Friend Recommendations with Self-Rescaling Graph Neural Networks

Xiran Song
National Engineering Research Center for Big Data Technology and System, Service Computing Technology and Systems Laboratory, Huazhong University of Science and Technology
Wuhan, China
xiransong@hust.edu.cn

Jianxun Lian
Microsoft Research Asia
Beijing, China
jianxun.lian@outlook.com

Hong Huang*
National Engineering Research Center for Big Data Technology and System, Service Computing Technology and Systems Laboratory, Huazhong University of Science and Technology
Wuhan, China
honghuang@hust.edu.cn

Mingqi Wu
Microsoft Gaming
Redmond, United States

Hai Jin
National Engineering Research Center for Big Data Technology and System, Service Computing Technology and Systems Laboratory, Huazhong University of Science and Technology
Wuhan, China

Xing Xie
Microsoft Research Asia
Beijing, China

ABSTRACT

Friend recommendation service plays an important role in shaping and facilitating the growth of online social networks. Graph embedding models, which can learn low-dimensional embeddings for nodes in the social graph to effectively represent the proximity between nodes, have been widely adopted for friend recommendations. Recently, *Graph Neural Networks* (GNNs) have demonstrated superiority over shallow graph embedding methods, thanks to their ability to explicitly encode neighborhood context. This is also verified in our Xbox friend recommendation scenario, where some simplified GNNs, such as LightGCN and PPRGo, achieve the best performance. However, we observe that many GNN variants, including LightGCN and PPRGo, use a static and pre-defined normalizer in neighborhood aggregation, which is decoupled with the representation learning process and can cause the scale distortion issue. As a consequence, the true power of GNNs has not yet been fully demonstrated in friend recommendations.

In this paper, we propose a simple but effective *self-rescaling network* (SSNet) to alleviate the scale distortion issue. At the core of SSNet is a generalized self-rescaling mechanism, which bridges the neighborhood aggregator's normalization with the node embedding learning process in an end-to-end framework. Meanwhile, we provide some theoretical analysis to help us understand the benefit of SSNet. We conduct extensive offline experiments on three large-scale real-world datasets. Results demonstrate that our proposed method can significantly improve the accuracy of various GNNs. When deployed online for one month's A/B test, our method achieves 24% uplift in adding suggested friends actions. At last, we share some interesting findings and hope the experience can motivate future applications and research in social link predictions.

CCS CONCEPTS

• **Information systems** → **Personalization**; **Social networks**; **Retrieval models and ranking**; • **Computing methodologies** → **Learning latent representations**.

KEYWORDS

Friend recommendation, graph neural networks, normalization

ACM Reference Format:
Xiran Song, Jianxun Lian, Hong Huang, Mingqi Wu, Hai Jin, and Xing Xie. 2022. Friend Recommendations with Self-Rescaling Graph Neural Networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539192

1 INTRODUCTION

Online social networks (OSNs) have been integrated with our daily lives. For example, we share personal lives with friends via Facebook or TikTok; we communicate with friends anywhere anytime using WhatsApp or WeChat; we build our professional networks by making connections on LinkedIn. To foster a healthy social network structure and promote community growth, friend recommendation (aka. social link prediction) becomes a common and critical feature for OSNs. At Xbox, we aspire to empower everyone to play not

*Hong Huang is the corresponding author.

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539192

only the games you want, but also with the people you want. One of the core services is friend recommendations, which helps users connect to their friends or favorite players, such that they may enjoy the fun together.

Traditionally, researchers manually extract some metrics [18, 19, 28], such as *Common Neighbors* (CN), *Adamic/Adar Index* (AA) [1], *Local Naive Bayes based Common Neighbors* (BCN) [20], and *Personalized PageRank* (PPR) [3], to represent the proximity between two nodes on a social network. Major drawbacks for metrics-based methods include: 1) handcrafted metrics can only cover a limited, static set of features for a social network, which are not comprehensive; 2) some metrics such as BCN and PPR require the high-order network context of a pair of nodes, which are not scalable for real-time serving; 3) recommended nodes are usually concentrated on the local neighborhood of the source node, which are highly homogeneous and aggravate the *Echo Chamber* and *Filter Bubble* problem [21, 25]. Thus, researchers resort to graph embedding techniques, such as DeepWalk [22] and Node2vec [10], to encode proximity information on the graph with low-dimensional embedding vectors. The relation between two nodes is measured by the interaction (such as dot product) between the corresponding embedding vectors, without requirements to retrieval graph context.

Recently, the success of *Graph Neural Networks* (GNNs) makes the research trend switch from shallow embedding to deep embedding [11, 15, 27]. Compared with shallow embedding methods such as Node2vec, GNNs have a graph context encoder, which aggregates information from the source node's neighborhood, so that the graph structure is explicitly encoded into the low-dimensional embedding vector. In our Xbox friend recommendation scenario, the superiority of GNNs over shallow graph embedding methods is also verified, and among various GNNs we have examined, LightGCN [12] and PPRGo [5] relatively perform the best. They happen to represent two types of simplified GNNs: LightGCN simplifies model parameters by removing the feature transformation and nonlinear activation components, and PPRGo simplifies the network structure by replacing the recursive neighborhood stacking operation with a pre-computed *Personalized PageRank* (PPR) neighborhood. Simplified GNNs are easier to scale and thus are more suitable for industrial recommendation systems.

