Social Network Node Analysis and Community Mining

Xinghua Jia, Boyuan Yao, Ji Zhang

Abstract
We have completed some classic graph tasks related to the community. First, we use Louvain and Infomap algorithm for the community detection task. Secondly, we apply various analysis on graphs, including some basic models that used to simulate the real-world graph. Third, we use GCNII and knowledge distillation to outperform some typical models for node classification. Fourth, we study the combination of positional encoding (PE) and equivariant neural network for link prediction.

Keywords
Network analysis, Community detection, Node classification, Link prediction

Contents

1. Community Detection

Various social networks can often be described in terms of complex networks that have a topology of interconnected nodes combining organization and randomness. Community detection is a crucial technique to unearth the latent structure inside these systems. We will mainly discuss the well-known Louvain algorithm[1] and Infomap algorithm[2]. We also try method such as asynchronous label propagation[3], but it doesn't seem to perform as good as the above two algorithms, so due to the limited space, we omit this part and left the corresponding code in our project files.

1.1 Louvain

Louvain[1] is a representative community detection algorithm based on modularity optimization.

1.1.1 Modularity

The quality of community division is defined by the following modularity equation in Louvain

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta\left(c_i, c_j\right) \tag{1}$$

Where A_{ij} represents the weight of the edge between i and j, $k_i = \sum_j A_{ij}$ is the sum of the weights of the edges attached to vertex i, c_i is the community to which vertex i is assigned, the δ-function $\delta(u,v)$ is 1 if $u = v$ otherwise is 0.

With this modularity equation, it is easy for us to calculate the difference of modularity after we moving an isolated node i into a community C by the following equation

$$\begin{aligned}\Delta Q = &\left[\frac{\sum_{in}+k_{i,in}}{2m} - \left(\frac{\sum_{tot}+k_i}{2m}\right)^2\right] \\ &- \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]\end{aligned} \tag{2}$$

With above preperation, we have the following algorithm

1.1.2 Algorithm

The Louvain algorithm has the following two phases

Phase I:

1. Assign a different community to each node of the network
2. For each node i consider the neighbours j of i and evaluate the gain of modularity that would take place by removing i from its community and by placing it in the community of j. The node i is then placed in the community for which this gain is maximum(only if the gain is positive).
3. Repeat step two until no positive gain is possible.

Phase II:

1. Building a new network whose nodes are now the communities found during the first phase, the link between the new nodes are given by the sum of the weight of the links between nodes in the corresponding two communities, the link between nodes of the same community lead to self-loops for this community in the new network.
2. Apply Phase I to the new graph to iterate.

1.2 Infomap

Infomap[2] starts from randomwalk, during the randwomwalk process, we will repeatedly jump from node i to node j with probability $P(j|i)$, generating a long sequence of node. The intuition of Infomap is to take advantage of the duality of minimizing the coding of the randomwalk sequence and the community detection itself. By minimizing the coding, it could find a community partition of high quality.

1.2.1 Map equation & Two-layer coding

To reach the above purpose, Infomap introduces a two-layer coding for the graph. The first layer is to encode the groups, the code for each group is unique. The second layer is to encode the nodes inside that belongs to one group. Notice that as the group code distinguishes one group from the other, two nodes in different group could share the same node code.

Inspired by information encoding, a good group division could leads to a shorter set of codes in the above structure. Starting from Shannon's information entropy, the author gives the lower bound on code length with respect to a module partition M with the following map equation:

$$L(\mathrm{M}) = q_{\curvearrowright} H(\mathscr{Q}) + \sum_{i=1}^{m} p^{i}_{\circlearrowright} H\left(\mathscr{P}^{i}\right) \tag{3}$$

Here $H(\mathscr{Q})$ is the frequency-weighted average length of codewords in the index codebook and $H\left(\mathscr{P}^{i}\right)$ is frequency-weighted average length of codewords in module codebook i. With $q_{i\curvearrowright}$ indicates the probability to exit module i, the index codesbook is used at a rate $q_{\curvearrowright} = \sum_{i=1}^{m} q_{i\curvearrowright}$, the probability that the random walker switches modules on any given step. And with p_{α} for the probablity to visit node α, module codebook i is used at a rate $p^{i}_{\circlearrowright} = \sum_{\alpha \in i} p_{\alpha} + q_{i\curvearrowright}$. Therefore, the entropy for the index codebook is

