ADLG Project 2: GNNs on Knowledge and 3D Graphs

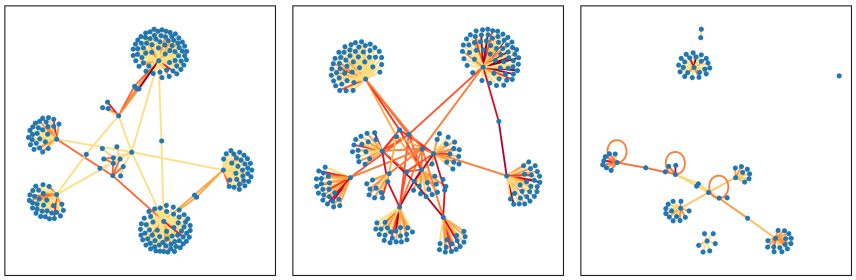
Ziheng Chi, Federico Felizzi, Jiaqing Xie
Student ID: 22-945-471, 09-901-240, 22-943-914
Department of Computer Science, ETH Zurich
{zihchi, ffelizzi, jiaxie}@student.ethz.ch

1 Knowledge graphs

1.1 Entity classification on AFIB dataset (Click Colab Link here)

Graph statistics. Like many other knowledge graphs, it lacks node features. It has **8285** nodes, **58086** edges, **4** entity classes, and **90** relation types. The whole graph is **undirected**, with diameter **8**.

Comparison between baselines and GNN. (1) Applied with a random classifier, the classification results are uniformly distributed. We run the test 100 times and achieve an average accuracy of 0.262 with standard deviation 0.070. **(2)** To build a feature classifier, we first augment the node features, i.e. adding one-hot features and interval features to each node. We choose interval features as example and embed it into a shallow MLP classifier. The accuracy is around 0.333 with standard deviation 0.118. **(3)** Our simple heterogeneous GNN classifier consists of two RGCN layers with hidden dimension 8. The model takes "None" as input instead of augmented features. We achieve an accuracy of 0.954 with standard deviation 0.026, which matches the SOTA RGCN performance.



(a) AFIB dataset (b) AM dataset (c) FB15k-237

Figure 1: Visualization of knowledge graphs (partly, triggered by 10 nodes and their neighbours).

Oversmoothing investigation. We check if the number of layers (depth) and the number of neurons (width) affect the accuracy of our model, where depth choices are 2, 5, 10, 20, 40, 80, and width choices are 4, 8, 16, 32, 64, 128. We find that the performance deteriorates with more layers. Adding neurons to a layer does not significantly decrease performance, and maintains an approximate on-par performance (Figure 3). We choose a model with 80 layers which suffers from oversmoothing according to the prediction results. We adopt **Dirichlet Energy** [4] as metrics, where a significant decrease in energy indicates oversmoothing. In order to overcome oversmoothing, we have tried three methods mentioned in the survey [4]: **PairNorm** [8], **Regularization**, and **Residual Connection**. We find that **PairNorm** not only improves the performance, but also overcomes the oversmoothing problem; residual connection overcomes oversmoothing but does not increase the performance significantly; regularization slightly mitigates oversmoothing.

1.2 Entity classification AM dataset (Click Colab Link here)

Graph statistics and baselines. Like AFIB dataset, it lacks node features. It has **1666764** nodes, **11976642** edges, **11** entity classes, and **266** relation types. The whole graph is **undirected**, with average node degree approximately **7.19**. The average accuracy of the random classifier is around 0.091 (approximately $\frac{1}{11}$), close

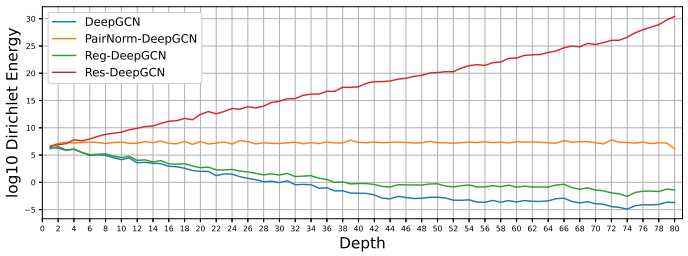


Figure 2: Depth vs Dirichlet energy with four methods.