Since the superiority of GNNs over shallow graph embedding comes from explicit neighborhood aggregation and encoding, how to aggregate information is undoubtedly critical. However, we observe one common weakness in the aggregation mechanism of many GNNs, including both the LightGCN and PPRGo. To aggregate the information of neighbors, (weighted) sum pooling or average pooling are commonly adopted by GNNs, which are simply set by heuristics and intuition. For example, LightGCN uses a weighted sum aggregator $\mathbf{e}_u^{(k+1)} = \sum_{i \in \mathcal{N}(u)} \alpha_{ui}\, \mathbf{e}_i^{(k)}$, in which a neighbor's feature is weighted by a normalized term $\alpha_{ui} = 1/\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}$. PPRGo uses a weighted sum aggregator with weights being neighbors' personalized PageRank score: $\mathbf{z}_i = \sum_{j \in \mathcal{N}(i)} \pi(i)_j\, \mathbf{h}_j$, where $\pi(i)$ is a sparse PPR vector preserving only top-k largest values. In both of the models, the weighting coefficients are **independent** of the learning of representations and kept **fixed**. We argue that this inflexible aggregating mechanism can easily cause undesirable effects such as the *scale distortion issue* (see more details in Section 2.3), and it is essential to bridge the normalizing coefficients with the representation learning and let the model automatically and adaptively determine an appropriate scaling factor according to each node's context.

To this end, we propose an effective approach to rescale the GNN embedding vector to an appropriate status in a self-adaptive manner. For a latent representation $\mathbf{h}$ generated by an original GNN, we feed it to a *self-rescaling network* (SSNet) which will produce a scalar factor γ, then adjust $\mathbf{h}$ with γ as the final output. We call this operation *representation rescaling*. The representation rescaling can be regarded as a model patch that can be mounted to various GNNs, such generalization ability avoids tuning specific components case by case. SSNet can be trained end-to-end with a base GNN model. We compare some alternative training methods such as pretrain-then-finetune and adversarial training, results demonstrate that end-to-end training is the best choice among them, which also makes SSNet's usage as convenient as plug-and-play.

We use three real-world social network datasets for experiments. To verify the effectiveness of SSNet comprehensively, we select the two most important tasks in the social link prediction scenario, including the candidate retrieval and friend ranking task, and conduct experiments with seven different GNNs. The results demonstrate that our approach can consistently improve different GNN models, and the performance gain in the candidate retrieval task is particularly significant. In summary, our main contributions include:

- *Problem Locating*. We highlight the vector distortion problem in aggregators of GNNs, which turns out to impact the representation quality severely and is overlooked in related literature.
- *Method Proposing*. We propose a light-weighted and effective method, SSNet, to remedy the problem in a representation rescaling approach. SSNet is model-agnostic and can be trained end-to-end with base GNNs to improve their performance. The source code is released at https://github.com/CGCL-codes/ssnet.
- *Experience Sharing*. We provide some theoretical analysis to help understand why SSNet helps GNNs. Besides, we share some interesting findings when applying graph embedding models to the Xbox friend recommendation scenario.
- *Result Verifying*. We conduct comprehensive offline experiments on three real-world social network datasets under two different types of tasks. Results verify the effectiveness of SSNet in enhancing various GNNs. Besides, a 24% gain in online A/B test also shows that high-quality graph embeddings can significantly improve the friend recommendation service.

2 METHODOLOGY

2.1 Problem Definition

Friend Recommendation. Given a social network $\mathcal{G} = (V, E)$, where V denotes the set of N users and E is the set of edges, E is usually formulated as an $N \times N$ adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, if there is a social relation from user i to user j, then $\mathbf{A}_{ij} = 1$, otherwise $\mathbf{A}_{ij} = 0$. The friend recommendation task is to predict the missing links in $\mathcal{G}$. From the graph embedding perspective, a GNN model learns to map each node $u \in V$ to a low-dimensional vector $\mathbf{e}_u$. The probability of an edge existing between node u and v is measured by the dot-product of embedding vectors: $y_{uv} = \mathbf{e}_u^T \mathbf{e}_v$.

2.2 GNN Preliminaries

We start by introducing the core parts of two GNN models – LightGCN [12] and PPRGo [5] – since they perform relatively best in friend recommendation tasks, and they also represent two typical types of GNNs.

