Commonsense Knowledge Graph Completion

Chunhua Liu

July 11, 2020

Contents

Chapter 1

2020-07-07

Check list for data creation

1. Sampling order
2. Ensure the nodes and triples num to be close
3. Output degree of the test and valid graph
4. Output the entity frequency on sampled graph, SWOW and ConceptNet
5. If h and t hold n relations, add all the relations to SWOW
6. Ensure the content of the final written triples is consistent with the orginal one

1.1 Dataset Creation

1.1.1 Statistics of Dataset

node degree distribution of valid and test learning curves on current model (current versionm, still need to debug)

1.1.2 Procedures of creating dataset

Category	File name	Triples	Entities	Relations	avg.Degree	Density
CN-100K	cn_100k_train_valid_test	102,400	78,334	34	1.275	1.7e-5
SWOW	swow_freq2.filter	413,481	34,829	1	11.871	3.4e-4
Overlap	overlap_triples	8,122	4,006	33		
Overlap	overlap_entities		8,382			
Train	rel_triples1	100,162	78,173	34	1.251	1.6e-5
Train	rel_triples2	413,504	34,813	34	11.864	3.41e-4
Train	ent_links_train		6,600			
Train	rel_links			33		
Test	rel_triples_test	1,125	889	27	1.165	1.42e-3
Test	ent_links_test		889			
Valid	rel_triples_valid	1,110	893	29	1.159	1.39e-3
Valid	ent_links_valid		893			

Table 1.1: Data Statistics for newly created dataset.

Algorithm 1: Procedures of creating aligned dataset

Data: Triples from ConceptNet and SWOW
Result: Triples for train $\mathcal{T}_{tr}$, valid $\mathcal{T}_{v}$, test $\mathcal{T}_{te}$
Result: Overlap entities for train, valid, test alignment task
Result: Overlap relations for train, valid, test

```
Initilize overlap triples T_o = set()
Initilize overlap nodes N_o = set()
Build a directional CN-Graph G_c
Build a directional SW-Graph G_s
for (h, t) ← G_s do
    if (h, t) ∈ G_c then
        for r ← G_c(h, t) do
            T_o ← (r, h, t)
            N_o ← (h, t)
            G_s ← (r, h, t)
        end
    end
end
Sample 3000 nodes from N_o and shuffle them
Split 3000 nodes into 2 sets, N_t, N_v for sampling valid and test respectively
for (r, h, t) ∈ E_o do
    if (h, t) ∈ N_t then
        N_t ← (r, h, t)
    end
    if (h, t) ∈ N_v then
        N_v ← (r, h, t)
    end
end
```

Code Checklist

1. Data
   (a) ~~Generate negative samples for test and valid triples~~
2. Evaluation
   (a) Compare my code with the others's to see whether there is any different
   (b) ☑Add filter rank code.
   Purpose of filtered rank: For a given (h, r) in the test set, we will compute the score with all the other entitities in the cadidate pool, e.g., (h, r, t'), which could possibly be a true triple in the train, valid, test sets, and gets a higher score than the golden t, and lower the rank of t, which is unfair. So the way of removing all the possible true triples from the corrupted triple list is called *filtered rank*, and the other one is called *raw rank*. (see explinations from **??**)
   (c) ☑Debug the code of restoring a trained model for re-testing relation classifier
   (d) ☑Measure alignment in both directions
   (e) Evaluate the dev and test with mini-batch
3. Training
   (a) ☑Set FLAG for training with maxinum epoch or early stoping
   (b) ☑Use $0.5*(align\_hits1+completion\_hits1)$ or $0.5*(align\_mrr+completion\_mrr)$ as the early stopping criteria
   (c) ☑If early stopping, save the model with the best performance
   (d) If early stopping, restore and test the saved model with the best performance

(e) If early stopping, visualize the curves of stopping criteria

(f) Sort out code, make sure the alignment and completion code to be consistent

(g) Add regularizatons for weight parameters in relation classifier

4. Model

(a) Make use of both embeddings learnt from CN and SW when doing evaluation.

Chapter 2

2020-06-23

Check List

1. ☑Figure out why 'NAN' appears during training.
   A: the scores for some relation types are too small and caused the underflow when apply log function to them. A small number $\epsilon = 1e-9$ is added to the scores for numerical stable.

2. Debug the relation classifier
   For now, when β_1 is increased, the relation prediction accuracy become higher, but the alignment and completion performances dropped. Try to figure out why, the code problem or the idea problem? Try to make the model and task to be simple to debug the relation classifier.

   (a) ☑Try simpler version of dataset, e.g., two relaiton types.

   (b) Observe some prediction outputs for alignment rank, completion rank, and relation prediction (both training and valid).