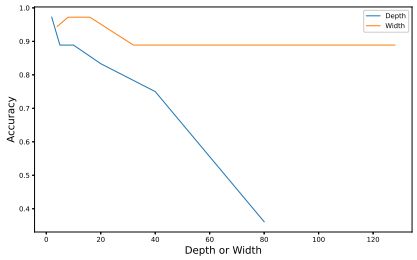


Figure 3: Depth/Width vs Accuracy.

to the uniform distribution. Then we augment features and construct an MLP model as a feature baseline, where seeds are set for reproduction. The average accuracy is around 0.253, with standard deviation 0.132.

Comparison of GNN models. Before applying GNN models, we first use `k_hop_subgraph` to sub-sample the graph so that it does not exceed CUDA memory. Then we choose two GNN models with a big performance difference in the AFIB dataset (width=8, depth=2 and width=8, depth=60) and apply them to the AM dataset. We run three experiments for each model with the same seeds and achieve average accuracies of 0.864 and 0.806 respectively. The reason why an expected big difference does not occur is that `k_hop_subgraph` has limited the graph depth and thus precluded the oversmoothing problem. To enhance the performance, we keep the depth unchanged while increase the width to 16, and achieve an accuracy of 0.874.

Model	Accuracy
Random Baseline	0.262 ± 0.070
Feature Baseline	0.333 ± 0.118
Shallow RGCN	0.954 ± 0.026
Deep RGCN + PariNorm	0.917
Deep RGCN + Regular.	0.417
Deep RGCN + Res.	0.417

Table 1: Results of AFIB dataset.

Model	Accuracy	Parameters
Random Baseline	0.092 ± 0.022	-
Feature Baseline	0.252 ± 0.132	-
Shallow RGCN	0.864	4,526,355
Deep RGCN	0.806	5,104,731
Enhanced RGCN	0.874	6,121,499

Table 2: Results of AM dataset.

1.3 Link prediction on FB15k-237 dataset (Click Colab Link here)

Data exploration and baselines. This dataset also lacks node features. It has **14541** nodes, **272115** training edges, **17535** validation edges, **20466** test edges, and **237** edge types. The whole graph is **directed**, with average node degree approximately **18.71**. A random baseline gives a MRR 0.00048 and H@10 0.0005. A feature baseline gives a a MRR 0.0042 and H@10 0.0059.

GNN architecture. We use encoder-decoder achitecture: RGCN encoder and TransE [1] decoder. The encoder consists two-layer RGCN with ReLU activation and dropout. We set the bias hyper-parameter in each GNN layer to False. Node embeddings are extract from the last layer of RGCN. For the decoder, we initialize relation embedding, extract head and tail embeddings and perform TransE. Other choices could be RGCN + RotatE or RGCN + DistMult, but here we only tried RGCN + TransE. MRR could achieve 0.25 and H@10 could achieve 0.4 under a normal setting. Then we perform a hyper-parameter search to find a state-of-the-art encoder-decoder architecture. During the hyper-parameter setting, we narrow the design space where we fix the number of layers to be 2, the dropout rate to be 0. We search the hidden dimension of GNN layers only to show an example of hyper-parameter searching since end-to-end optimization takes a long time. A larger hidden dimension led to better results.

Feature augmentations on graphs. We consider two types of augmentation methods on graphs: feature level and structure level. Feature level augmentation for knowledge graph is to initialize node embeddings since it lacks node features. One-hot features or constant features are taken into consideration. We fix the hyper-parameters which were explored by hyper-parameter search. RGCN with one-hot features reached MRR 0.2437 and H@10 0.3969. RGCN with constant features reached MRR 0.2372 and H@10 0.3659 . Feature augmentation on heterogeneous graphs such as FB15k-237 is useless. Instead we may consider structural augmentation on heterogeneous graphs: adding virtual node or dropping edges. Adding virtual node gave an on-par performance with MRR 0.2485 and H@10 0.4 approximately. We show that general augmentation methods for homogeneous graphs do not work well for heterogeneous graphs.

