2020 Spring GCN Final Report

Reproduction of paper *Graph Convolution Network for Road Network* [2]

Juhyeon Kim
Seoul National University
cjdeka3123@snu.ac.kr

June 2020

1 Summary of the original paper

In the final project, I investigated paper titled 'Graph Convolutional Networks for Road Networks' [2]. Before discuss the reproduced result, let's first briefly review the original paper.

[2] pointed out that traditional graph convolutional networks cannot reflect important features of road networks such as edge-relational property and volatile homophily. To overcome this shortcomings they proposed *Relational Fusion Network (RFN)* that both considers *node-relational* and *edge-relational* view.

First let's review how road network can be mathematically defined. Considering intersections as nodes and road segments as edges, road network can be represented as an attributed directed graph. One thing to note here is that attributes are defined not only on nodes/edges, but also between edges. They called this *between-edge*. Therefore, edge/between-edge set of primal graph becomes node/edge set of dual graph.

[2] proposed concept of relational fusion that can be applied on both primal and dual graph. Relational fusion is composed of fuse step and aggregate step (Fig. 1). At the fuse step, for each directed edge, node and edge representation is fused. They can be either just concatenated (AdditiveFuse) or fused with modeling interactions between representations (InteractionalFuse). At the aggregate step, fused features in previous step is aggregated over all neighbors. Attention can be used or not. To sum up, there can be total four variations of fuse/aggregate in relational fusion.

Relational fusion layer performs aforementioned relational fusion on both primal and dual graph. As described inf Fig. 2, relational fusion on primal graph takes vertex/edge representation as an input and produces next layer's vertex representation. Relational fusion on dual graph takes edge/between-edge(joined with vertex) representation as an input and produces next layer's edge representation. Next layer's between-edge representation is calculated by

simple feed forward network. Stacking several relational fusion layers, we can finally get relational fusion network.

[2] did experiments on driving speed estimation and speed limit classification task. For baseline algorithms, grouping estimator, MLP, GraphSAGE and GAT were used. For RFN models, four variations on fuse/aggregate methods were used. They found that RFN outperforms baseline modules. Especially a relational fusion network using the attentional aggregator with interactional fusion showed the best result.

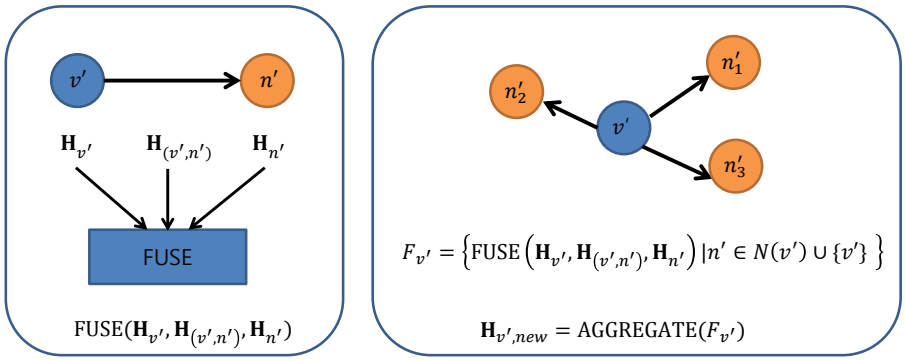


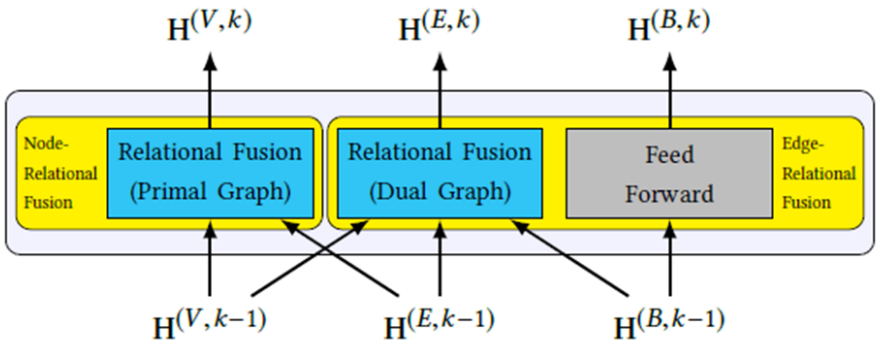
Figure 1: Relational fusion is composed of 'fuse' step (a) that fuses node/edge representations and 'aggregate' step (b) that aggregates neighbors' features.

