[1: Meeting (IA, 24/01/20) 2](#_Toc33003253)

[2: Attempting Batching 2](#_Toc33003254)

[Using DataLoader 3](#_Toc33003255)

[Issues 3](#_Toc33003256)

[Speedup: 5](#_Toc33003257)

[3: Experiments and Hyperparameters - I 6](#_Toc33003258)

[3.1: Full overfit on mini-dataset 6](#_Toc33003259)

[3.1.1. 6](#_Toc33003260)

[3.1.2 7](#_Toc33003261)

[3.1.3 8](#_Toc33003262)

[Next steps 8](#_Toc33003263)

[4: Modifications 9](#_Toc33003264)

[Visualizing predictions – Round 1 9](#_Toc33003265)

[Experiment 9](#_Toc33003266)

[Samples 10](#_Toc33003267)

[Observations 13](#_Toc33003268)

[Phrases 13](#_Toc33003269)

[Punctuation 13](#_Toc33003270)

[Visualizing predictions – Round 2 14](#_Toc33003271)

[Experiment 14](#_Toc33003272)

[Samples 15](#_Toc33003273)

[Observations 17](#_Toc33003274)

[Memory & Recurrence 18](#_Toc33003275)

[Current version: RGCN 18](#_Toc33003276)

[Observation: basis decomposition included by default 18](#_Toc33003277)

[Manual RGCN 19](#_Toc33003278)

[5: Batch normalization 20](#_Toc33003279)

[Introduction 20](#_Toc33003280)

[Method and observations 20](#_Toc33003281)

[Necessity of Batch Renormalization 21](#_Toc33003282)

# 1: Meeting (IA, 24/01/20)

ToDo list:

* complete batching
* lower the learning rate from 0.01 to 10^-3 or -4
* do not plot the loss for each batch, but instead the average training loss over the epoch
* make another overfit test. It should go down all the way to 0
* Include <UNK> in the processing. Initialized as the average of all globals at start
* RRGCN, recurrent RGCN, to add the RNN logic to the Language model task

# 2: Attempting Batching

In the current version, I do not have real, parallel batching.

I am using a for cycle on the elements of the batch:

**for** i **in** range(len(input\_indices\_lts) - 1):  
 (x, edge\_index, edge\_type) = batch\_rgcn\_input\_ls[i]  
  
 predicted\_globals, predicted\_senses = model(x, edge\_index, edge\_type)

…

Proposal: send batch\_rgcn\_input as the input to the model’s forward().

It depends on the input-to-the-forward-call for each node.

As of now, we collect it in a list for all nodes in the batch as follows:

forward\_input\_ls.append((area\_x, edge\_index, edge\_type))

**Problem**: forward\_input\_ls is a tuple of 3 tensors.

The shapes of (area\_x, edge\_index, edge\_type) are, respectively:

torch.Size([32, 300])  
torch.Size([2, 2048])  
torch.Size([2048])

As they are, they can not be stacked.

They could be padded with -1s, and stacked side-by-side:

torch.Size([32, 300]) 🡪 torch.Size([32, 300])  
torch.Size([2, 2048]) 🡪 torch.Size([32, 2048]) 🡪 torch.Size([32, 4396])  
torch.Size([2048]) 🡪 torch.Size([32, 2048])

Then, each element of the batch can be stacked vertically, thus obtaining a batch-dimension that can be used for parallel processing.

With batch\_size=8, obtain: torch.Size([8, 32, 4396])

## Using DataLoader

**Problem**: I am not able to specify that [8, 32, 4396] contains the 8 elements of a batch.

Considering: I have already the graphArea\_matrix, precomputed, that contains the graph-input for any word/node.  
I also have the training dataset.

**Choice**: construct a PyTorch DataLoader, that can take in 1 sample, or possibly more if I specify a batch size > 1.

This will also make the padding mechanism redundant, simplifying part of the code..

This involves the creation of a TextDataser(Dataset) class, that returns the next (X, y) item:  
the 3 input features (x, edge\_index, edge\_type) + the labels of the next token.

### Issues

{

Issues: Error operating on the H5. Possibility of throwing a wrong StopIteration exception.

The vocabulary of globals has the columns ‘word’ and ‘frequency’…

Hypothesis: the error on the vocabulary\_of\_globals.h5 is due to the parallel access of the DataLoader’s num\_workers > 1.

Information that supports the hypothesis: parallel hdf5 is a separate implementation:  
“Starting with version 2.2.0, h5py includes support for Parallel HDF5.  
Parallel HDF5 is a configuration of the HDF5 library which lets you share open files across multiple parallel processes. It uses the MPI (Message Passing Interface) standard for interprocess communication. .. This is accomplished through the mpi4py Python package ...”