Model	MRR	H@10
Random Baseline	0.0005	0.0004
Feature Baseline	0.0042	0.0058
RGCN + TransE	**0.2534**	**0.4046**
RGCN + TransE + Constant	0.2372	0.3659
RGCN + TransE + One-Hot	0.2437	0.3969
RGCN + TransE + Virtual Node	0.2485	0.3999
RGCN + TransE Constrast + TransE	0.2007	0.2959
RGCN + DistMult Constrast. + TransE	0.1987	0.2789
RGCN + DistMult Constrast + DistMult	0.1538	0.2822

Table 3: Results of FB15k-237 dataset.

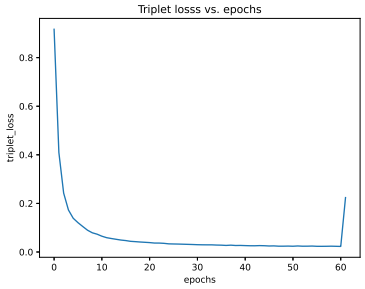


Table 4: Triplet loss vs epochs.

Contrastive learning. We use RGCN to extract the embedding for each node. As for anchor, we treat an existing triplet head-relation-tail as an anchor. The positive triplet has the same relation type as the anchor's relation, which also exists in training dataset. The negative triplet has the relation different than those with anchor and positive triplets. Then the embedding is then computed by DistMult or TransE for anchor, positive and negative. We use the Tripletloss to pretrain the embedding for entities. We find that pretraining with TransE has a lower MRR and H@10 than an end-to-end optimization pipeline. We also try to perform contrastive learning with DistMult distance and use the embedding with either DistMult or TransE decoder. We find that under our encoder-decoder settings, contrastive learning method does not work as expected. As mentioned in a recent research, a pure GNN-based method, pure translational distance model (transe) or pure semantic matching models (distmult) should be more suitable to contrastive learning in terms of consistency[2].

2 Graphs in 3D

2.1 Voxel format (Click Colab Link here)

Dataset statistics and baselines. In the ModelNet10 dataset, each data is a 3D shape, with an array $\mathbb{R}^{N\times 3}$ storing the coordinates of points, an array $\mathbb{R}^{3\times F}$ storing the point indices of faces, and a label describing its class. The dataset consists of a training set of 3991 shapes and a test set of 908 shapes, and contains 10 classes. We preprocess the dataset by normalizing the range of point coordinates to [-1, 1] for scale consistency and numerical stability. The random classifier returns a random class for each sample, where the accuracy is around 0.1, following the uniform distribution.

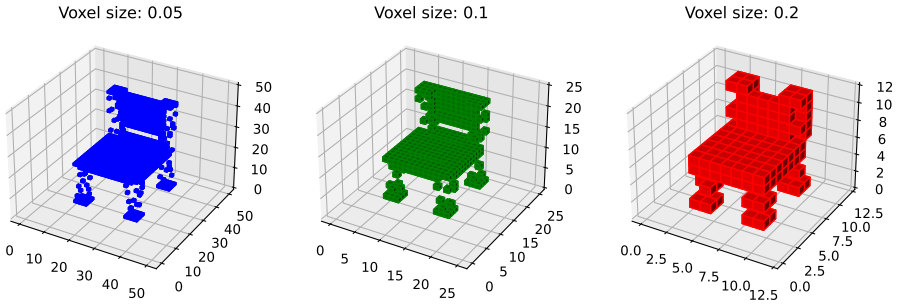


Figure 4: Chair example with different voxel sizes.

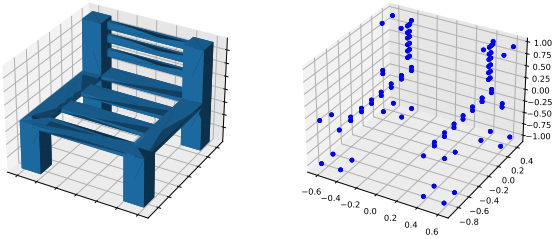


Figure 5: Chair example in mesh and point cloud.

Voxelization and CNN. We use the Open3D library to voxelize the dataset by creating a 3D array with elements 1 or 0 indicating whether the shape occupies the corresponding voxels. We compare three voxel sizes 0.05, 0.1, 0.2. First, we visualize the voxelization results in Figure 4. Then, we build a 3D CNN for the classification task. Drawing the idea from 3D ShapeNet [6], we use three `Conv3d` layers followed by two `Linear` layers, and we use ReLU as activation layers. We have achieved accuracies of around 0.896, 0.893, 0.889 respectively. The accuracy slightly decreases as the voxel size increases, since a smaller voxel size captures more details of the shape.