$$H(\mathscr{Q}) = -\sum_{i=1}^{m} \frac{q_{i\curvearrowright}}{\sum_{j=1}^{m} q_{j\curvearrowright}} \log\left(\frac{q_{i\curvearrowright}}{\sum_{j=1}^{m} q_{j\curvearrowright}}\right) \tag{4}$$

and entropy for module codebook i is

$$\begin{aligned} H\left(\mathscr{P}^{i}\right) = &-\frac{q_{i\curvearrowright}}{q_{i\curvearrowright} + \sum_{\beta \in i} p_{\beta}} \log\left(\frac{q_{i\curvearrowright}}{q_{i\curvearrowright} + \sum_{\beta \in i} p_{\beta}}\right) \\ &-\sum_{\alpha \in i} \frac{p_{\alpha}}{q_{i\curvearrowright} + \sum_{\beta \in i} p_{\beta}} \log\left(\frac{p_{\alpha}}{q_{i\curvearrowright} + \sum_{\beta \in i} p_{\beta}}\right). \end{aligned} \tag{5}$$

With the above objective, Infomap algorithm has the following steps

1.2.2 Algorithm

The Infomap algorithm using the same approach as in PageRank to obtain the access probability of all nodes. After the probability generation, Infomap detects the communites with the following steps:

1. Assign each node to its own module.
2. In random sequential order, each node is moved to the neighboring module that results in the largest decrease of the map equation.
3. Repeat second step until there is no longer any move generates a decrease of the map equation.

1.3 Results

We apply above two algorithm to ego-Facebook dataset and attain the following results

(a) Louvain **(b)** Infomap

Figure 1. Community detection

	Louvain	Infomap
number of communities	16	79
modularity	0.835	0.811

(6)

We could see that Infomap provides more communities, while the Louvain works better on the metric modularity. As the Louvain aims to optimize the modularity, it is predictable that Louvain could outperform Infomap on this metric. But Infomap is also an excellent method which might be able to discover a more refined community structure.

2. Network Analysis

2.1 Centrality Measurements

We select ego-facebook dataset for mentioned on last section for this experiment, using 6 kinds of centrality measurements: degree centrality, eigenvector centrality, Katz centrality, pagerank centrality, betweenness centrality and closeness centrality. We use the threshold that highlights nodes that have at least 50% of biggest centrality with red and nodes that have at least 25% of biggest centrality with cyan. The followings are the results

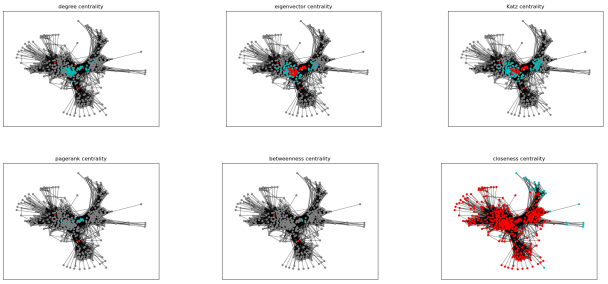

Figure 2. Centrality Measurements

We could see that under different measurements, the number of nodes with high centrality(i.e. the threshold mentioned above) varies in a big range.

2.2 Network Attributes

We also try to measure network attributes of ego-Facebook dataset and various methods that are used to generate graphs with same number of nodes as ego-Facebook and similar number of edges, including random graph generate through Erdos-Renyi model(ER), Watts-Strogatz model of small world(WS) and preferential attachment graph(BA model). We measure the average clustering coefficient, average distance and diameter of the above graphs. Moreover, we visualize the distribution of degree of the above graphs. Table1, Table2 and Figure3 show the results.

From the results, we could see that Erdos-Renyi random graph could only fit the average distance of the real-world graph, the small world model could excellently fit the average clustering coefficient and the diameter, but lose efficiency while fitting the degree distribution. The BA model is able to generate graph with degree distribution that fit the power law distribution, but has much smaller average clustering coefficient than the real-world graph.