2 Reproduction Results

A reference implementation of the Relational Fusion Network (RFN) is uploaded by the authors on github[1]. However the code contains only minimal part. It only shows an example on simple graph and does not shows how to generate real road network nor how to determined attributes on the road network. It also skips baseline modules. Therefore, although I referenced the original code, I had to implement a lot of missing parts. Followings are what I did. Code is uploaded on my github page[2].

- Road network generation on real cities (Korean cities).
- Determining node/edge/between-edge attributes.
- Implementation of baseline models (GraphSAGE, GAT).
- Miscellaneous things on establishing training/testing environment.

[1] `https://github.com/TobiasSkovgaardJepsen/relational-fusion-networks`
[2] `https://github.com/juhyeonkim95/GCN2020FinalProject`



(b) Relational Fusion Layer

$$
\begin{aligned}
\mathbf{H}^{(V,k)} &\leftarrow \textsc{RelationalFusion}^k(G^P, \mathbf{H}^{(V,k-1)}, \mathbf{H}^{(E,k-1)}) \\
\mathbf{H}^{(B',k-1)} &\leftarrow \textsc{Join}(\mathbf{H}^{(V,k-1)}, \mathbf{H}^{(B,k-1)}) \\
\mathbf{H}^{(E,k)} &\leftarrow \textsc{RelationalFusion}^k(G^D, \mathbf{H}^{(E,k-1)}, \mathbf{H}^{(B',k-1)}) \\
\mathbf{H}^{(B,k)} &\leftarrow \textsc{FeedForward}^k(\mathbf{H}^{(B,k-1)})
\end{aligned}
$$

Figure 2: Relational fusion is composed of 'fuse' step (a) that fuses node/edge representations and 'aggregate' step (b) that aggregates neighbors' fused features.

2.1 Dataset

2.1.1 Road Network Generation

As the original paper, I used OpenStreetMap data to construct road network. I used python library called *osmnx*[3]. I prepared road network of the top 12 Korean cities in order of population[4]. Unfortunately, Seoul was excluded because its road network was too large so caused memory error. Road networks of these cities can be found in Fig. 5.

2.1.2 Input Attributes Encoding

The original paper used zone-type as the node attributes, but I could not access that data. Therefore I just encoded each of the node attributes using a input/output degree. I thought this is somewhat reasonable because higher degree means more road or more urbanized. For edge attributes, fortunately, osmnx was providing information about the category and length of road. There were total 15 different categories ('primary', 'secondary', 'residential', etc). Road categories are one-hot encoded and lengths are encoded as continuous value.

[3] https://github.com/gboeing/osmnx

[4] `https://en.wikipedia.org/wiki/List_of_cities_in_South_Korea_by_population`

Finally for between-edge attributes, I used turn angles (-180 to 180 degrees) as the original paper. Using latitude and longitude values, we can calculate bearing (or orientation) of the road and by subtracting two roads' bearing, we can obtain turn angle. It was discretized in four classes, i.e., right-turn, left-turn, U-turn, or straight-ahead. Conventional GCNs can only use a single source of attributes, so as the paper did, I concatenated the source and target node features to the the edge attributes.

2.1.3 Target Data (Driving Speed Data)

The original paper did experiments on driving speed estimation and speed limit classification task. For driving speed estimation, they used collected historical data. Unfortunately, I have not been able to obtain road speed data over time or date. Instead, osmnx was providing function that adds default driving speed information to each road segment (`add_edge_speeds`), so I used this function. However, one problem was that the type of driving speed was very limited. For example, there were only 7 types of driving speed in Suwon (Table 1, visualized in Fig. 3). This is because `add_edge_speeds` imputes free-flow travel speeds for all edges based on mean 'maxspeed' value of edges. Thus, the speed limit classification task has been omitted because the average speed is already estimated through the speed limit. Also in the original paper, historical record is converted to a driving speed and concatenated to a road segment. However because historical records were not used in my experiment, I did not do such.

Table 1: Type of speed and corresponding road counts in Suwon.

Speed (kph)	Counts
31.0	14600
60.0	2702
51.9	2190
30.0	1405
50.4	321
90.0	28
40.0	8

2.2 Algorithms

I used same algorithms as the original paper. However, because grouping estimator was not clearly described, I excluded if from baseline models. Instead, I added GCN(or simplified ChebNet) as a baseline model. Followings are baseline models.

- MLP : A standard multi-layer perceptron. Neighbor's information is not used.

Figure 3: Road network of Suwon (my hometown). City is appended on the left top corner. The color means type of driving speed attribute. Color only distinguishes the type of speed, not the size of the speed. Note that there are only few types and it seems driving speed type strongly depends on the size of the road (boulevard, road or street).