Opinion that opposes the hypothesis from StackOverflow:  
“Parallel reads are fine with h5py, no need for the MPI version. But why do you expect a speed-up here? Your job is almost entirely I/O bound, not CPU bound. Parallel processes are not gonna help because the bottleneck is your hard disk, not the CPU. It wouldn't surprise me if parallelization in this case even slowed down the whole reading operation. Other opinions?”

Conclusion: if it is a way to avoid that HDF5 bug, I use 1 worker.}

{

The next issue is due to not being able to pass next\_token\_tuple properly, as an argument to the Dataset’s \_\_getitem\_\_(), when iterating over the DataLoader.

Maybe :

1. I can get the Dataset & DataLoader to return only the input, and just add manually the label. After all, the next\_token\_tuple does not need any processing
2. Modify the TextDataset, to keep a variable for the next token, and return it without any need for input.

}

{

Issue: I have an error because I am still throwing the Utils.MustSkipUNK\_Exception.

However, we added self-loops to the UNK nodes, so they are not disconnected and without edges (that was a cause of error) anymore.

I should review how they are handled, and initialized.

This covers one of the points-of-order from the last meeting, that was:   
“*•Include <UNK> in the processing. Initialized as the average of all globals at start*”

Error: “Raising Utils.MustSkipUNK\_Exception with word= Fulton County Grand Jury”  
I must redirect these cases to the <UNK> token

<unk> is, in fact, already present in the vocabulary\_of\_globals.h5}

{

When sending a batch\_dimension > 1 :

RuntimeError: invalid argument 0: Sizes of tensors must match except in dimension 0. Got 49 and 57 in dimension 2

Problem: the number of edges must be aligned. Considering 2 elements, we have:

area\_x.shape=torch.Size([32, 300])  
edge\_index.shape=torch.Size([2, 49])  
edge\_type.shape=torch.Size([49])

area\_x.shape=torch.Size([32, 300])  
edge\_index.shape=torch.Size([2, 57])  
edge\_type.shape=torch.Size([57])

Possible solutions:

1. Pad the vectors to the same dimension in Dataset, and select relevant elements in the forward()
2. Implement manually the def collate\_fn(data) function passed to the Dataloader. I could also use it as the point where to add padding.

}

Observation:

Apparently, I still need to implement the padding with -1 to a common size, in order to perform a stacking along the batch dimension (0).

What to do about the label tuple, a.k.a. the next token’s tuple e.g (238, 16015) ?

Major problem, that may undermine the concept of using batching with RGCN:  
The core call is: tF.relu(self.conv1(x, edge\_index, edge\_type))  
However, we are not able to send standardised edge\_index, because some nodes may have only few edges. Or few neighbours, in which case the number of rows in x will be << grapharea\_size. **No stacking --> no batching**

Error, in select\_valid\_features:  
return torch.stack(valid\_elems\_all)  
RuntimeError: invalid argument 0: Sizes of tensors must match except in dimension 0. Got 49 and 57 in dimension 1

### Speedup:

With the previous version, a batch size of 32 has an iteration time of 0.73/0.90s

Now, even if we do not have 100% proper batching because the RGCN layer call is not parallelized, thanks to the Dataset + DataLoader, and to the parallel input creation and loss computation, a batch size of 8 has an iteration time of 0.012/0.14s, and a batch of 32 has \_\_\_.

(I consider this an absolute win)

# 3: Experiments and Hyperparameters - I

## 3.1: Full overfit on mini-dataset

### 3.1.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| batch size | 8 |  | training tokens | 496 |
| graph\_area | 32 |  | epochs | 100 |
| learning rate | 0.001 |  | final global step | 6200 |
|  |  |  | token-steps | 49600 |



Final epoch nll\_loss= 4.158

However, my objective is to have a *full* overfit, with the loss value approaching 0.

I will try to:

* Increase the number of epochs
* Reduce the number of samples

### 3.1.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| batch size | 8 |  | **training tokens** | **176** |
| graph\_area | 32 |  | **epochs** | **300** |
| learning rate | 0.001 |  | final global step | 6600 |
|  |  |  | token-steps | 52800 |



Training, epoch nll\_loss= 2.21

### 3.1.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| batch size | 4 |  | **training tokens** | **128** |
| graph\_area | 32 |  | **epochs** | **1000** |
| **learning rate** | **0.001** |  | final global step | 32000 |
|  |  |  | token-steps | 26400 |



Training, epoch nll\_loss= 1.2242

Increasing the learning rate (e.g. 0.005) does not bring any improvement, only a bounce effect.  
The minimum training loss I can currently achieve, even trying to overfit on a very small dataset, is ~1.2

Even using grapharea\_size = 64 does not bring the nll\_loss < 1.

### Next steps

2 directions / modifications are needed:

1. Check the solution-tokens and the predicted globals&senses. The model may not be able to read/predict something, what is it?
2. Extend the vocabulary of globals. Check the current status of the vocabulary of senses as well.

