perplexity=1.01 Sense perplexity=1.06

## Senses: self-attention on the k most likely globals

### Design

Instead of trying to use as prediction the self-attention scores derived by queries and values on the senses of the likely globals,

I can operate with the full self-attention mechanism:

* **query**: the input and/or the context. Project it with Wq
* **keys**: either the k likely globals, or their senses. Project them with Wk
* **values**: the k likely globals/senses. Project them with Wv.

We computed self-attention scores from sofmax(Q\*K/sqrt(d\_k)).

Our result can be Σ(score\*value).

Finally, we use this result as the input of a FF-NN that leads to the senses’ logits.

### Implementation

### Overfit on mini-dataset

**Mini-experiment n.1**

Objective: Overfit on small training dataset (fragment of SemCor)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Training hyperparameters** | | **Input signals & GNN** | | **Architecture** | |
| batch\_size | 4 | grapharea | I: 32n | II: 181e | GRU layers | 2 |
| learning rate | 1\* 10^(-3) | Input signals | 1) current word embedding | hidden dim.s | 600,300 |
| TBPTT length | 16 | 2) current word node-state | Senses | Self-attention with Query=the 3 input signals and Keys&Values=*k* likely globals, + FF-NN to logits |
| GNN | 2 GATs, 4 heads |  | 3) current *sense* node-state |  |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity* | |
|  | Globals | Senses |
| 1 |  |  |
| 10 |  |  |
| 20 |  |  |
| 30 |  |  |
| 40 |  |  |
| 50 |  |  |
| 100 |  |  |
| 200 |  |  |
| 300 |  |  |

## Globals – LSTM

### Mini-experiment – overfit LSTM on fragment of WT2

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| GRU RNN  (see the full GRU formula) | 1) The word embedding of the current global | batch\_size=4 |
| 3 layers: 600,300,300 | TBPTT length=16 |
|  | learning rate=0.001 |
|  |  |

|  |  |
| --- | --- |
| **Epoch** | **Training perplexity** |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 10 |  |
| 30 |  |
| 50 |  |
| 75 |  |
| 100 |  |

### Experiment – LSTM on WT2

|  |  |  |
| --- | --- | --- |
| **Architecture** | **Input Signals** | **Hyperparameters** |
| GRU RNN  (see the full GRU formula) | 1) The word embedding of the current global | batch\_size=80 |
| 3 layers: | TBPTT length=35 |
|  | learning rate=10^(-5) |
|  |  |

|  |  |  |
| --- | --- | --- |
| *Epoch* | *Training perplexity* | *Validation perplexity* |
| 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 |  |  |
| 26 |  |  |
| 27 |  |  |
| 28 |  |  |
| 29 |  |  |
| 30 |  |  |
| 31 |  |  |
| 32 |  |  |
| 33 |  |  |
| 34 |  |  |
| 35 |  |  |
| 36 |  |  |
| 37 |  |  |
| 38 |  |  |
| 39 |  |  |
| 40 |  |  |
| 41 |  |  |
| 42 |  |  |
| 43 |  |  |
| 44 |  |  |
| 45 |  |  |
| 46 |  |  |
| 47 |  |  |
| 48 |  |  |
| 49 |  |  |
| 50 |  |  |
| 51 |  |  |
| 52 |  |  |
| 53 |  |  |
| 54 |  |  |
| 55 |  |  |
| 56 |  |  |
| 57 |  |  |
| 58 |  |  |
| 59 |  |  |
| 60 |  |  |