Table 1. Graph Basic Features

Graph	#-Nodes	#-Edges
Facebook	4039	88324
ER	4039	89551
WS	4039	88858
BA	4039	88374

Table 2. Network Attributes

Edges	Clustering	Distance	diameter
Facebook	0.606	3.69	8
ER	0.011	2.60	4
WS	0.711	4.11	6
BA	0.037	2.51	4

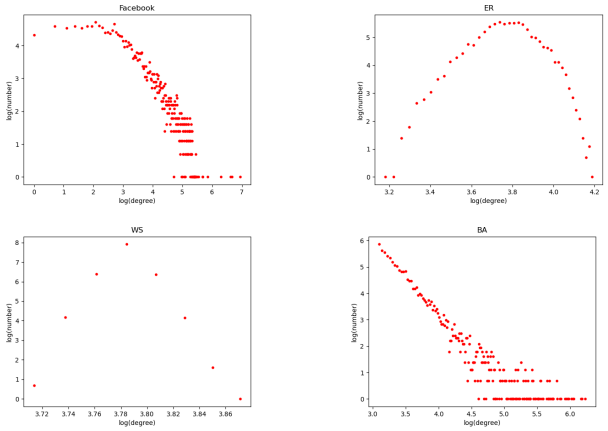

Figure 3. Degree Distribution

2.3 Network Evolution

In this part we try to use different link probability in Erdos-Renyi model to simulate the network evolution. Starting from a graph with 1000 nodes and 0 edge, we show some intermediate time steps below

Table 3. Attributes Over Time

Average Degree	Clustering	Distance	Diameter	Size
0.014	0.0	1.0	1	2
0.092	0.0	1.33	2	3
1.068	0.0	11.80	27	149
9.878	0.01	3.27	6	1000

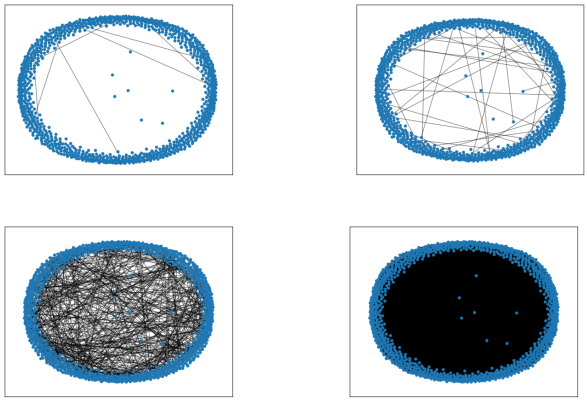

Figure 4. Evolution Visualization

The size attribute means number of nodes of the biggest subgraph in the current time step. We could see that while average degree is about 1, the graph has the biggest average distance and diameter.

3. Node classification

Node classification aims to assign labels to nodes with unknown labels in the graph with the help of connectivity relations between nodes and few nodes with known labels. In this project, we focus on the combined use of graph neural networks and knowledge distillation. Specifically, we do the following studies:

- Reproduce and study the performance of some typical graph neural networks in this task.
- Analyze the bottlenecks that limit the performance of these typical models.
- Study the use of deep graph neural networks to break the upper performance limit.
- Stduy the use of knowledge distillation to extract knowledge into simpler models.

3.1 Baseline: Node2Vec

We use the model performance of Node2Vec [4] as the baseline for the node classification task. Node2Vec is a modification of DeepWalk for node embedding that adjusts the probability of random wandering so that the result of embedding is a trade-off between 'homogeneity' and 'structure'. After obtaining the vector embedding of the nodes, machine learning methods can be used to predict the node labels.

In our experiments, node2vec can achieve 73.8% test accuracy on the cora dataset. The rigid combination of the two processes makes the performance far from satisfactory, so we choose it as a baseline for subsequent comparison.

3.2 Typical graph neural networks

Graph neural networks (GNNs) are the most popular models for performing node classification tasks. Based on Pytorch-Geometric, an easy-to-use GNNs library built upon PyTorch, we implement GCN [5],GAT [6] and compare their performance on the Cora dataset.

3.2.1 Graph Convolution Network

Graph Convolution Network (GCN) is a convolutional neural network that operates directly on graphs and is able to perform semi-supervised learning on graph structured data. Motivated by a localized first-order approximation of spectral graph convolution, GCN follows the following rules for propagation between layers:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{7}$$

Where $\tilde{A} = A + I_N$ denotes the adjacency matrix of the undirected graph with added self-connections; $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ denotes the degree matrix of the nodes; $W^{(l)}$ denotes the layer-specific trainable weight matrix; $\mathbf{H}^{(\ell)}$ denotes the output of the l^{th} layer; $\sigma(\cdot)$ denotes an activation function like ReLU.