2.2.1 LightGCN. LightGCN simplifies the classical GNN by removing the feature transformation and nonlinear activation components and achieves better performance. LightGCN is originally introduced for recommendation tasks (which is a user-to-item link prediction). We adjust the equations accordingly because, in our case, graph $\mathcal{G}$ is homogeneous, with all nodes on the graph being users. The graph convolution operation in LightGCN is defined as:

$$\mathbf{e}_u^{(k+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(k)} = \frac{1}{\sqrt{|\mathcal{N}_u|}} \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(k)} \quad (1)$$

where $\mathbf{e}_i^{(k)}$ indicates the hidden embedding vector of node i at the k-th convolutional layer. The only trainable model parameters are the embeddings at the 0-th layer: $\mathbf{e}_i^{(0)}$. The final representation of a user is a linear combination of all layers' embedding vectors:

$$\mathbf{e}_u = \sum_{k=0}^{K} \frac{1}{K+1} \mathbf{e}_u^{(k)} \quad (2)$$

2.2.2 PPRGo. Instead of stacking multiple convolutional layers to incorporate multi-hop neighborhood information, which leads to over smoothing problems as well as poor scalability, PPRGo conducts PPR for every root node and uses the PPR scores to select the most important local neighborhood for a given root node. Specifically, it first adopts the push-flow algorithm [2] to obtain the approximate PPR scores $\mathbf{\Pi}^{(\epsilon)}$ for all nodes, then truncates $\mathbf{\Pi}^{(\epsilon)}$ to contain only the top k largest entries for each row, which means that for user i, her new neighborhood is constructed as the users with top k largest scores according to $\pi(i)$. The final representation $\mathbf{z}_i$ is a weighted sum of the neighborhood representations:

$$\mathbf{z}_i = \sum_{j \in \mathcal{N}_i^k} \pi(i)_j \mathbf{e}_j \quad (3)$$

2.3 The Scale Distortion Issue

We observe that the neighborhood aggregating operation is prone to cause the vector scale distortion issue. In LightGCN, neighbor embeddings are normalized according to node degree, i.e. $1/(\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|})$, to avoid the scale of embeddings from exploding during stacked aggregation. However, this type of weighting coefficient is hard-coded with heuristics, which lacks the flexibility to adjust to different contexts. For better understanding, here we provide two illustrations. First, the normalizer $1/(\sqrt{|\mathcal{N}_u|})$ may not ideally reflect the necessary squash level for a node with degree $|\mathcal{N}_u|$. Suppose user u and v belong to the same social circle. u has five friends: $\mathcal{N}_u = \{p_1, p_2, ..., p_5\}$. v is also connected to these five persons, besides, v has five more friends whom u has not connected to: $\mathcal{N}v = \mathcal{N}_u \bigcup \{p_6, p_7, ..., p_{10}\}$. Then, an one-layer LightGCN model will represent user u by:

$$\mathbf{e}_u = \frac{1}{\sqrt{5}} \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i \overset{\text{def}}{=\!=} \frac{1}{\sqrt{5}} \mathbf{F}_1 \quad (4)$$

Similarly, for user v we have:

$$\mathbf{e}_v = \frac{1}{\sqrt{10}} \sum_{i \in \mathcal{N}_v} \frac{1}{\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i = \frac{1}{\sqrt{10}} \mathbf{F}_1 + \frac{1}{\sqrt{10}} \mathbf{F}_2 \quad (5)$$

In terms of the vector magnitude, for u, we have: $||\mathbf{e}_u||^2 = \frac{1}{5}\mathbf{F}_1^2$; while for v we have: $||\mathbf{e}_v||^2 = \frac{1}{10}\mathbf{F}_1^2 + \frac{1}{10}\mathbf{F}_2^2 + \frac{1}{5}\mathbf{F}_1 \cdot \mathbf{F}_2$. Since user u and v belong to the same social circles and neighborhood features are highly homogeneous and positively correlative, it is reasonable to assume that $\mathbf{F}_1^2 \approx \mathbf{F}_2^2$ and $\mathbf{F}_1 \cdot \mathbf{F}_2 > 0$. Thus, $||\mathbf{e}_v||^2 > ||\mathbf{e}_u||^2$, we have the magnitude distortion issue when scaling with $|\mathcal{N}_u|$ in the sum aggregator.

Next, let us consider the scale distortion issue in two nodes u and v with the same degree n. Suppose user u is in the center of a community, then most of her neighbors belong to the same community; user v is at the boundary of two communities, which means her neighbors are split into different communities. For both users, their magnitude is calculated like:

$$||\mathbf{e}_u||^2 = ||\frac{1}{\sqrt{n}} \sum_{i \in \mathcal{N}_u} \widetilde{\mathbf{e}}_i||^2 \quad (6)$$

$$= \frac{1}{n}(\sum_{i \in \mathcal{N}_u} ||\widetilde{\mathbf{e}}_i||^2 + \sum_{i,j \in \mathcal{N}_u, i \neq j} \widetilde{\mathbf{e}}_i \cdot \widetilde{\mathbf{e}}_j) \quad (7)$$

where we denote $\widetilde{\mathbf{e}}_i = \frac{1}{\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(0)}$. Suppose the initial embeddings $\mathbf{E}^{(0)}$ are all unit vectors, so they are well aligned in magnitude. Thus, $||\mathbf{e}_u||^2 = C + \frac{1}{n} \sum_{i,j \in \mathcal{N}_u, i \neq j} \widetilde{\mathbf{e}}_i \cdot \widetilde{\mathbf{e}}_j$, where C is a constant which is equal for both u and v. Since user u is in the center of a community while user v is at the boundary of a community, the neighborhood features of user u are more positively correlated, so that we have $\sum_{i,j \in \mathcal{N}_u, i \neq j} \widetilde{\mathbf{e}}_i \cdot \widetilde{\mathbf{e}}_j > \sum_{i,j \in \mathcal{N}_v, i \neq j} \widetilde{\mathbf{e}}_i \cdot \widetilde{\mathbf{e}}_j$. Thus, $||\mathbf{e}_u||^2 > ||\mathbf{e}_v||^2$, the vector scale is distorted between user u and v.