3. Relation classification overfitting: try different training strategy to alleviate the problem.

   (a) ☑Play with hyperparameters: lr, batch_size, and training_epoch.

   (b) ☑Pretrain the relation classifier.

   (c) ☑Train the relation classifier every K ($K > 1$) epoch instead of every epoch.

2.1 Train 2 relation types

When I was doing the experiments of learning only two relation types and visualizing the correlations between rel-acc and other metrics (hits@N, mr, mrr), I noticed that the prvious conclusion: "**when rel-acc goes up, the other metrics go down**" is not completely correct (see Figure **??**). The vibration at high rel-acc is because of more datapoints. From these curves, we can see when rel-acc is high, the other metrics could either high or low. It seems that there is no positive relationships between rel-acc and other metrics. Although I haven't figure out why, it gives me a hint that I can train them separately. So, I made the following modifications for training the relation classifier:

1. Separate the training process of 1) learning entity and relation embedding matrices, and 2) learning relation types. Then two optimizers are used to train two parts separately (previous: losses are added into one total loss and train it with one optimizer.)

2. The relation classifier is trained along with the embedding matrices at first N epoches. After that, it's updated every K epoches to avoid overfitting. This can ensure the relation classifier is well trained and also reduce the overfitting problem (see Algorithm **??**).

The key difference is the gradients from the margin loss L_e for triples (see Figure **??**) to the parameter θ in the relation classifier are cut down. So θ is only updated by $L_r and L_p$.

Algorithm 2: Asynchronously trainining embedding matrices and relation types

Input: Triples from ConceptNet and SWOW
Input: Aligned entity seeds
Input: N, M signify the max trining epoch for L_e and L_{rp}, $N \geq M$. K denotes an int number.
Result: Entity embedding matrix $\mathbf{E}$, relation embedding matrix $\mathbf{R}$
Result: Relation type classifier with parameter θ
initialization;
for $i \leftarrow 0$ *to* N **do**
  if $i < M$ **then**
    Update θ
  else
    if $i \geq M \vee i\%K = 0$ **then**
      Update θ
    end
  end
  Update $\mathbf{E}, \mathbf{R}$
end