3.2.2 Graph attention network

Graph attention network (GAT) proposed a masked self-attentive layer to address the shortcomings of previous methods based on graph convolution or its approximations. The features of node j in the graph are propagated forward in the network according to the following equation:

$$\vec{h}_i' = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\vec{h}_j\right) \tag{8}$$

Where $\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})}$ and e_{ij} denotes LeakyReLU $\left(\vec{\mathbf{a}}'\left[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j\right]\right)$; $\vec{\mathbf{a}}$ denotes the role of a single-layer feedforward neural network; and W denotes the trainable weight matrix, same meaning as $W^{(l)}$ in GCN.

3.2.3 Comparison

In this section, we compare the test accuracy of Node2Vec, GCN and GAT on the cora dataset for longitudinally. To be fair,we train each model for 600 epochs on Cora dataset, using SGD optimizer with learning rate=0.01 and weight decay=5e-4. The results are shown in the table 4.

Table 4. Performance of different models on Cora

Model	Test Acc(%)
Node2Vec	73.8
GCN	**82.3**
GAT	81.8

Observe that GCN (82.3%) and GAT (81.8%) get much better results than the baseline Node2Vec(73.8%). This outperformance is due to the fact that GNNs learn the graph as a whole, i.e. transductive learning. In addition, GCN outperforms GAT, but in our experiments we found that GCN peaked in accuracy at 527 rounds while GAT peaked at 69 rounds, so the internal performance of GNNs needs further investigation.

3.3 Bottlenecks in GNNs

Based on the phenomena in the previous section, we experiment further with more GNNs such as ChebNet [7] with the same hyperparameter settings, and tried some additional techniques such as correct & smooth [8].

3.3.1 Overfit

It was found that almost all GNNs methods tested on the Cora dataset intervened between 80% and 83.5% accuracy, with or without the use of additional techniques. At the same time, their accuracy on the training set is very close to 100%, which means that GNNs generally suffer from overfitting problems.

3.3.2 Over-smoothing

On the other hand, the GNNs in the previous experiments were generally 2-layer deep networks, and stacking more layers instead resulted in a significant drop in accuracy. This phenomenon has been noted as over-smoothing [9], where neighbouring nodes become increasingly similar as the number of layers of forward propagation increases, making the model less discriminating.

3.4 Model imporvment

To break through the above bottleneck, we referenced related work to replicate GCNII [9] and extracted its knowledge to a simple MLP model using knowledge distillation [10].

3.4.1 GCNII

GCNII is an improved approach to GCN that solves the over-smoothing problem by retaining certain initial features as well as weight information during feature propagation, specifically, the feature forward propagation equation of GCNII is as follows:

$$\mathbf{H}^{(\ell+1)} = \sigma\left(\left((1-\alpha_\ell)\tilde{\mathbf{P}}\mathbf{H}^{(\ell)} + \alpha_\ell\mathbf{H}^{(0)}\right) * \left((1-\beta_\ell)\mathbf{I}_n + \beta_\ell\mathbf{W}^{(\ell)}\right)\right) \tag{9}$$

Where $\tilde{\mathbf{P}}$ denotes $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ and $\tilde{D}_{ii}, \tilde{A}, \mathbf{H}^{(\ell)}, \sigma(\cdot)$ have been defined in formula 7; α_ℓ and β_ℓ are artificially settable weighting parameters.

By using GCNII, the model can achieve better performance as the number of layers increases. Experimentally, we found that using 64 layers of GCNII was able to achieve the highest test accuracy (85.5%) on the Cora dataset, in line with the experimental results in the original paper.

GCNII also mitigates overfitting by solving the problem of over-smoothing. Compared to typical GNNs, GCNII's test accuracy is at least 2% better, although overfitting is still present, i.e. it is very close to 100% accuracy on the training set.

3.4.2 Knowledge distillation

On the other hand, the increase in the number of network layers inevitably leads to an increase in computational cost, and in experiments, GCNII was at least five times slower from training to classification than typical GNNs, which again is far from satisfactory.

Therefore, we refer to the idea of knowledge distillation in related work and try to train fewer rounds on GCNII and extract its knowledge into a simple MLP model, and because MLP requires less information in terms of graph structure than GNNs, the authors of the original paper named this approach Graph-less Nerual Network (GLNN). Specifically, the framework for knowledge distillation is as follows.