# 4: Modifications

## Visualizing predictions – Round 1

### Experiment

Proceeding in reverse from the numerical indices, I can now log the predicted senses and globals.

I will print the predictions in the last epoch, after the model has stabilized, for instance in

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| batch size | 8 |  | training tokens | 128 |
| graph\_area | 32 |  | epochs | 100 |
| learning rate | 0.003 |  | final global step |  |
|  |  |  | token-steps |  |



Final nll\_loss = 2.20181

### Samples

Sentence, from the start of semcor.xml:

“The Fulton\_County\_Grand\_Jury said Friday an investigation of Atlanta 's recent primary\_election produced &quot no evidence &quot that any irregularities took\_place”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Global** | **Probability** |  | **Sense** | **Probability** |
| said | Solution |  | state.v.01 | solution |
| , | 38.93% |  | state.v.01 | 95.58% |
| said | 38.03% |  | produce.v.04 | 2.17% |
| <unk> | 11.38% |  | far.r.02 | 0.7% |
| and | 3.21% |  | person.n.01 | 0.46% |
| which | 3.2% |  | mission.n.03 | 0.38% |

|  |  |
| --- | --- |
| Sample | Comment |
| **Label: the next global is: Friday**  INFO : The top- 5 predicted globals are:  **INFO : Word: , ; probability = 41.48%**  **INFO : Word: in ; probability = 33.9%**  **INFO : Word: Friday ; probability = 23.78%**  INFO : Word: . ; probability = 0.37%  INFO : Word: was ; probability = 0.26% | The comma ends up being the first prediction.  Occurrences in the text=3: said Friday said in said , |
| **Label: the next global is: an**  INFO : The top- 5 predicted globals are:  **INFO : Word: an ; probability = 100.0%** | v |
| **Label: the next global is: investigation**  **INFO : Label: the next sense is: probe.n.01**  INFO : The top- 5 predicted globals are:  **INFO : Word: investigation ; probability = 97.0%**  INFO : Word: of ; probability = 2.18%  INFO : Word: by ; probability = 0.34%  INFO : Word: was ; probability = 0.19%  INFO : Word: <unk> ; probability = 0.16%  INFO : The top- 5 predicted senses are:  **INFO : Sense: probe.n.01 ; probability = 98.99%** | v |
| **Label: the next global is: of**  INFO : The top- 5 predicted globals are:  **INFO : Word: <unk> ; probability = 32.29%**  **INFO : Word: of ; probability = 31.81%**  INFO : Word: the ; probability = 11.58%  INFO : Word: said ; probability = 7.53%  INFO : Word: in ; probability = 5.38% | <unk> versus of.  ‘of’ should have been the only reasonable alternative.  Occurrences in the text=1:  ‘investigation of‘  Issue of the globals-prediction system. By the way, what is the node ‘of’ connected to? |
| **Label: the next global is: Atlanta**  INFO : The top- 5 predicted globals are:  **INFO : Word: the ; probability = 35.87%**  INFO : Word: possible ; probability = 17.19%  INFO : Word: such ; probability = 16.8%  INFO : Word: voters ; probability = 14.04%  **INFO : Word: Atlanta ; probability = 12.82%** | We observe that Atlanta is too low after 100 epochs.  Occurrences in the text=5: of Atlanta [Atlanta=1] of the (x2) [the=15] of possible [possible=1] of such [such=1] of voters [voters=1]  All alternatives are represented. It is interesting to notice that words that are more common in language are more prominent even when their frequency in the training text is identical  (possible > such > voters > Atlanta) |
| **Label: the next global is: s**  INFO : The top- 5 predicted globals are:  **INFO : Word: s ; probability = 99.93%** | v |
| **Label: the next global is: recent**  **INFO : Label: the next sense is: late.s.03**  INFO : The top- 5 predicted globals are:  **INFO : Word: recent ; probability = 92.28%**  INFO : Word: . ; probability = 2.38%  INFO : Word: , ; probability = 1.31%  INFO : Word: term ; probability = 1.12%  INFO : Word: <unk> ; probability = 0.94%  INFO : The top- 5 predicted senses are:  **INFO : Sense: late.s.03 ; probability = 96.5%**  INFO : Sense: term.n.02 ; probability = 1.79% | v |
| **Label: the next global is: <unk>**  **INFO : Label: the next sense is: primary.n.01**  INFO : The top- 5 predicted globals are:  **INFO : Word: <unk> ; probability = 99.99%**  INFO : Word: reports ; probability = 0.01%  INFO : Word: in ; probability = 0.0%  INFO : Word: a ; probability = 0.0%  INFO : Word: September ; probability = 0.0%  INFO : The top- 5 predicted senses are:  **INFO : Sense: primary.n.01 ; probability = 99.98%** | primary\_election is evaluated as a <unk>  ISSUE: there is a discrepancy between the phrases in the sense-labeled corpus and a vocabulary of globals.  Maybe it is necessary to try out the vocabulary from SLC. Or to build phrases in the vocabulary from WikiText |