- GCN : The graph convolutional network in [3].
- GraphSAGE : The GraphSAGE model used in [1].
- GAT : The graph attention network in [4].

Followings are RFN variants.

- RFN-Int-Att : RFN with interactional fuse, attentional aggregator.
- RFN-Int-Non : RFN with interactional fuse, non-attentional aggregator.
- RFN-Add-Att : RFN with additional fuse, attentional aggregator.
- RFN-Add-Non : RFN with additional fuse, non-attentional aggregator.

2.3 Experimental Setting

I used several different libraries. For GCN, GraphSAGE and GAT, I used *deep graph library*[5] with PyTorch backend. MLP was also implemented using PyTorch. For fairness, the depth and unit number of the hidden layer were set equal to all models (depth to 4 unit number to 16). For MLP, there is no graph structure, so I used mini-batch (256 sized) gradient descent.

In the original paper, train/test data set were divided by the date of historical data. However, as mentioned before, I couldn't access this date-by-date data. So I split the train/test data set using multiple cities. 8 cities were used for training (Busan Daejeon, Goyang, Gwangju, Incheon, Changwon, Seongnam, Cheongju), while 4 cities were used for testing (Daegu, Suwon, Ulsan, Yongin). I trained each model 100 times. Each time one city is picked from the list and the pick order was set to be same for all models. Adam optimizer was used with learning rate 1e-3. Mean square error loss was used.

2.4 Result

The error according to the training iteration is on 4 and the result on test data set is on Table. 2. As the paper said RFN gave better result than baseline models. All four variations gave better result, but especially RFN with interactional fusion and attentional aggregate gave the best result. Conventional graph neural networks failed to accurately predict driving speed. Unlike the paper, MLP gave better results than GCN. Even in the training process, it converged to a near zero error, which is lower than the RFN. I think this is because of two reasons: mini batch training was used and speed values are not actually measured, but are roughly classified by osmnx. Also, as the error was greater than RFN in the test process, it is considered that serious overfitting occurs when using MLP.

Table 2: Mean square loss of driving speed estimation for each algorithm/city.

Algorithm/city	Daegu	Suwon	Ulsan	Yongin	average
RFN_Int_Att	**0.01380**	**0.00647**	**0.01754**	**0.00821**	**0.01150**
RFN_Int_Non	0.01725	0.01252	0.01902	0.02812	0.01923
RFN_Add_Att	0.01598	0.01097	0.01968	0.02828	0.01873
RFN_Add_Non	0.01582	0.01013	0.01954	0.02983	0.01883
GAT[4]	0.04189	0.02764	0.03788	0.05231	0.03993
GCN[3]	0.04681	0.03600	0.04592	0.06588	0.04865
GraphSAGE[1]	0.04487	0.03236	0.03112	0.04475	0.03827
MLP	0.02547	0.01225	0.02236	0.01405	0.01853

[5] `https://www.dgl.ai/`

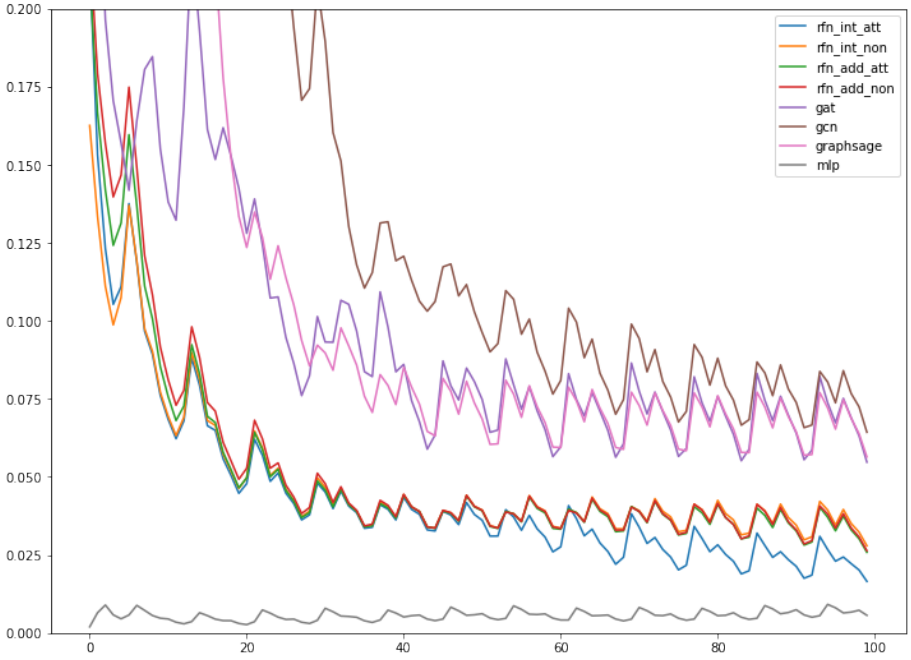


Figure 4: Training error per iteration. Y axis means MSE loss while x axis means iteration number.