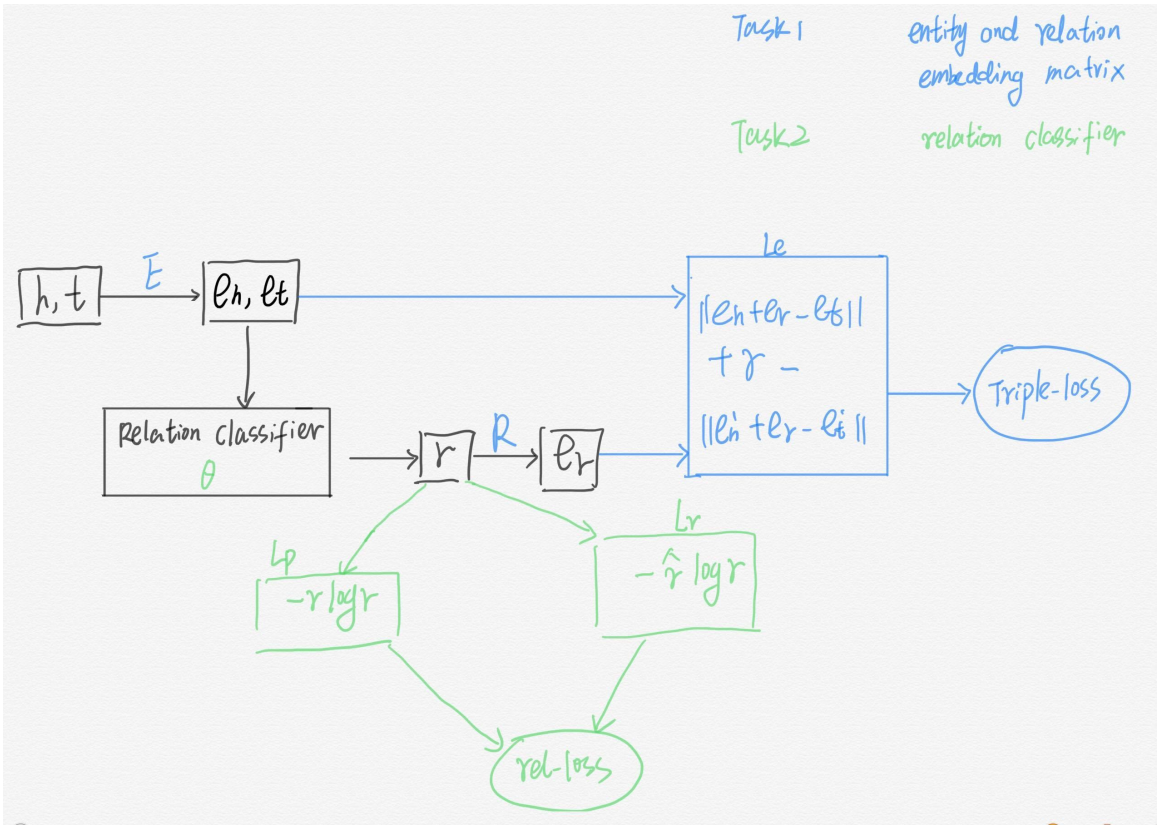


Figure 2.1: The data flow in the model.

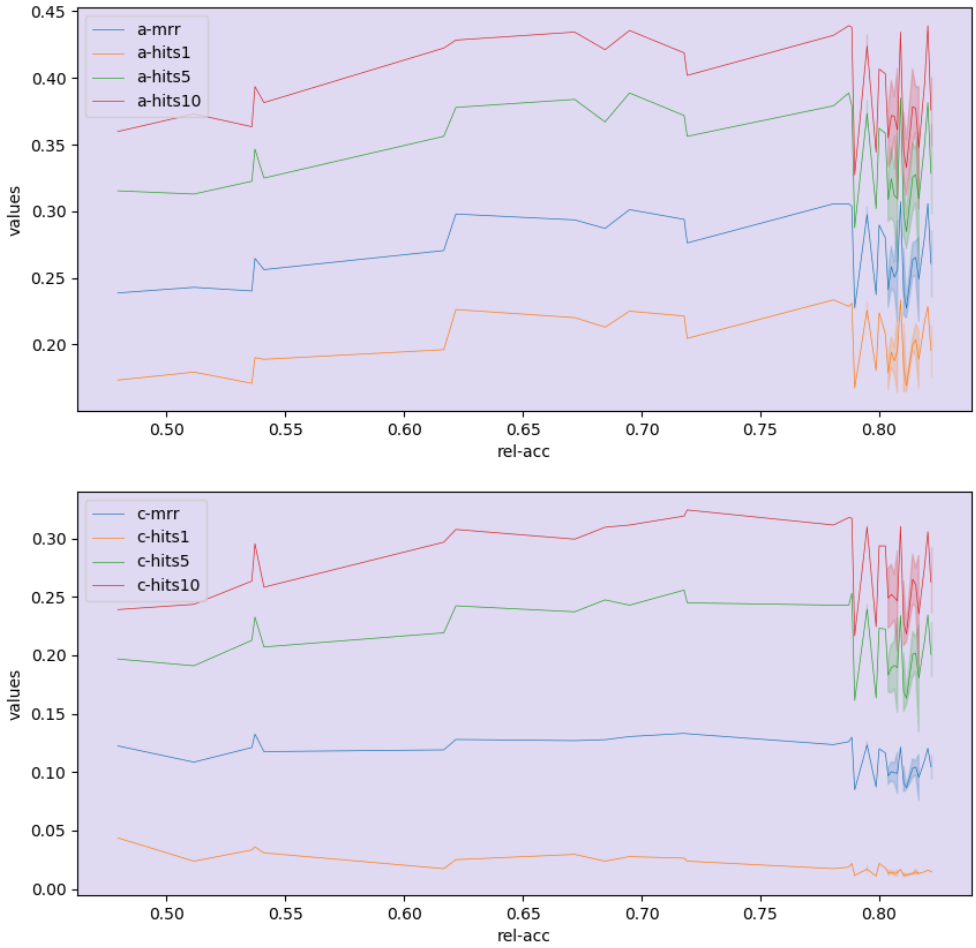


Figure 2.2: Correlations between rel-acc and other metrics. Embedding matrices and relations types are optimized with one optimizer. Parameters in relation classifier are updated with embedding matrices at every epoch.