| **Label: the next global is: produced**  **INFO : Label: the next sense is: produce.v.04**  INFO : The top- 5 predicted globals are:  **INFO : Word: which ; probability = 44.2%**  **INFO : Word: produced ; probability = 40.69%**  INFO : Word: , ; probability = 5.12%  INFO : Word: <unk> ; probability = 2.47%  INFO : Word: said ; probability = 2.27%  INFO : The top- 5 predicted senses are:  **INFO : Sense: produce.v.04 ; probability = 94.99%**  INFO : Sense: state.v.01 ; probability = 4.2% | ‘produced’ follows a <unk>.  We have no other information, other than the likelihood of a word to follow <unk> |
| **Label: the next global is: <unk>**  INFO : The top- 5 predicted globals are:  **INFO : Word: <unk> ; probability = 79.9%**  INFO : Word: was ; probability = 5.18%  INFO : Word: the ; probability = 2.58%  INFO : Word: of ; probability = 2.48%  INFO : Word: by ; probability = 2.1% | This is the ‘&quot’ HTML symbol.  It should be turned into “  Then, it will be covered by our decision regarding punctuation. |
| -**Label: the next global is: no**  INFO : The top- 5 predicted globals are:  **INFO : Word: <unk> ; probability = 21.16%**  INFO : Word: The ; probability = 10.93%  INFO : Word: that ; probability = 9.87%  INFO : Word: primary ; probability = 6.34%  INFO : Word: in ; probability = 6.29%  INFO : | I am, apparently, unable to predict this global word after a <unk> |
| **Label: the next global is: evidence**  **INFO : Label: the next sense is: evidence.n.01**  INFO : The top- 5 predicted globals are:  **INFO : Word: the ; probability = 34.81%**  INFO : Word: <unk> ; probability = 23.1%  INFO : Word: handful ; probability = 15.71%  INFO : Word: which ; probability = 13.39%  **INFO : Word: evidence ; probability = 11.14%**  INFO : The top- 5 predicted senses are:  **INFO : Sense: handful.n.01 ; probability = 57.37%**  **INFO : Sense: evidence.n.01 ; probability = 40.85%**  INFO : Sense: potential.a.01 ; probability = 0.75%  INFO : Sense: such.s.01 ; probability = 0.75%  INFO : Sense: conduct.v.01 ; probability = 0.09% | In the text, we have only one “no”, followed by “evidence”.  Why does the global prediction fail?  Rare event here: the sense-prediction system fails. |
| … |  |
| **Label: the next global is: praise**  **INFO : Label: the next sense is: praise.n.01**  INFO : The top- 5 predicted globals are:  **INFO : Word: election ; probability = 24.7%**  **INFO : Word: <unk> ; probability = 23.24%**  INFO : Word: widespread ; probability = 9.07%  INFO : Word: number ; probability = 8.91%  INFO : Word: size ; probability = 8.77%  INFO : The top- 5 predicted senses are:  **INFO : Sense: election.n.01 ; probability = 25.94%**  INFO : Sense: size.n.01 ; probability = 12.53%  INFO : Sense: manner.n.01 ; probability = 11.47%  INFO : Sense: location.n.01 ; probability = 11.43%  **INFO : Sense: praise.n.01 ; probability = 11.37%** | ‘praise’ follows ‘the’.  Since ‘the’ appears many times, we will have a widespread probability of having different next words.  ‘the election’ is found repeatedly, and this is mirrored in the prediction. |
| **Label: the next global is: and**  INFO : Label: the next sense is: None  INFO : tensor([ 41, 25, 65, ..., 2180, 13499, 7587], device='cuda:0')  INFO : The top- 5 predicted globals are:  **INFO : Word: and ; probability = 99.77%**  INFO : Word: . ; probability = 0.14%  INFO : Word: of ; probability = 0.03%  INFO : Word: the ; probability = 0.03%  INFO : Word: October ; probability = 0.02% | v |
| **Label: the next global is: thanks**  **INFO : Label: the next sense is: thanks.n.01**  INFO : The top- 5 predicted globals are:  **INFO : Word: the ; probability = 64.9%**  **INFO : Word: thanks ; probability = 34.91%**  INFO : Word: was ; probability = 0.14%  INFO : Word: and ; probability = 0.03%  INFO : Word: charge ; probability = 0.02%  INFO : The top- 5 predicted senses are:  **INFO : Sense: thanks.n.01 ; probability = 99.96%**  INFO : Sense: mission.n.03 ; probability = 0.03%  INFO : Sense: probe.n.01 ; probability = 0.01%  INFO : Sense: potential.a.01 ; probability = 0.0%  INFO : Sense: such.s.01 ; probability = 0.0% | ‘and’ has n.occurrences=2.  ‘and thanks’, ‘and the’ |
| … |  |
| **Label: the next global is: October**  INFO : Label: the next sense is: None  INFO : The top- 5 predicted globals are:  **INFO : Word: October ; probability = 91.28%**  INFO : Word: and ; probability = 2.84%  INFO : Word: , ; probability = 1.09%  INFO : Word: . ; probability = 1.07%  INFO : Word: to ; probability = 1.0% | v |
| **Label: the next global is: term**  **INFO : Label: the next sense is: term.n.02**  INFO : The top- 5 predicted globals are:  **INFO : Word: term ; probability = 88.13%**  INFO : Word: which ; probability = 6.36%  INFO : Word: recent ; probability = 1.78%  INFO : Word: to ; probability = 1.62%  INFO : Word: October ; probability = 0.77%  INFO : The top- 5 predicted senses are:  **INFO : Sense: term.n.02 ; probability = 97.7%**  INFO : Sense: late.s.03 ; probability = 1.19%  INFO : Sense: produce.v.04 ; probability = 0.79%  INFO : Sense: overall.s.02 ; probability = 0.27%  INFO : Sense: state.v.01 ; probability = 0.04% | v |