2.5 Conclusion

To summarize, although limited data was used, it was possible to verify the effectiveness of RFN over traditional graph convolution networks as the paper says. If actual measurement data are used, it is thought that more interesting results can be obtained.

It was a great experience to be able to actually implement the various GCN models I learned in class. I have no doubt that it will be a great help in my future research.

References

[1] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems*. 2017, pp. 1024–1034.

[2] Tobias Skovgaard Jepsen, Christian S Jensen, and Thomas Dyhre Nielsen. "Graph convolutional networks for road networks". In: *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2019, pp. 460–463.

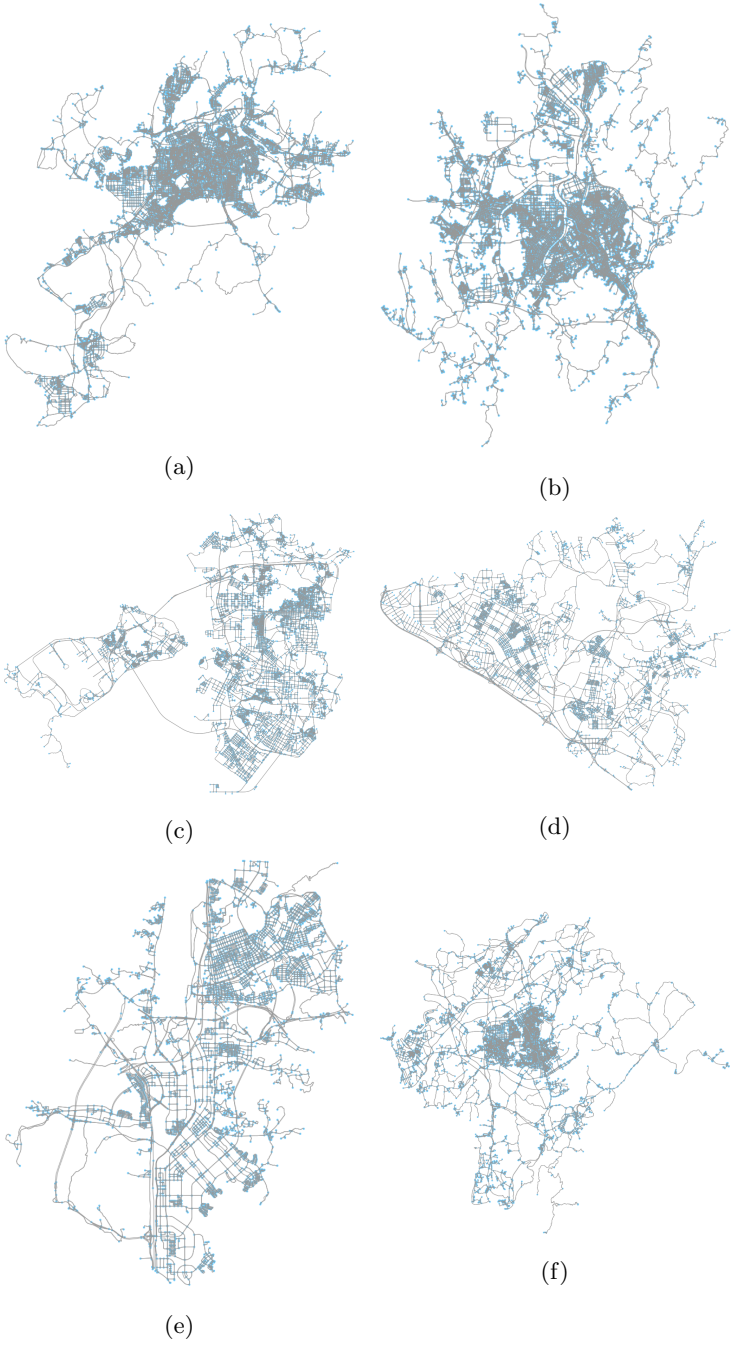


Figure 5: More examples of road networks of (a): Daegu (b): Daejeon (c): Incheon (d): Goyang (e): Seongnam (f): Cheongju

[3] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[4] Petar Veličković et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).