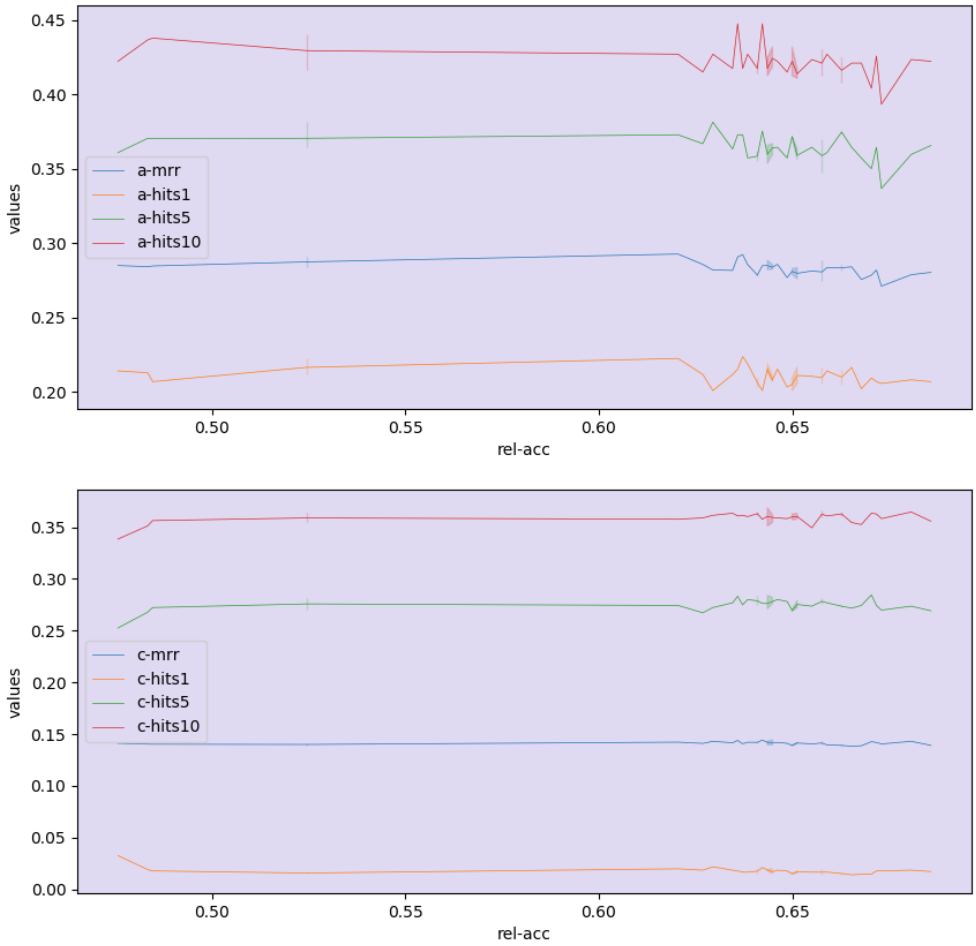


Figure 2.3: Correlations between rel-acc and other metrics. Embedding matrices and relations types are optimized with two optimizers. Parameters in relation classifier are updated at first 100 epoches and then updated every 30 epoches in the remaining 900 epoches. Embedding matrices are updated at every epoch. The vibration is reduced compared with Figure **??** and also the values of other metrics increased slightly. However, the average relaiton accuracy also dropped from 0.80 to 0.65 (underfitting). So in the next section, I tried to use less dropout rate and increase the learning rate and the hidden dimensions of relation classifier.

2.2 Train 34 relation types

Using similar startegy described as above. I run the model on the dataset with 34 types relations. Results are shown in Figure **??**. We can see the good news is the the overfitting problem is solved and the balance between rel-acc and other metrics is decent. When increasing the rel_hidden_dim from 100 to 500, the rel-acc also increases.

One problem I observed from Figure **??** is that the vibrations of these loss curves are very strong (see the backgrounds of three losses.) They are decreasing overall, but not very stable. This could suggest that the leraning rate is too large, I should try to lower it. However, the curves of metrics (Figure **??**) show lowering the learning rate also cause the dropping of performances. May be I could use a large learning rate at the beginning and a small one in the later phase.