2.4 The Self-Rescaling Network

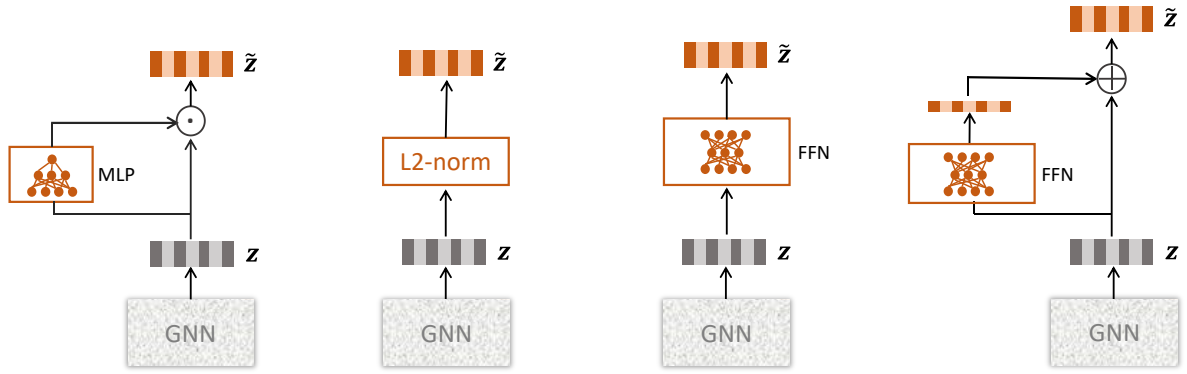
We argue that the static scaling coefficients in GNNs, such as $(\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|})^{-1}$ in LightGCN and $\pi(i)_{top-k}$ in PPRGo, will impact their representation learning ability. It is of great importance to bridge the scaling factor with neighborhood aggregators, so that GNNs can determine the degree of embedding scaling in an adaptive manner according to the context of a specific node's neighborhood. To this end, we propose a simple but effective self-adaption method to stretch an embedding vector $\mathbf{z} \in \mathbb{R}^D$ generated from an original GNN. All we need is an extra scaling network $G(\cdot)$, which takes $\mathbf{z}$ as input and returns a scaling factor:

$$G(\mathbf{z}) = \sigma(\mathbf{w}_2 \, Tanh(\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1) + b_2) \quad (8)$$

where $\mathbf{W}_1 \in \mathbb{R}^{H \times D}$, $\mathbf{b}_1 \in \mathbb{R}^H$, $\mathbf{w}_2 \in \mathbb{R}^H$, $b_2 \in \mathbb{R}$ are the trainable parameters of $G(\cdot)$, σ is the Sigmoid function to force the output value of $G(\mathbf{z})$ in $(0, 1)$. Eq. 8 is a 2-layer MLP, we can add more hidden layers in the MLP if necessary. The original GNN embedding $\mathbf{z}$ will be adjusted by this scaling factor:

$$\tilde{\mathbf{z}} = G(\mathbf{z}) \cdot \mathbf{z} \quad (9)$$

$G(\cdot)$ can adaptively control the scaling intensity according to the embedding vector. Besides, this self-adaption method is GNN model-agnostic. It can be applied to various types of GNNs' representation vectors, which demonstrates generalization ability.



(a) Scaling Network ✓ (b) L2 Normalization ✗ (c) Feed-Forward Net ✗ (d) Additive Network ✗

Figure 1: Illustrations of different dynamic scaling methods, (a) is our proposed SSNet, (b-d) are three other alternatives.

To fulfill the goal of self-adaption, technically, there are some alternative methods other than SSNet, such as the ones illustrated in Figure 1. Formally, they include:

• *L2 Normalization.* L2 normalization is commonly adopted to transform embedding vectors into unit vectors so that all of them have magnitude being exactly 1 unit. From the expression $\tilde{\mathbf{z}} = \mathbf{z}/||\mathbf{z}||^2$ we can observe that L2 normalization is a special case of scaling neural network, with $G(\mathbf{z})$ being $1/||\mathbf{z}||^2$. However, normalizing everything to a unit vector is too strict, which may hurt models' performance.

• *Feed-Forward Network.* We can also count on a fully connected *Feed-Forward Network* (FFN) to transform the original embedding vector to a new, well aligned vector space. FFN is a key component in Transformers [26]. It consists of two linear transformations with a non-linear activation in between:

$$\tilde{\mathbf{z}} = FFN(\mathbf{z}) = \mathbf{W}_2 Tanh(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1) + \mathbf{b}_2 \tag{10}$$

where $\mathbf{W}_2 \in \mathbb{R}^{D\times H}$ and $\mathbf{b}_2 \in \mathbb{R}^D$.

• *Additive Network.* Instead of generating a scaling factor to multiply the original embedding vector, we can also try the additive operation. In this regard, the network will produce a bias vector and then add to the original embedding vector:

$$\tilde{\mathbf{z}} = \tilde{G}(\mathbf{z}) + \mathbf{z}\,, \quad \tilde{G}(\mathbf{z}) = \mathbf{W}_2 Tanh(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1) + \mathbf{b}_2 \tag{11}$$

However, in practice we find that those alternative methods are not performing as well as the scaling network. We summarize their performance in Section 3.3.1.