- step I: Training teacher model. The teacher model can be any well-performing model such as GCN, GAT etc. In our project we use GCNII as the teacher model.

- step II: Training student model by teaching. Student models should theoretically be as simple as possible, so in our project we have used an MLP with 2 layers as the student model. Unlike training the MLP directly, the loss function needs to be modified in the knowledge distillation to the following equation:

$$\begin{aligned}\mathscr{L} =& \lambda \sum_{v\in\mathscr{V}_L} \mathscr{L}_{\text{label}}\left(\hat{y}_v, y_v\right) \\ &+(1-\lambda)\sum_{v\in\mathscr{V}} \mathscr{L}_{\text{teacher}}\left(\hat{y}_v, z_v\right)\end{aligned} \tag{10}$$

Where $\mathscr{L}_{\text{label}}$ denotes the training loss between MLP predictions and ground truth labels and $\mathscr{L}_{\text{teacher}}$ denotes the training loss between MLP predictions and teaxher predictions; λ is artificially settable weighting parameter.

In our experiments, we still use the Cora dataset and train the teacher model and the student model for 200 epochs each, i.e. about one third of the training cost of GCNII, and achieve a performance comparable to the highest accuracy of the teacher model (85.5%) on the test set.

3.4.3 Result analysis

To summarise, the test accuracy of the model on the Cora dataset before and after optimisation is shown in Table 5:

Table 5. Performance before and after optimisation on Cora

Model	Test Acc(%)
GCN	82.3
GCNII	**85.5**
MLP	59.9
GLNN(GCN)	82.2
GLNN(GCNII)	**85.4**

GLNN(GCN | GCNII): GLNN using GCN | GCNII as teacher model

It was observed that considerable accuracy gains were achieved using GCNII compared to typical GCNs; and using knowledge distillation, the student model achieved an optimal accuracy comparable to the teacher model, both with GCN as the teacher model and with GCNII as the teacher model, while the overall computational cost of the framework was only about one-third of that originally required to train the teacher model.

In addition, GLNNs benefit from the inductive nature of MLP in practice. When a new node is added to the original graph, GLNNs can directly classify it, whereas for traditional GNNs they can only be retrained on the new graph in order to classify it.

3.5 Visualization

In order to be able to visualize the effect of the classification of the model nodes, we visualize and downscale the Cora nodes using the t-SNE algorithm, shown as Graph 5.

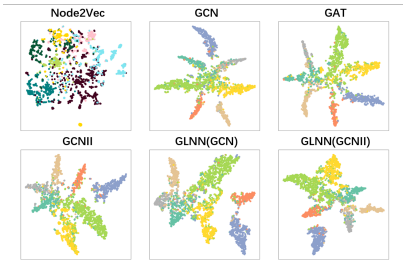

Figure 5. Node Classification Visualization

It was observed that the latter five models were close in visualisation; the Node2Vec visualisation was more cluttered, which can be reflected in its lower classification accuracy.

4. Link prediction

Link prediction is a fundamental problem in social network analysis. We mainly study the combination of positional encoding (PE) and equivariant neural network in this project. To be specific, we do the following things in this project:

- Study why PE is important in link prediction.
- Study how to solve the problem of unstability of PE with equivariant GNN.
- Design a new architecture that delicately combine GAT[6] with PE.
- Design a new equivariant RGCN[11] with PE on heterogeneous graph.

4.1 Introduction of PE in link prediction.

Graph neural networks have recently become a mainstream method to tackle a lot of prediction tasks in graph and outperform many traditional methods. However, for link prediction, GNN models like GCN[5], VGAE[12] cannot beat traditional methods in several public datasets like Collab and PPA in OGB[13]. The failure is partly due to their inability to differentiate nodes with symmetric local struture. For example, if we run a GNN encoder on a 4-node circle structure, we will obtain the same feature for each node, which makes it impossible to reconstruct the original structure. In a large social network, we may have hundreds of nodes with close feature.

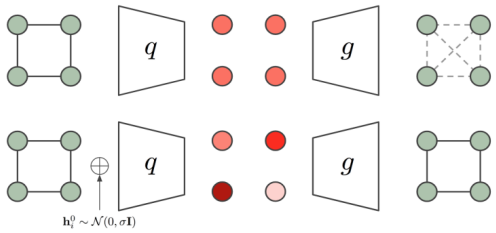

Figure 6. 4 node circle from EGNN

So we should introduce an extra feature to differentiate these nodes.