### Observations

Since we have no memory for the context, we end up computing the probability of obtaining the next word W(t+1)=*b* after W(t)=*a*.

This probability is influenced by:

1. The number of occurrences of “… *a* *b* …” in the text, like a n-gram language model.
2. The frequency of *a* and *b* in the text
3. The centrality/”mainstream nature” of the possible completions in *b*(e.g. possible > such > voters > Atlanta)

Issues that emerge from this experiment:

* **Context**: we should be able to use as a base more than 1 word. Recurrence / memory mechanism (e.g. RNNs) needed
* **Punctuation**: the ‘&quot’ HTML symbol is seen as a <unk>. It should be turned into “. Then, it will be covered by our decision regarding punctuation
* **Phrases**: there is a discrepancy between the phrases in the sense-labeled corpus and a vocabulary of globals. Maybe it is necessary to try out the vocabulary from SLC. Or to build phrases in the vocabulary from WikiText

## Phrases

We use the vocabulary from the Sense-LabeledCorpus itself instead of the one from WikiText-2.

Now: primary\_election is still not present, most probably because of the low frequency. However,we find took\_place, pointed\_out and others.

## Punctuation

Q: If I take the vocabulary from the SLC, does it mean that the HTML-encoded elements from punctuation (e.g. &quot) are already present?

If so, then all that is needed is to turn them into symbols when printing the predictions.

“ Atlanta’s “

The vocabulary from SLC reads [Atlanta], [‘s]

The RGCN has been reading [Atlanta], [s]

----> I should keep the apostrophe

Examining: MyRGCN.train() -> DataLoading.TextDataset ->

self.generator = SLC.read\_split(self.split\_name) -> SLC. dataset\_generator(xml\_fpath)

Output from the generator on the training dataset:

Out[16]: {'surface\_form': 'produced', 'lemma': 'produce', 'pos': 'VBD', 'wn16\_key': 'produce%2:39:01::', 'wn30\_key': 'produce%2:39:01::'}

gen.\_\_next\_\_()

Out[17]: {'surface\_form': '"', 'pos': "''"}

The SLC-vocabulary does not have anything for [&quot], nor for [“]

It is the vocabulary’s fault.

Commas and punctuation signs are present in the Sense-Labeled Corpus as separate tokens.

Therefore, it makes sense to keep them in the vocabulary, and thus in the graph, as globals.

I have no way to connect them to anything else, though, apart from the self-loop.

They will be placed and then trained based on reading the text corpus – no input from dictionary sources.

With min\_count=10 (from here on it will be changed to min\_count=5), we have a relatively limited number of globals. In the graph:

X\_definitions.shape=torch.Size([13046, 300])

X\_examples.shape=torch.Size([16200, 300])

X\_senses.shape=torch.Size([13046, 300])

X\_globals.shape=torch.Size([5528, 300])

And the graph-

dataObject will be:

Data(edge\_index=[2, 54080], edge\_type=[54080], node\_types=[47820],  
 num\_relations=[1], x=[47820, 300])

## Visualizing predictions – Round 2

### Experiment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| batch size | 8 |  | training tokens | 128 |
| graph\_area | 32 |  | epochs | 100 |
| learning rate | 0.003 |  | final global step | 1600 |



Training, final nll\_loss= 2.44262

(Since we include the punctuation, we have more symbols to choose from and we need more epochs to overfit)