2.5 Optimization

To train a GNN model with SSNet, there are three feasible strategies:

• *End-to-End Training.* The number of new parameters brought by SSNet is $O(HD+HH)$, where D and H denote the dimension of node embedding of GNN and hidden layers in SSNet, respectively. The parameter of the base embedding table of GNN is $O(ND)$. Because D (such as 64 in our experiments) is much smaller than N (such as 3 million), additional new parameters in SSNet are negligible. And all of them are differential, so a GNN model equipped with SSNet can be directly trained from scratch in an end-to-end manner.

• *Pretrain-then-Finetune.* Since SSNet is a plug-and-play like component whose parameters are not blended with GNN's parameters, another intuitive strategy is to first pretrain the original base GNN model until convergence, then only finetue the SSNet's parameters, so that SSNet serves as an embedding adjustment module.

• *Adversarial Training.* Both of the above strategies jointly train SSNet and base GNN model under the same recommendation signals. Another idea is to provide some auxiliary signals for the SSNet to guide its training process, so that the final embeddings can possess some properties that match our prior knowledge. As a first reasonable try, we hypothesize the adjusted embeddings from SSNet should be unbiased in regard of node degree. Thus, we propose an adversarial training strategy: there is a discriminator that tries to detect the node degree information from the node embedding vector; the GNN and SSNet not only minimize the recommendation loss, but also minimize the discriminator's accuracy (which serves as an auxiliary loss), so that the adjusted embedding vector does not leak the node degree information.

Through experiments (see Section 3.3.2) we find that the end-to-end training strategy achieves pretty good performance. Considering that it is much simpler and easier than the other two approaches, it is the suggested and default strategy in this paper.

3 EXPERIMENTS

3.1 Experiment Settings

3.1.1 Datasets. We use three real-world social network datasets for experiments. The **Xbox** dataset is constructed from Xbox gaming social network. We extract a subgraph which contains 3 million users from US, one of the major markets of Xbox. Besides, we use two largest social networks – **Pokec** and **LiveJournal** – from the SNAP public datasets (https://snap.stanford.edu/data/index.html#socnets). We leave more descriptions of datasets and basic statistics to A.2 in the appendix.

3.1.2 Tasks. We evaluate models' performance on two important tasks in recommender systems: **candidate retrieval** and **friend ranking**. The major difference between the two tasks lies in the selection of negative instances for evaluation: for candidate retrieval, we try to recall the positive candidate from the entire graph (which

Table 1: Overall performance of different GNN models on the candidate retrieval task. We highlight the cells when SSNet improves the base GNN model.

	Pokec			LiveJournal			Xbox		
	HR@100	HR@300	NDCG@300	HR@100	HR@300	NDCG@300	HR@100	HR@300	NDCG@300
Node2vec	0.0383	0.0755	0.0146	0.0384	0.0636	0.0132	0.0218	0.0360	0.0080
GraphSAGE (pool)	0.0629	0.1256	0.0239	0.1453	0.2248	0.0487	0.0411	0.0650	0.0140
GraphSAGE (pool) + SS	0.0606	0.1233	0.0228	0.1868	0.2743	0.0603	0.0498	0.0768	0.0164
GraphSAGE (LSTM)	0.0769	0.1502	0.0285	0.1840	0.2711	0.0592	0.0643	0.0944	0.0212
GraphSAGE (LSTM) + SS	0.0737	0.1510	0.0278	0.2206	0.3115	0.0696	0.0684	0.1010	0.0222
GIN	0.0789	0.1474	0.0277	0.1938	0.2587	0.0616	0.0394	0.0599	0.0136
GIN + SS	0.0754	0.1459	0.0271	0.1979	0.2666	0.0612	0.0441	0.0645	0.0143
GAT	0.0372	0.1202	0.0178	0.0322	0.1100	0.0161	0.0259	0.0590	0.0096
GAT + SS	0.0893	0.1784	0.0313	0.2119	0.3225	0.0635	0.0879	0.1248	0.0287
SAGN	0.0554	0.1178	0.0209	0.1607	0.2336	0.0525	0.0356	0.0558	0.0126
SAGN + SS	0.0632	0.1319	0.0234	0.1800	0.2573	0.0566	0.0544	0.0784	0.0179
FAGCN	0.0901	0.1719	0.0301	0.1083	0.2163	0.0347	0.0286	0.0701	0.0120
FAGCN + SS	0.1349	0.2300	0.0449	0.2431	0.3249	0.0743	0.0874	0.1231	0.0271
LightGCN	0.0654	0.1468	0.0236	0.0537	0.1608	0.0240	0.0506	0.0821	0.0164
LightGCN + SS	0.1645	**0.2605**	0.0536	0.2604	0.3432	0.0747	0.0893	0.1247	0.0287
PPRGo	0.1604	0.2460	0.0520	0.2824	0.3666	0.0841	0.1095	0.1382	0.0356
PPRGo + SS	**0.1727**	**0.2602**	**0.0554**	**0.2903**	**0.3761**	**0.0861**	**0.1169**	**0.1460**	**0.0379**

contains a few million nodes). We follow this time-consuming evaluation method because [17] points out that conducting negative sampling will make evaluation metrics inconsistent with their exact version in candidate retrieval. *Hit-Rate* (HR@100, HR@300) and *Normalized Discounted Cumulative Gain* (NDCG@300) are adopted as evaluation metrics. For friend ranking, we sample 99 nodes as negative instances from the 2-hop neighborhood of the source node in the positive pair. This is to simulate the scenario that the ranker needs to rank the positive node from a small set of strong negative candidates. NDCG and *Area Under the Curve* (AUC) are used for evaluating the friend ranking task.