iptranse_C_S_V6_0623

filename	beta1	beta2	learning_rate	rel_inp_dropout	rel_hid_dropout	rel_hidden_dim	tensorboard	a-hits1	a-hits5	a-hits10	a-hits50	a-mr	a-mrr	c-hits1	c-hits5	c-hits10	c-hits50	c-mr	c-mrr	rel-acc	Category 2
▼ Group 1																					
C_S_V6_271_5fold_1_20200622214118	2.0	1.0	0.01	0.2	0.1	500	tensorboar	0.1818	0.4065	0.5172	0.7325	79.6876	0.2915	0.0208	0.2367	0.3257	0.5117	170.3472	0.1253	0.4655	Group 1
C_S_V6_271_5fold_1_20200623041012	2.0	1.0	0.02	0.2	0.1	500	tensorboar	0.1808	0.4284	0.5486	0.8056	64.8495	0.2987	0.0215	0.2666	0.3492	0.5722	147.0787	0.1425	0.4681	Group 1
C_S_V6_271_5fold_1_20200623025141	2.0	1.0	0.03	0.2	0.1	500	tensorboar	0.2581	0.5319	0.6416	0.837	56.9091	0.3904	0.0202	0.2575	0.3453	0.5696	154.3231	0.1342	0.4538	Group 1
C_S_V6_271_5fold_1_20200622233723	2.0	1.0	0.04	0.2	0.1	500	tensorboar	0.2675	0.5852	0.6907	0.8548	51.604	0.4059	0.0176	0.2692	0.3628	0.6079	141.7399	0.1411	0.4603	Group 1
C_S_V6_271_5fold_1_20200622233709	2.0	1.0	0.06	0.2	0.1	500	tensorboar	0.279	0.5611	0.6594	0.8621	52.4044	0.41	0.0221	0.2731	0.3726	0.606	143.7971	0.144	0.4265	Group 1
C_S_V6_271_5fold_1_20200622214218	2.0	1.0	0.07	0.2	0.1	500	tensorboar	0.2727	0.5695	0.6771	0.8433	53.5287	0.405	0.0176	0.2718	0.3706	0.5897	146.6847	0.1416	0.4616	Group 1
▼ Group 2																					
C_S_V6_271_5fold_1_20200621234239	0.07	0.07	0.006	0.7	0.3	100	tensorboar	0.233	0.4765	0.582	0.7847	63.187	0.3504	0.0143	0.2542	0.3407	0.5787	143.6229	0.1297	0.2393	Group 2
C_S_V6_271_5fold_1_20200621234139	0.07	0.07	0.007	0.7	0.3	100	tensorboar	0.2445	0.4943	0.6123	0.7931	64.2341	0.3625	0.0247	0.249	0.3394	0.5774	143.8244	0.1348	0.3589	Group 2
C_S_V6_271_5fold_1_20200621234226	0.05	0.08	0.01	0.7	0.3	100	tensorboar	0.2644	0.5643	0.6688	0.8349	55.0637	0.3972	0.0228	0.275	0.3791	0.6086	134.2003	0.1444	0.329	Group 2
C_S_V6_271_5fold_1_20200621234026	0.05	0.05	0.01	0.7	0.3	100	tensorboar	0.2769	0.559	0.6698	0.8328	56.6071	0.4004	0.026	0.2692	0.3602	0.6151	136.2243	0.1435	0.3121	Group 2
C_S_V6_271_5fold_1_20200621202250	0.05	0.08	0.01	0.7	0.3	100	tensorboar	0.2027	0.4702	0.5789	0.7847	67.7586	0.3307	0.028	0.2471	0.3414	0.5735	146.8127	0.1386	0.1651	Group 2
C_S_V6_271_5fold_1_20200622213409	2.0	1.0	0.01	0.2	0.1	100	tensorboar	0.187	0.4222	0.5392	0.7409	78.8077	0.3008	0.0111	0.2191	0.3043	0.502	172.0299	0.1136	0.3732	Group 2

Figure 2.4: Results for asynchronously training on 34 relation types.

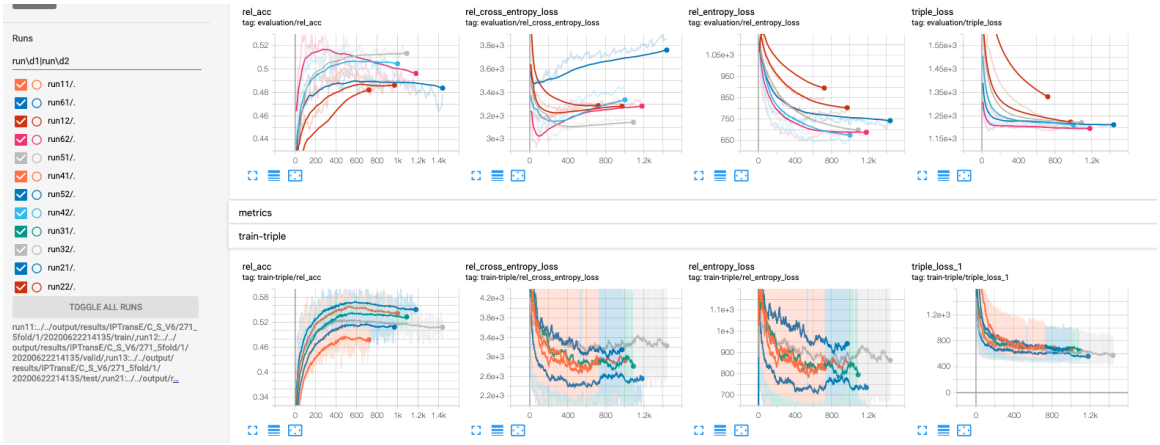


Figure 2.5: Learning curves for Group1 in Figure **??** The top part is the evaluation curves, and the bottom is the training curves (run1* to run6* indicate the learning rate increases from 0.01 to 0.07. (0.05 is still running)).