2.2 Point clouds (Click Colab Link here)

Coordinate baseline. The second way to represent a 3D shape is a point cloud, storing a list of point coordinates. After normalization, we use the `SamplePoints` function to sample 1024 points for each shape. First, we visualize the point cloud in Figure 5. Then we build a GNN coordinate baseline. Since a point cloud contains no edge information, we utilize the `knn_graph` function to generate edges based on the distance between points, where we choose 16 neighbors for each point. After obtaining edge indices, we build a two-layer GCN network with ReLU and `global_max_pool`. We achieve an accuracy of 0.868.

PointNet. We then test the performance of PointNet [3], using the implementation of pointNet.pytorch. We first apply a transformation to the input, then use three `Conv1d` layers without feature transformation, followed by a max pooling and three `Linear` layers. We choose three sampling sizes 512, 1024, 2048 and achieve accuracies of around 0.900, 0.893, 0.889 respectively. We find that the accuracy slightly decreases as the sampling size increases, which is probably due to overfitting, especially when the sampling size is larger than the original size. Comparison of 3D CNN and PointNet: (1) Information aggregation: CNN extracts neighboring information at each `Conv3d` layer, while PointNet only uses one max pooling layer; (2) Input adaptivity: one architecture of CNN only receives one voxel size, while PointNet has no limitations on sampling size. Our experiments show that these two methods achieve comparable results.

2.3 Meshes (Click Colab Link here)

Mesh GNN. The third way to represent a 3D shape is a mesh. Since point coordinates can be viewed as node features and faces can be transferred into edge indices, GNN can be applied. First, we only use the connection information as input, that is, to the edge indices while replace the coordinates by scalars. The accuracy is only 0.117, close to random guess. This happens because edges only provide relative information which is much weaker than the absolute information from the coordinates. Then we add coordinates back and achieve an accuracy of 0.807. Compared with original edges, we find that dynamically generated edges (`DynamicEdgeConv`) [5] can achieve a much better accuracy of 0.925.

Answers to report questions: 1. Similar to voxels and point clouds, mesh also allows resolution adjustment. We can insert or delete points at certain locations according to our need, and the corresponding edges and faces are also changed. 2. Since PointNet takes no connection information as input, dataset augmentations on structural modifications such as virtual nodes/edges will only reduce the performance of GNN.

Rotation-invariant GNN. In order to handle transformations in the dataset, we apply LGR-Net [7] and compare its performance with PointNet and fixed-edge GNN. We conduct three tests for each model: 1. original training set & original test set; 2. original training set & rotated test set; 3. rotated training set & rotated test set. We find that LGR-Net shows a robust performance, with negligible differences in rotated datasets, while the other two models are sensitive to rotations.

Model	Accuracy
3D CNN	0.896
PointNet	0.900
GNN: Edges	0.117
GNN: Coords + Edges	0.807
GNN: Coords + DynEdges (knn_graph)	0.868
GNN: Coords + DynEdges (EdgeConv)	**0.925**

Table 5: Results of different representations.

Model	ori→ori	ori→rot	rot→rot
LGR-Net	0.839	**0.837**	**0.839**
PointNet	0.900	0.135	0.591
GNN: Coords + Edges	0.807	0.131	0.318

Table 6: Results of rotation-invariant test. Here ori and rot refer to original and rotated dataset.

References

[1] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[2] Ke Liang, Yue Liu, Sihang Zhou, Wenxuan Tu, Yi Wen, Xihong Yang, Xiangjun Dong, and Xinwang Liu. Knowledge graph contrastive learning based on relation-symmetrical structure. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[3] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[4] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

[5] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds, 2019.

[6] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes, 2015.

[7] Chen Zhao, Jiaqi Yang, Xin Xiong, Angfan Zhu, Zhiguo Cao, and Xin Li. Rotation invariant point cloud classification: Where local geometry meets global topology, 2021.

[8] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.