3.1.3 Base GNNs. To verify the generalization ability of SSNet, we experiment with a variety of GNNs as base models, covering simple neighborhood aggregator including **GraphSAGE** [11],**GIN** [30], **LightGCN** [12], **PPRGo** [5], and attentive neighborhood aggregator including **GAT** [27], **SAGN** [23], **FAGCN** [4]. More introductions of these base GNNs, as well as training settings, can refer to A.1 and A.4 in the appendix.

3.2 Overall Performance

Table 1 and 2 report the overall performance of the self-rescaling mechanism on three datasets and two recommendation tasks. In the tables, the symbol *+ SS* indicates applying the SSNet on the base model. We have the following observations.

• For almost all the settings (a setting means a combination of [base GNN, dataset, task]), adding the self-rescaling operation can improve the base GNN. For some certain settings, the performance gain is quite significant, e.g., in [LightGCN, Pokec, Retrieval], self-rescaling increases the HR@100 by 150%. This demonstrates the effectiveness as well as the necessity of self-rescaling mechanism for GNNs in social link prediction tasks.

• The overall improvement on the ranking task is much smaller than on the retrieval task. This is because the retrieval task is a more open task, in which the model needs to find relevant candidates within the entire item space (which contains millions of items). Thus, a small interference can lead to a substantial difference in top-k results. In contrast, the information uncertainty in the ranking task has been greatly reduced due to that the candidates to be ranked are already narrowed down to a small scope.

Table 2: Overall performance on the friend ranking task. Cells are highlighted when SSNet improves the base GNN.

	Pokec		LiveJournal		Xbox	
	AUC	NDCG	AUC	NDCG	AUC	NDCG
Node2vec	0.6645	0.2657	0.6486	0.2748	0.6982	0.3110
SAGE(pool)	0.7408	0.2984	0.7686	0.3387	0.7405	0.3411
SAGE+SS	0.7516	0.3020	0.7905	0.3621	0.7451	0.3521
SAGE(LSTM)	0.7670	0.3141	0.7925	0.3600	0.7699	0.3765
SAGE+SS	0.7715	0.3175	0.8110	0.3808	0.7697	0.3805
GIN	0.7672	0.3250	0.7801	0.3766	0.7132	0.3279
GIN + SS	0.7669	0.3249	0.7795	0.3750	0.7182	0.3350
GAT	0.7875	0.3324	0.7642	0.3143	**0.7851**	0.3954
GAT + SS	0.7875	0.3371	0.8058	0.3739	**0.7859**	**0.4074**
SAGN	0.7542	0.3066	0.7839	0.3615	0.7253	0.3361
SAGN + SS	0.7633	0.3117	0.7962	0.3723	0.7382	0.3536
FAGCN	0.7687	0.3212	0.7942	0.3613	0.7526	0.3323
FAGCN + SS	0.7974	0.3606	0.8242	0.4195	0.7799	0.3914
LightGCN	0.7752	0.3327	0.8110	0.3768	0.7569	0.3588
LightGCN+SS	0.8169	0.3922	**0.8501**	**0.4535**	0.7808	0.4003
PPRGo	0.8087	0.3840	0.8340	0.4174	0.7781	0.3966
PPRGo + SS	**0.8193**	**0.3988**	0.8395	0.4229	0.7825	0.4056

• The improvement of self-rescaling on PPRGo and GraphSAGE is less significant than on other GNNs like LightGCN, which is in expectation. PPRGo and GraphSAGE use a fixed number of neighbors for each node, and their vector scale distortion issue is less serious.

• Among all the base GNNs, PPRGo performs the best (and adding self-rescaling can further improve it) in the retrieval task. One possible reason is that for social link predictions, the local community

is the key to identify a user and her social relations. In PPRGo, there is a Personalized PageRank step to effectively detect the most important neighborhood and the local community.

3.3 Variants Study

3.3.1 Alternative Transformation Methods. We explore whether there exist some better choices for representation modulation rather than using the self-rescaling network. As listed in Figure 1, there are three other common approaches to modulate the embedding vector, including L2 normalization, going through a feed-forward network, and using an additive network. All of them have the potential to do self-adaption. Table 3 reports the results of the candidate retrieval task. Since the trend on the friend ranking task is similar, for conciseness, we move Table A3 to the appendix. SSNet outperforms the other three approaches consistently on different datasets, on different tasks, and with different base GNNs, which verifies that the superiority of the self-rescaling network cannot be easily replaced by some other trivial approaches. In addition, the three variants' performance is not stable. For example, FFN performs well on the candidate retrieval task, but it performs badly on the friend ranking task; L2 normalization only performs well on the Pokec and LiveJournal dataset with the candidate retrieval task and LightGCN, while for the rest of the cases, it performs badly. In this regard, the performance of the scaling network is very stable.