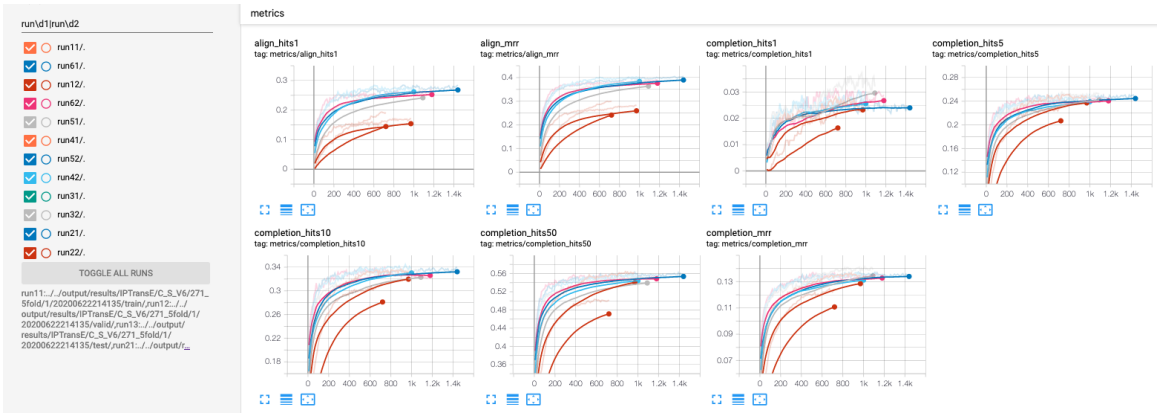


Figure 2.6: Curves of evaluation metrics for Group1 results in Figure **??**

2.3 Meeting Notes

To do list

1. Observe the model outputs on rank, completion, relaiton classification task.
   (a) Case study: observe head, relation, and tail predictions
   (b) Relation classification confidence measure
   (c) Alignment rank confidence measure
   (d) Completion rank confidence measure
   (e) Visualize the relation labelled SWOW graph.

Possible future directions: Expand to multilingual commonsense knowledge graphs alignment or completion. Becauae both ConceptNet and SWOW has multiple languages version.

Chapter 3

2020-04-28

To do list: Problems: Speed and Debug speed: backward process is very slow

There are two places need to tensors copies between devices: GPU → CPU → GPU.

The first one is when using FAISS to retrieve the neighbours, the ConceptNet and SWOW embedding matrix have to be copied from GPU to CPU. Because gpu tensors can not be input directly to the FAISS to build index, and search. After searching, these neighbour tensors have to be copied back to GPU.

The second is when constructing and training new graphs. Currently, I am using the output embeddings from GCN to retrieve neighbours. So I have to construct the graph dynamically in each epoch. This process increases two graphs with more nodes to train. For example, if the nodes number of ConceptNet and SWOW is N_C and N_S respectively. Then the increased two graph nodes number would be $N'_C \approx N_S + N_C$ and $N'_S \approx N_S + N_C$

	graph_batch_size	epoch_time (minutes)	forward_batch_time(s)	backward_batch_time(s)
Original	12,000 × 2	0.83	0.0177	0.0896
Now	12,000 × 2 + 15,000 × 2	7.8333	0.0203	0.8505

Table 3.1: Time consuming comparison

Which part slows down this process? Can you find a better method?

3.1 How to determine the parameters of IndexIVFFlat()

There are two parameters to the search metho nlist, the number of cells, and nprobe, the number of cells (out of nlist) that are visited to perform a search. The search time roughly increases linearly with the number of probes plus some constant due to the quantization.

1. How to choose a suitable nlist?
   If we have a matrix with n vectors, 4 * sqrt(n) is usually reasonable, or some other O(sqrt(n)).
2. How to choose a suitable nprobe?
3. Hot to decide the number of neighbours?

3.2 Code Details

```
import numpy as np
  def gpu_pytorch_neighbours(self, xb_t, xq_t):
          gpu_index_ivf = faiss.GpuIndexIVFFlat(self.res, self.d,\
                                                self.nlist, faiss.METRIC_L2)

          assert not gpu_index_ivf.is_trained
          xb_t = xb_t.detach().cpu().numpy()
          gpu_index_ivf.train(xb_t)        # faiss doesn't accept gpu tensors
```

```
        assert gpu_index_ivf.is_trained

        gpu_index_ivf.add(xb_t)

        D, I = search_index_pytorch(gpu_index_ivf, xq_t, self.k)

        return D, I
```

Listing 3.1: Python example

3.3 Model Outputs

I tried to compare the outputs of train one graph and jointly train two graphs.