Definition 4.1 *(Positional Encoding). Given a graph $G = (A,X)$, PE works on A and gives $Z = \mathrm{PE}(A) \in \mathbb{R}^{N\times p}$ where each row Z_v gives the positional feature of node v.*

As in the above figure, some works introduce random features or deterministic distance encoding which is either too noisy or high-cost. Several works have introduced Laplacian eigenmap(LE) as PE[14] which is more reliable.

Definition 4.2 *(Laplacian matrix). Graph can be denoted as $\mathscr{G} = (A,X)$, where A is the adjacency matrix. $X \in \mathbb{R}^{N\times F}$ denotes the node features, where the v th row, X_v, is the feature vector of node v. Denote D as the diagonal degree matrix where $D_{vv} = \sum_{u\in[N]} A_{vu}$ for $v \in [N]$. Let $d_{\max} = \max_{v\in[N]} D_{vv}$. Denote the normalized adjacency matrix as $\hat{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ and the normalized Laplacian matrix as $L = I - \hat{A}$.*

Definition 4.3 *(Laplacian eigenmap). Let Z_{LE} denote LE, which includes the eigenvectors that correspond to the p smallest eigenvalues of the normalized Laplacian matrix L.*

However, the above works didn't consider the situation that the matrix may have multiple same eigenvalues or several very close eigenvalues which will lead to a $SO(n)$ disturbance to eigenvectors. In such situation, the value of PE will be unstable, which is dangerous in real-world application! We will introduce a new paper PEG[15] in the next part which succeeded to solve this problem.

4.2 Equivariant and stable PE for more powerful graph neural networks(PEG)

4.2.1 The problem of LE

Eigenvalue decomposition of a positive semidefinite(PSD) matrix is not unique. To be more specific, see the following Lemma.

Lemma 4.1 *EVD is not unique. If all the eigenvalues are distinct, i.e., $\lambda_i \neq \lambda_j, U$ is unique up to the signs of its columns, i.e., replacing u_i by $-u_i$ also gives EVD. If there are multiple eigenvalues, say $(\lambda_{i-1} <)\, \lambda_i = \lambda_{i+1} = \ldots = \lambda_{i+k-1}\, (< \lambda_{i+k})$, then $[u_i, u_{i+1}, \ldots, u_{i+k-1}]$ lie in an orbit induced by the orthogonal group $\mathrm{SO}(k)$, i.e., replacing $[u_i, u_{i+1}, \ldots, u_{i+k-1}]$ by $[u_i, u_{i+1}, \ldots, u_{i+k-1}]\, Q$ for any $Q \in \mathrm{SO}(k)$ while keeping eigenvalues and other eigenvectors unchanged also gives EVD.*

After this Lemma, we can naturally introduce the definition of PE-matching and PE-distance. PE-distance measures distance between two eigenvector space or two PE.

Definition 4.4 *(PE-matching). Consider two groups of positional features $Z^{(1)}, Z^{(2)} \in \mathbb{R}^{N\times p}$. Their matching is given by $Q^*\left(Z^{(1)}, Z^{(2)}\right) \triangleq \arg\min_{Q\in\mathrm{SO}(p)} \left\|Z^{(1)} - Z^{(2)}Q\right\|_{\mathrm{F}}$. Later, Q^* is used if it causes no confusion. Define the distance between them as $\eta\left(Z^{(1)}, Z^{(2)}\right) = \left\|Z^{(1)} - Z^{(2)}Q^*\right\|_{\mathrm{F}}$.*

The specific definition of stability is defined as follows. This PE-stability is the ultimate goal of PEG which is ignored by former works.

Definition 4.5 *(PE-stability & PE-equivariance). Consider a GNN layer g that uses PE. When it works on any two graphs $\mathscr{G}^{(i)} = \left(A^{(i)}, X^{(i)}\right), i \in \{1,2\}$ and gives $\left(\hat{X}^{(i)}, \hat{Z}^{(i)}\right) = g\left(A^{(i)}, X^{(i)}, Z^{(i)}\right)$, let P^* be the matching between the two graphs. g is PE-stable, if for some constant $C > 0$ we have*

$$\left\|\hat{X}^{(1)} - P^*\hat{X}^{(2)}\right\|_{\mathrm{F}} + \eta\left(\hat{Z}^{(1)}, P^*\hat{Z}^{(2)}\right) \leq Cd\left(\mathscr{G}^{(1)}, \mathscr{G}^{(2)}\right).$$

A weaker condition of PE-stability is PE-equivariance: If $A^{(1)} = PA^{(2)}P^T$ and $X^{(1)} = PX^{(2)}$ for some $P \in \Pi$, we expect a perfect match between the updated node features and positional features, $\hat{X}^{(1)} = P\hat{X}^{(2)}$ and $\eta\left(\hat{Z}^{(1)}, P\hat{Z}^{(2)}\right) = 0$.