Table 3: Comparisons of model variants on the candidate retrieval task. The metrics are HR@300 and NDCG@300.

	Pokec		LiveJournal		Xbox	
	HR	NDCG	HR	NDCG	HR	NDCG
LightGCN						
Base	0.1468	0.0236	0.1608	0.0240	0.0821	0.0164
L2-norm	0.1656	0.0337	0.2976	0.0732	0.0591	0.0162
FFN	0.1754	0.0349	0.2528	0.0633	0.0779	0.0179
Additive	0.1541	0.0252	0.1590	0.0231	0.0792	0.0135
SSNet	**0.2605**	**0.0536**	**0.3432**	**0.0747**	**0.1247**	**0.0287**
PPRGo						
Base	0.2460	0.0520	0.3666	0.0841	0.1382	0.0356
L2-norm	0.1869	0.0396	0.3072	0.0713	0.0862	0.0247
FFN	0.2053	0.0427	0.3243	0.0738	0.1085	0.0277
Additive	0.2468	0.0522	0.3487	0.0779	0.1232	0.0311
SSNet	**0.2602**	**0.0554**	**0.3761**	**0.0861**	**0.1460**	**0.0379**

3.3.2 Training Strategies Comparison. In Section 2.5 we introduce three training strategies. Table 4 reports the results on the Xbox dataset with PPRGo as the base GNN. We do not observe a significant difference between the three strategies. The conclusions are similar on the other two datasets. Since end-to-end training is the most convenient approach, we adopt it as the default training method.

3.4 Pattern Analysis

Next, we study what types of information SSNet learns. As discussed in Section 2.3, some factors may cause the scale distortion issue, such as the node's degree and neighborhood's assortativity level. To verify this motivation, we can test some observable patterns of the scaling factor with certain coefficients. In Figure 2(a,b) we plot the correlation between the self-rescaling factor (on the y-axis) and the node's degree or the embedding vector's magnitude (on the x-axis), on the Xbox dataset. We can observe that with the increase of node degree/vector magnitude, SSNet tends to shrink the vector, which implies that the original static aggregator over amplifies graph information for nodes with high degree. Figure 2(c) shows that the adjusted embeddings become more concentrated in a mean status in terms of magnitude. As a benefit, degree bias in recommendations can be alleviated. For example, in Figure 2(d), we can observe that the original LightGCN model tends to recommend high-degree nodes (such as a degree approaching 1000) to users, indicating a severe *popularity bias*. After using SSNet as a treatment, the degree bias is significantly alleviated, ordinary nodes (such as degree between 10 and 100) get more chances for exposure.

Table 4: A comparison of different training strategies. The dataset is Xbox. The base GNN is PPRGo.

	Recall		Ranking	
	HR@100	HR@300	AUC	NDCG
End2end	0.1169	0.1460	0.7825	0.4056
Pretrain-finetune	0.1142	0.1416	0.7803	0.4006
Adversarial	0.1145	0.1432	0.7816	0.4052

3.5 Online Experiments

Unlike other recommendation scenarios, such as e-commerce or online news recommendations, where items may refresh frequently and users tend to interact with recommended items promptly, the friend recommendation scenario owns a much slower feedback nature, and we need to reserve enough time and traffic for a reliable online test. Thus, it is hard to conduct online ablation study in a fine-grained level. Instead, we launch only one group of online A/B test, with the treatment group containing our best graph embeddings (PPRGo + SSNet) and the control group without graph embedding signals, to verify how graph embedding improves friend recommendations. More deployment details can refer to Section 4.2.3. The online experiment lasts from Dec 1, 2021 to Jan 1, 2022, covering about 10% of the main traffic in US market. Users can see the suggested friends on the homepage of Xbox console. Results show that the *add suggested friends actions* is uplifted by 24%. In addition, driven by the higher quality of suggested friends, we observe a few chain reactions, including 267% uplift in *view user profile*, 89% uplift in *search players*, 109% uplift in *likes in activity feed*, and the most important one, 19% uplift in *join friends' game*. The results fully demonstrate the importance of friend recommendation service in fostering an engaged gaming community.

4 DISCUSSIONS AND EXPERIENCE

4.1 Theoretical Analysis

We notice that the potential benefits of the self-rescaling network can be further interpreted from two different perspectives.

4.1.1 From the Spectral View. In this section, we will analyze our scaling network from the graph spectrum's perspective. For simplicity, we suppose the base model is LightGCN.