### Samples

|  |  |
| --- | --- |
| Sample | Comment |
| **Label: the next global is: said**  **INFO : The top- 5 predicted globals are:**  **INFO : Word: , ; probability = 33.88%**  **INFO : Word: said ; probability = 31.93%**  INFO : Word: further ; probability = 9.0%  INFO : Word: <unk> ; probability = 8.13%  INFO : Word: had ; probability = 6.35%  **INFO : Label: the next sense is: state.v.01**  **INFO : The top- 5 predicted senses are:**  **INFO : Sense: state.v.01 ; probability = 82.11%**  INFO : Sense: far.r.02 ; probability = 13.1%  INFO : Sense: person.n.01 ; probability = 3.76%  INFO : Sense: produce.v.04 ; probability = 0.72%  INFO : Sense: late.s.03 ; probability = 0.13% | The comma is the first global predicted. It happened already in the previous version |
| **Label: the next global is: Friday**  **INFO : The top- 5 predicted globals are:**  **INFO : Word: , ; probability = 38.21%**  **INFO : Word: in ; probability = 34.14%**  **INFO : Word: Friday ; probability = 27.1%**  INFO : Word: recent ; probability = 0.14%  INFO : Word: was ; probability = 0.13% | Same as previous version |
| **Label: the next global is: an**  INFO : The top- 5 predicted globals are:  **INFO : Word: an ; probability = 98.09%** | v  Same |
| **Label: the next global is: investigation**  **INFO : Label: the next sense is: None**  **INFO : The top- 5 predicted globals are:**  **INFO : Word: investigation ; probability = 97.76%**  INFO : Word: an ; probability = 0.94%  INFO : Word: jury ; probability = 0.64%  INFO : Word: <unk> ; probability = 0.24%  INFO : Word: of ; probability = 0.22% | The next global is correct, as it was before.  *However*, why are we not recognizing ‘probe.s.01’ as the sense solution, as we did before? |
| **Label: the next global is: of**  **INFO : The top- 5 predicted globals are:**  **INFO : Word: September ; probability = 28.37%**  **INFO : Word: of ; probability = 24.26%**  INFO : Word: jury ; probability = 23.94%  INFO : Word: <unk> ; probability = 22.95%  INFO : Word: investigation ; probability = 0.32% | previously, we predicted the globals: <unk>, of, the, said, in |
| **Label: the next global is: Atlanta**  **INFO : Label: the next sense is: None**  **INFO : The top- 5 predicted globals are:**  INFO : Word: the ; probability = 30.69%  INFO : Word: possible ; probability = 15.35%  INFO : Word: such ; probability = 14.83%  INFO : Word: voters ; probability = 13.66%  INFO : Word: s ; probability = 12.69% | Worse than previously, when it was [the, possible, such, voters, Atlanta]. Probably due to overfitting less? |

|  |  |
| --- | --- |
| **Label: the next global is: s**  **INFO : Label: the next sense is: None**  **INFO : The top- 5 predicted globals are:**  INFO : Word: the ; probability = 30.69%  INFO : Word: possible ; probability = 15.35%  INFO : Word: such ; probability = 14.83%  INFO : Word: voters ; probability = 13.66%  **INFO : Word: s ; probability = 12.69%** | We do not see the apostrophe. Is it not printed, or is it just ignored?  Originally, it was: INFO : Word: s ; probability = 99.93% |
| **Label: the next global is: recent**  **INFO : Label: the next sense is: late.s.03**  INFO : The top- 5 predicted globals are:  **INFO : Word: recent ; probability = 61.05%**  INFO : Word: of this ; probability = 16.86%  INFO : Word: " ; probability = 7.7%  INFO : Word: the ; probability = 2.89%  INFO : Word: of ; probability = 2.27%  **INFO : The top- 5 predicted senses are:**  INFO : Sense: late.s.03 ; probability = 92.13%  INFO : Sense: jury.n.01 ; probability = 2.17%  INFO : Sense: end.n.02 ; probability = 1.37%  INFO : Sense: produce.v.04 ; probability = 1.3%  INFO : Sense: primary.n.01 ; probability = 0.82% | v, as before |
| **Label: the next global is: <unk>**  **INFO : Label: the next sense is: primary.n.01**  **INFO : The top- 5 predicted globals are:**  **INFO : Word: <unk> ; probability = 99.23%**  INFO : Word: " ; probability = 0.28%  INFO : Word: was ; probability = 0.25%  INFO : Word: of this ; probability = 0.09%  INFO : Word: in ; probability = 0.06%  **INFO : The top- 5 predicted senses are:**  **INFO : Sense: primary.n.01 ; probability = 99.73%**  INFO : Sense: happen.v.01 ; probability = 0.14%  INFO : Sense: jury.n.01 ; probability = 0.07%  INFO : Sense: end.n.02 ; probability = 0.03%  INFO : Sense: conduct.v.01 ; probability = 0.01% | As before   primary\_election is evaluated as a <unk> due to low frequency  The primary.n.01 sense is still correct |
| **Label: the next global is: produced**  **INFO : Label: the next sense is: produce.v.04**  **INFO : The top- 5 predicted globals are:**  **INFO : Word: which ; probability = 30.99%**  **INFO : Word: produced ; probability = 28.51%**  INFO : Word: " ; probability = 10.24%  INFO : Word: had ; probability = 6.73%  INFO : Word: <unk> ; probability = 4.33%  **INFO : The top- 5 predicted senses are:**  **INFO : Sense: produce.v.04 ; probability = 65.75%**  INFO : Sense: person.n.01 ; probability = 21.68%  INFO : Sense: far.r.02 ; probability = 8.04%  INFO : Sense: state.v.01 ; probability = 2.76%  INFO : Sense: late.s.03 ; probability = 0.94% | as before: ‘which’ and ‘produced’ are nearly tied as the predicted global. The sense is ok |
| **Label: the next global is: "**  **INFO : Label: the next sense is: None**  **INFO : The top- 5 predicted globals are:**  **INFO : Word: " ; probability = 25.56%**  INFO : Word: of ; probability = 19.37%  INFO : Word: the ; probability = 13.3%  INFO : Word: was ; probability = 7.77%  INFO : Word: end ; probability = 7.51% | New, and correct |
| **Label: the next global is: no**  **INFO : Label: the next sense is: None**  **INFO : The top- 5 predicted globals are:**  INFO : Word: , ; probability = 11.24%  INFO : Word: that ; probability = 10.73%  INFO : Word: in ; probability = 9.91%  INFO : Word: Only ; probability = 9.65%  INFO : Word: for ; probability = 9.6% | As before: I am, apparently, unable to predict this global word after a <unk> |