Lemma 3.4 in PEG implies that small perturbation of graph structures may yield a big change of eigenvectors and thus a big change of particular column of $\mathrm{LE}(A)$ if there is a small eigengap. However, Lemma 3.5 in PEG implies that

the eigenspace , i.e., the space spanned by the columns of LE(A) could be much more stable. This motivates that if we want to achieve PE-stability, we should make the GNN layer invariant to the selection of bases of the eigenspace for the positional features. They mentioned two conditions PE-stable GNN layer g should satisfy:

1) Permutation equivariance w.r.t. all features;

$$(P\hat{X}, P\hat{Z}) = g\left(PAP^T, PX, PZ\right), \forall P \in \Pi(N), \tag{11}$$

2) Rotation equivariance w.r.t. positional features, i.e.,

$$(\hat{X}, \hat{Z}Q) = g(A, X, ZQ), \forall Q \in \mathrm{SO}(p). \tag{12}$$

4.2.2 PEG architecture

Condition (9),(10) can be satisfied by EGNN[16]. EGNN considers feature and physical coordinate of nodes and design a GNN layer that satisfies $SO(n)equivariance$ for physical coordinates. In PEG, they use a simplified EGNN and view positional encoding features as pysical coordinate: Use a GCN layer with edge weights according to the distance between the end nodes of the edge and keep the positional features unchanged. This gives the PEG layer,

$$\text{PEG}: \quad g_{\text{PEG}}(A, X, Z) = (\psi[(\hat{A} \odot \Xi)XW], Z),$$

where

$$\Xi_{uv} = \phi\left(\|Z_u - Z_v\|\right), \forall u, v \in [N].$$

Here ψ is an element-wise activation function, ϕ is an MLP mapping from $\mathbb{R} \to \mathbb{R}$ and $\odot$ is the Hadamard product. Note that if $\hat{A}$ is sparse, only Ξ_{uv} for an edge uv needs to be computed.

4.3 Equivariant GAT with PE.

On understanding the principle of PEG, we find that PEG and GAT both essentially utilize distance information source to re-weight the neighbor features. Recall the weight in GAT:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\overrightarrow{\mathbf{a}}^T\left[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\overrightarrow{\mathbf{a}}^T\left[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k\right]\right)\right)} \tag{13}$$

In our experiment, we use a multi-head version of GAT:

$$h_i'(K) = \|_{k=1}^{K}\sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k W^k h_j\right) \tag{14}$$

We devised several kinds of architecture to natually combine GAT and PEG, we describe two important architecture in the report. The idea of GAT-PE-v1 is nearly the same as PEG. It simply multiplies p_{ij} on α_{ij} to update the weight.

Version 1:

$$p_{ij}^h = MLP^h(\left\|Z_i - Z_j\right\|) \tag{15}$$

$$\alpha_{ij}^h = \frac{\exp\left(\text{LeakyReLU}\left(\overrightarrow{\mathbf{a}}_h^T\left[\mathbf{W}_h\vec{h}_i \| \mathbf{W}_h\vec{h}_j\right]\right)\right)p_{ij}^h}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\overrightarrow{\mathbf{a}}_h^T\left[\mathbf{W}_h\vec{h}_i \| \mathbf{W}_h\vec{h}_k\right]\right)\right)} \tag{16}$$

where h denotes feature and parameter of the hth head. However, this simple kind of combination is too naive to leverage the power of GAT and PEG. The simple multiplication will destroy the original structure of GAT.

To average weight coming from GAT and PE more delicately and interact feature weight from different head channel, we devise the following architecture:

Version2: First compute p_{ij}^h and α_{ij}^h as (11) and (13), then we use a MLP layer to encourage weight average and interaction between GAT and PE.

$$\alpha_{ij} = \mathbf{W}concat(\alpha_{ij}, p_{ij}). \tag{17}$$

here $\mathbf{W}$ is linear mapping from $\mathbb{R}^{2heads} \to \mathbb{R}^{heads}$. And $\mathbf{W}$ is initialized by all 0.5s.