Triple	ConceptNet Alone	rank	Jointly Train	rank
{"e1": "do housework", "e2": "clean house", "relation": "Causes"}	0.5661	1	0.6026	1
{"e1": "pen", "e2": "write", "relation": "UsedFor"}	0.5554	1	0.4404	1
{"e1": "apple", "e2": "green", "relation": "HasProperty"}	0.1712	1	0.5432	1
{"e1": "bottle", "e2": "plastic", "relation": "MadeOf"}	-	-	0.0185	9
{"e1": "bottle", "e2": "glass", "relation": "MadeOf"}	0.1614	1	0.0395	2

Table 3.2: Sampled top 10 predictions. - indicates the target is outside top10.

Sampled a weird instance and a positive instance.


```
 1 {"gold_triple": {"e1": "hammer", "e2": "drive nail", "relation": "UsedFor"},
780  "candidates": [{"e1": "hammer", "e2": "make music", "relation": "UsedFor", "score": "0.07095331"},
  1  {"e1": "hammer", "e2": "play music", "relation": "UsedFor", "score": "0.037341055"},
  2  {"e1": "hammer", "e2": "fun", "relation": "UsedFor", "score": "0.024378173"},
  3  {"e1": "hammer", "e2": "walk", "relation": "UsedFor", "score": "0.021624254"},
  4  {"e1": "hammer", "e2": "chop firewood", "relation": "UsedFor", "score": "0.02084147"},
  5  {"e1": "hammer", "e2": "tell time", "relation": "UsedFor", "score": "0.019344108"},
  6  {"e1": "hammer", "e2": "create music", "relation": "UsedFor", "score": "0.018772697"},
  7  {"e1": "hammer", "e2": "transport person", "relation": "UsedFor", "score": "0.018136388"},
  8  {"e1": "hammer", "e2": "measure time", "relation": "UsedFor", "score": "0.017449081"},
  9  {"e1": "hammer", "e2": "cut", "relation": "UsedFor", "score": "0.016184594"}]}
pk_candidates_conceptnet.jsonl
 17 {"gold_triple": {"e1": "hammer", "e2": "drive nail", "relation": "UsedFor"},
 16  "candidates": [{"e1": "hammer", "e2": "play music", "relation": "UsedFor", "score": "0.0399366"},
 15  {"e1": "hammer", "e2": "build thing", "relation": "UsedFor", "score": "0.03643348"},
 14  {"e1": "hammer", "e2": "fun", "relation": "UsedFor", "score": "0.034949657"},
 13  {"e1": "hammer", "e2": "make music", "relation": "UsedFor", "score": "0.03290259"},
 12  {"e1": "hammer", "e2": "storage", "relation": "UsedFor", "score": "0.024126869"},
 11  {"e1": "hammer", "e2": "cut", "relation": "UsedFor", "score": "0.019199098"},
 10  {"e1": "hammer", "e2": "decoration", "relation": "UsedFor", "score": "0.019029146"},
  9  {"e1": "hammer", "e2": "communication", "relation": "UsedFor", "score": "0.015224525"},
  8  {"e1": "hammer", "e2": "tool box", "relation": "UsedFor", "score": "0.014363699"},
  7  {"e1": "hammer", "e2": "build house", "relation": "UsedFor", "score": "0.013378359"}]}
```