### Observations

**Why we do not have ‘probe.s.01’ as the sense solution, as we did before?**

Using the SLCvocabulary, the indices\_table.sql file is of size 442 KB instead of 647 KBs.

We retrieve the senses from WordNet, using as a base the vocabulary of globals…

Hypothesis:

Fewer globals -> fewer senses

|  |  |  |
| --- | --- | --- |
| WT-2 Vocab, min\_f=10 | SLC Vocab, min\_f=10 | SLC Vocab, min\_f=5 |
| X\_definitions.shape=  torch.Size([19008, 300])  X\_examples.shape=  torch.Size([20191, 300])  X\_senses.shape=  torch.Size([19008, 300])  X\_globals.shape=  torch.Size([13675, 300]) | X\_definitions.shape=  torch.Size([13046, 300])  X\_examples.shape=  torch.Size([16200, 300])  X\_senses.shape=  torch.Size([13046, 300])  X\_globals.shape=  torch.Size([5528, 300]) | X\_definitions.shape=  torch.Size([17843, 300])  X\_examples.shape=  torch.Size([20088, 300])  X\_senses.shape=  torch.Size([17843, 300])  X\_globals.shape  =torch.Size([9858, 300]) |
| Tot.nodes = 71882 | Tot.nodes =47800 | Tot.nodes = 65632 |

We assume that this solves the problem of not containing “probe”, which prevents us from including ‘probe.s.01’. A later check will follow.

No, we still do not have probe, but we have added other senses. During the development process, I keep it as it is.

# Memory & Recurrence

## Current version: RGCN

We recall the rules of the current version, that handles multiple types of edges but has no recurrence:

The update rule to the node state for basic GCNs is:

The update-&-propagation model for the R-GCN is the following:

“ where Nir denotes the set of neighbor indices of node i under relation r.

cir can be =|Nir|

Different from regular GCNs, we introduce relation-specific transformations, i.e. depending on the type and direction of an edge.

To ensure that the representation of a node at layer (l + 1) can also be informed by the corresponding representation at layer (l), we add a single self-connection () of a special relation type 0 to each node in the data.

A neural network layer update consists of evaluating in parallel for every node in the graph.”

### Observation: basis decomposition included by default

The original aim of RGCNs was to deal with Knowledge Bases, that can easily have thousands of relations/edge types.

With this aim in mind, they are meant to have block-diagonal and basis decompositions, to decrease the number of parameters.

The pre-implemented version of RGCN in PyTorch-Geometric includes, mandatorily, a basis-decomposition:

* each weights matrix for a relation r on layer l, , is defined as follows:

as a linear combination of a lower number of common Bases *Vb*, where only the coefficients depend on *r*.

I set #*Vb*-s = 5, since I do not need this feature.

**Idea:**

I could implement the RGCN formula myself, with the aim of streamlining it, avoiding the basis decomposition, and preparing the ground for the introduction of recurrence.