It is simple to prove that both versions of GAT-PE are permutation equivariant and eigenvector basis equivariant.

We list the results of experiments on Cora as follows:

Table 6. Performance on the traditional link prediction tasks, measured in ROC AUC.

Method	Cora
VGAE-PE	87.96
GAT	90.03
PEG	**94.20**
GAT-PE-v1	82.23
GAT-PE-v2	**93.00**

We can see from the above table that GAT-PE-v2 perform better than GAT and GAT-PE-v1 perform the worst. For the failure of GAT-PE-v1, We speculate that this is because we cannot directly multiply weight coming from PE and GAT, we should average them. Besides, GAT-PE-v2 is not as good as PEG while GAT is better than VGAE-PE, it implies that we still can improve GAT-PE-v2 and leverage PE more delicately.

4.4 Equivariant RGCN with PE on heterogeneous graph.

RGCN[11] namely Relation Graph Convolutional Network, extends various classical graph tasks (e.g. entity classification, link prediction) to the heterogeneous graphs. It starts from the message-passing framework below

$$h_i^{(l+1)} = \sigma\left(\sum_{m\in\mathcal{M}_i} g_m\left(h_i^{(l)}, h_j^{(l)}\right)\right) \tag{18}$$

Where $h_i^{(l)}$ denotes the hiddenstate of node v_i in the l-th layer of the neural network, $\mathcal{M}_i$ denotes the set of incoming messages for node v_i and is often chosen to be identical to the set of incoming edges. $g_m(\cdot,\cdot)$ is typically chosen to be a (message-specific) neural network-like function or simply a linear transformation. To modeling the different kinds of relations in the heterogeneous graphs, RGCN modifies the above equation into the following form

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathscr{R}}\sum_{j\in\mathscr{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right) \quad (19)$$

Where $\mathscr{N}_i^r$ denotes the set of neighbor indices of node i under relation $r \in \mathscr{R}$, $c_{i,r}$ is a problem-specific normalization constant that can either be learned or chosen in advance.

To combine the RGCN with PE, the first problem came into our mind is that the original PE mentioned in PEG is based on homogeneous graphs, meaning that the positional encoding based on Laplacian Eigenmap is finished on a adjacent matrix derives from the graph that has only one relation between nodes. However, heterogeneous graphs have more than one relations. We fix this gap by viewing one heterogeneous graph as several homogeneous graphs. Then we could get several adjacent matrices and assign each node with several positional encodings. When adding PE into RGCN message-passing process, we need to carefully choose the positional encoding that belongs to the given relation. By doing so, we have the following message-passing framework

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathscr{R}}\sum_{j\in\mathscr{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)}p_{ij}^r + W_0^{(l)}h_i^{(l)}\right) \quad (20)$$

where $p_{ij}^r = MLP(||Z_i^r - Z_j^r||)$, Z_i^r is the positional encoding of node v_i with respect to relation r.

However, due to the limited time and experiment conditions, we are unable to finish the experiment of this part. We only run the original experiment of RGCN on dataset FB15k-237 and the result is close to the result in original paper.

References

[1] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[2] Martin Rosvall, Daniel Axelsson, and Carl T Bergstrom. The map equation. *The European Physical Journal Special Topics*, 178(1):13–23, 2009.

[3] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

[4] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[6] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *arXiv e-prints*, page arXiv:1606.09375, June 2016.

[8] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining Label Propagation and Simple Models Out-performs Graph Neural Networks. *arXiv e-prints*, page arXiv:2010.13993, October 2020.

[9] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and Deep Graph Convolutional Networks. *arXiv e-prints*, page arXiv:2007.02133, July 2020.

[10] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less Neural Networks: Teaching Old MLPs New Tricks via Distillation. *arXiv e-prints*, page arXiv:2110.08727, October 2021.

[11] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[12] Thomas Kipf and Max Welling. Variational graph auto-encoders. *ArXiv*, abs/1611.07308, 2016.

[13] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *CoRR*, abs/2005.00687, 2020.

[14] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[15] Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. Equivariant and stable positional encoding for more powerful graph neural networks, 2022.

[16] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. *CoRR*, abs/2102.09844, 2021.