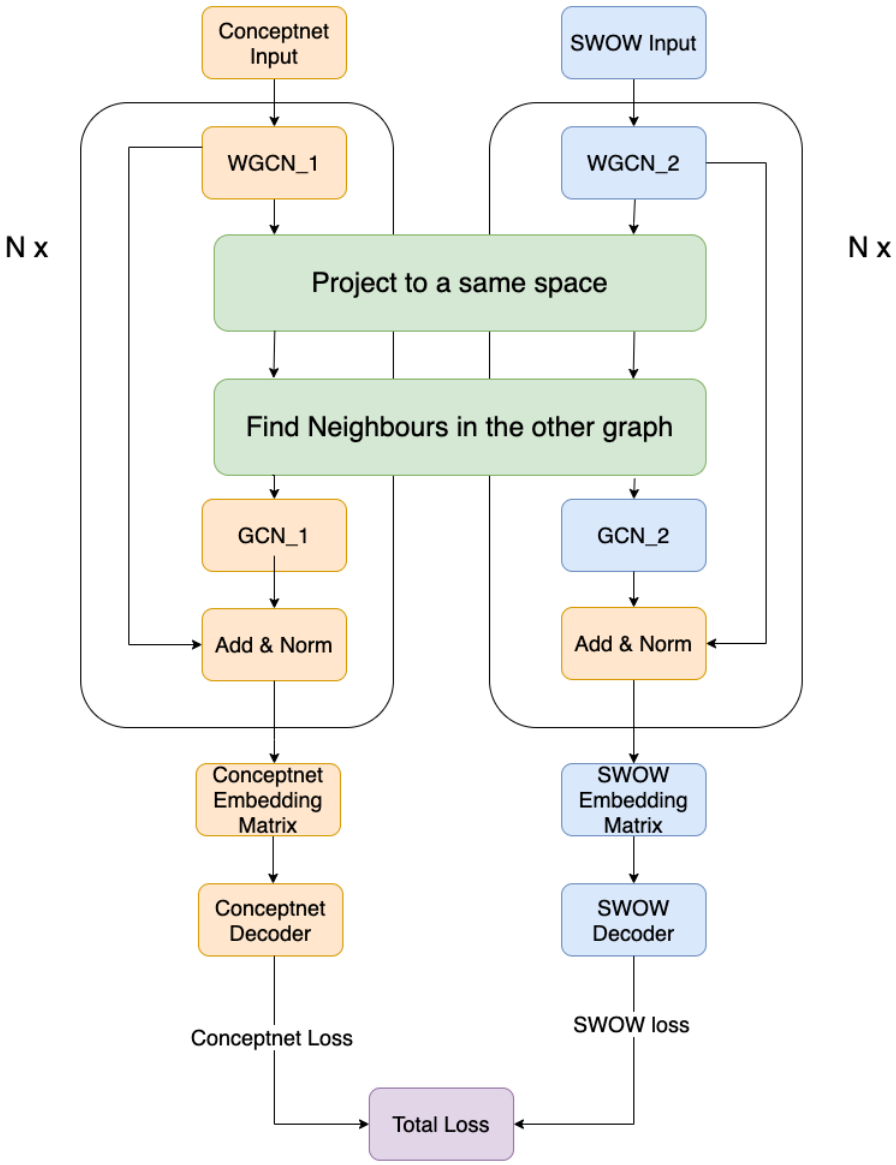

Figure 3.1: Hammer_top10_predictions. The top is from train ConceptNet alone and the bottom is from jointly training. Jointly train model is align before gcn embedding with pointwise pooling.

The prediction files can be found at here

Chapter 4

2020-04-21

4.1 Model Architecture



In previous methods, the neighbours of a node in a graph are static or fixed, and will not updated during the whole training process. The core view of this idea is to update the neighbours of a node in the training process. In this way, we can compute the neighbours of a the node and thus gain a another embedding for this node. Then, for each node, we can combine the two embeddings with different weight to gain a new embedding. For the sources of the neighbours, there can be two options, from the its own graph and another graph. If it is from its own graph, then it can be seen as the "self-attention" in Transformer. If it's from another graph, it can introduce more external knowledge. One thing needs to be noted is that we can only obtain the neighbour nodes, the edge label (relation type) information cannot be obtained. So the GCN for initial graph and updated graph will be a little bit different. Take the latter option for example, the process can be formulated with the following equations:

Notations:
C: the initial ConceptNet graph (with initialized node embedding matrix, relation types, each node has initial neighbours)
S: the initial SWOW graph. (similar as ConceptNet)
C_n: updated Concept graph with neighbours from SWOW (with nodes in original ConceptNet and also their embeddings. Each node has a set of neighbours, whose embeddings come from SWOW, no relation types)
S_n: updated SWOW graph with neighbours from ConceptNet. (similar to C_n)

$$\mathbf{E_c} = WGCN(C), \qquad \mathbf{E_s} = WGCN(S) \tag{4.1}$$
$$\mathbf{E_c^m} = f_m(\mathbf{E_c}), \qquad \mathbf{E_s^m} = f_m(\mathbf{E_s}) \tag{4.2}$$
$$C_n = g(\mathbf{E_c^m}, \mathbf{E_s^m}), \qquad S_n = g(\mathbf{E_s^m}, \mathbf{E_c^m}) \tag{4.3}$$
$$\mathbf{E_c'} = GCN(C_n), \qquad \mathbf{E_s'} = GCN(S_n) \tag{4.4}$$
$$E_c^o = f_l(\mathbf{E_c} + \mathbf{E_c'}), \qquad E_s^o = f_l(\mathbf{E_s} + \mathbf{E_s'}), \tag{4.5}$$

Equation **??** uses function f_m to project the two embedding matrices into a same space
Equation **??** finds the neighbours in in each other's graph.
Equation **??** passes the new graph (neighbours are from another graph) to another gcn.
Equation **??** adds the two embedding matrices from two gcn layer and performs a layernorm on them.

The above five equations will be wrapped as a whole and repeated many times.