**Observations on recurrence:**

Regarding recurrence and the forward() call:

I need to include in the input matrix **x** all the nodes that are involved in the batch.

Batch = text window = BPTT window.

**x**: must contain all the area nodes for the words in the batch.

### Manual RGCN

The batchinput\_ls contains elements (x, edge\_index, edge\_type).

When using the pre-made implementation, we have:

rgcn\_conv = self.conv1(x, edge\_index, edge\_type)

x\_Lplus1 = tfunc.relu(rgcn\_conv)

The weight matrices are . We must have:

* 1 W per layer. We need only 1 if we keep the previous architecture, of:  
  input > relu(rgcn\_conv) > representation > 2 linear FF-NNs > (logits\_global, logits\_sense)
* 1 W per edge type
* 1 W for the direct connection from the previous level

It is necessary to split the edge\_index, edge\_type into |R|=5 (def, ex, sc, syn, ant) adjacency matrices.

Then, and

**Observation**:

The way the product is written in the formulas is actually misleading with respect to the order of factors and the dimensions.

i.e. in

we have that must have the same dimensions as , since it is added to it (before the ReLu that does not change the dimensionality)

W : (d x d) , and is (d x 1) --> the only way to have a result of (d x 1) is to execute the product using as a **column**. We are effectively summing up the values of , depending on the weights.

This is corroborated by the github implementations of GCNs.

Currently, my rgcn\_convolution(H, Ar\_all, W\_all) uses a for-cycle on the *Ar* matrices that store the neighbourhoods for a given edge type *r*.

For each relation, we execute a standard GCN convolution, divided by a normalization constant . Then, they are summed up.

### Split by relation into subgraphs

It is also necessary to implement an instrument that splits the input of the pre-made RGCN call, (x, edge\_index, edge\_type), into K=5 different adjacency matrices.

This can be used both with our manual RGCN and

x.shape = (32, 300). We have a graph\_area\_size of 32 nodes around the current selected one. d = 300.

edge\_index.shape = (2,45). In general, (2, num\_edges).

It is made of (vector\_of\_sources, vector\_of\_destinations). If we transpose it, we will have the classic form [(source1,dest1),…, ]:

tensor([[ 1, 0],

[20, 7],

[11, 4],

[15, 5], …)

edge\_type.shape = (45). In general, (num\_edges).

e.g.: tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 3, 3 …

The vector specifying the type of edges. We should use it to perform the split.

Note: I am not using any bias term currently (set to 0s in the GCN convolution).

### Using PyTorch-Geometric tools

# 5: Batch normalization

## Introduction

Originally, when batch normalization was introduced, it was believed that it could mitigate the problem of internal covariate shift:  
During the training stage of networks, as the parameters of the preceding layers change, the distribution of inputs to the current layer changes accordingly, such that the current layer needs to constantly readjust to new distributions.

In 2018, researchers have found that batch normalization does not reduce internal covariate shift, but rather smooths the objective function to improve the performance.

## Method and observations

Batch normalization can be implemented during training by calculating the mean and standard deviation of each input variable to a layer per mini-batch and using these statistics to perform the standardization.

**Caveat:**

For small [mini-batch sizes](https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/) or mini-batches that do not contain a representative distribution of examples from the training dataset, the differences in the standardized inputs between training and inference (using the model after training) can result in noticeable differences in performance.

This can be addressed with a modification of the method called Batch Renormalization (or BatchRenorm for short) that makes the estimates of the variable mean and standard deviation more stable across mini-batches.

**Tip**:

Batch normalization may be used on the inputs to the layer *before* or *after* the activation function in the previous layer.

It may be appropriate **before** the activation function for activations that may result in non-Gaussian distributions like the rectified linear activation function, the modern default for most network types.

**Consequence**:

The network is more stable during training.

We can use higher learning rates because batch normalization makes sure that there is no activation that has gone really high or really low. (n: it is opportune to increase the decay rate for the learning rate, as well).

**Note**:

Further, it may not be a good idea to use batch normalization and dropout in the same network.

The reason is that the statistics used to normalize the activations of the prior layer may become noisy given the random dropping out of nodes during the dropout procedure.

## Necessity of Batch Renormalization

From a discussion on the PyTorch forum:

“For the past few days, I’ve been training a model that uses batch normalization. While this normalization is crucial to speed up training, performance drops severely once I switch to eval instead of train mode…

The problem seems to be caused by the fact the running estimates are not reliable when using small batch sizes. For a lot of problems (e.g. segmentation), however, increasing the batch size is not feasible due to memory constraints…

The authors of the batch normalization paper acknowledged this issue and wrote a follow-up paper about batch renormalization, a similar technique which should also work with smaller batches.

I was wondering if there were any plans to implement this batch renormalization in PyTorch?”

Answer:” You could always try instance norm (unless you have really few features per channel), which takes away the difference between training and evaluation…”