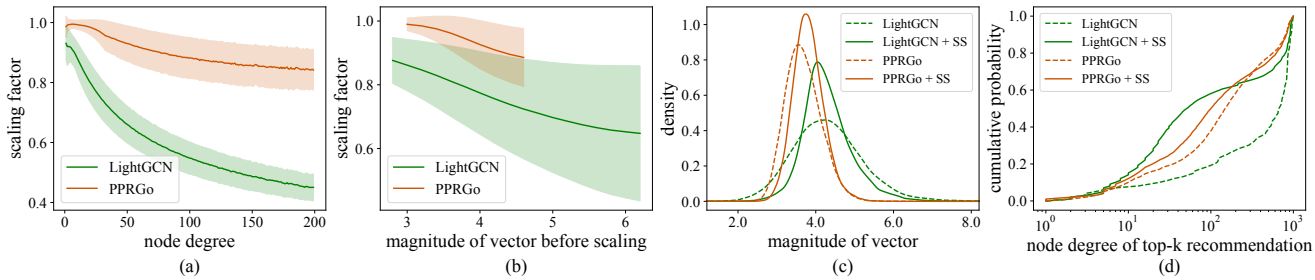


Figure 2: Pattern analysis for SSNet on the Xbox dataset. In (a) and (b), the shaded area indicates one standard deviation around the mean value. In (b), to remove noises which cause severe fluctuation in the figure, we remove vectors whose magnitude ranked in the bottom/top 2.5%.

The graph Laplacian matrix is defined as: $\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ where $\mathbf{D} = diag(\sum_j A_{1,j}, \sum_j A_{2,j}, ..., \sum_j A_{N,j})$ is the diagonal degree matrix and $\mathbf{I}_N$ is the identity matrix. From the view of graph signal processing [8], a graph signal $\mathbf{x}$ is a length N vector with x_i indicating the feature value for node i, and one physical interpretation for the graph Laplacian matrix is in calculating the smoothness of the graph signal $\mathbf{x}$:

$$S(\mathbf{x}) = \sum_{i,j} A_{i,j}(x_i - x_j)^2 = \mathbf{x}^T\mathbf{L}\mathbf{x} \tag{12}$$

A smaller $S(\mathbf{x})$ indicates a smoother signal. In the social link prediction scenario, users tend to connect with like-minded friends, so most of the social networks should demonstrate strong smoothness. For an undirected graph, $\mathbf{L}$ is a real symmetric matrix, we can have its spectral decomposition: $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ where the columns of $\mathbf{U}$ are N eigenvectors, which can serve as a set of orthogonal bases for the graph space. The *Graph Fourier Transform* (GFT) transforms an original graph signal $\mathbf{x}$ into the orthogonal spectral domain by $\hat{\mathbf{x}} = \mathbf{U}^T\mathbf{x}$. The inverse GFT operation is $\mathbf{x} = \mathbf{U}\hat{\mathbf{x}}$. According to the Fourier Transform theory, performing convolution in the spatial domain corresponds to performing multiplication in the spectral domain:

$$f * \mathbf{x} = \mathbf{U}((\mathbf{U}^T f) \odot (\mathbf{U}^T\mathbf{x})) = \mathbf{U}g_\theta(\mathbf{\Lambda})\mathbf{U}^T\mathbf{x} \tag{13}$$

where $*$ denotes the convolution operation, f denotes the convolution kernel function, $\odot$ denotes the element-wise product of vectors. When designing model from the spectral view, we do not need to know what f is, so researchers directly study how to parameterize the kernel g_θ. Spectral CNN [6] directly uses N trainable parameters to model the kernel:

$$g_\theta(\mathbf{\Lambda}) = diag(\{\theta_i\}_{i=1}^N) \tag{14}$$

The major drawbacks are two-fold: (i) it is not localized in space and (ii) too many free parameters are involved. ChebNet [7] uses a polynomial filter to parameterize g_θ:

$$g_\theta(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k \mathbf{\Lambda}^k \tag{15}$$

so only K parameters are involved. The convolution kernel of LightGCN is a special case of ChebNet, with $K = 1$ and $\theta = \frac{1}{2}\theta_0 = -\theta_1$:

$$g_\theta(\mathbf{\Lambda}) = \theta(\mathbf{I}_N - \mathbf{\Lambda}) \tag{16}$$

By comparing Eq.(14) with Eq.(15) and Eq.(16), it is easy to observe that the SpectralCNN and the ChebNet/LightGCN go to two different extremes: too many parameters (overfit) or too few parameters (underfit). Our proposed scaling network can be regarded as a compromise of the two extreme cases: the scaling network produces a personalized parameter for θ_i according to node i's neighborhood context, so that θ_i is neither global fixed nor fully free.

4.1.2 From the Graph Isomorphism View. Keyulu et al. [30] propose an interesting framework for analyzing the expressive power of GNNs from the view of graph isomorphism. They point out that ideally, a maximally powerful GNN can distinguish different graph structures by representing them as different vectors in the embedding space, which implies solving the graph isomorphism problem. A key conclusion from their theory is that if the neighbor aggregator and graph-level readout functions are injective, then the GNN is as powerful as the *Weisfeiler-Lehmann* (WL) test. Guided by this theory, they conclude that a maximally powerful GNN should aggregate and update node representation iteratively with

$$\mathbf{e}_v^{(k)} = \phi(\mathbf{e}_v^{(k-1)}, f(\{\mathbf{e}_u^{(k)} : u \in \mathcal{N}(v)\})) \tag{17}$$

where ϕ and f are injective functions. They propose a simple instantiation of Eq 17, which is called GIN:

$$\mathbf{e}_v^{(k)} = MLP^{(k)}\left((1+\epsilon^k)\cdot\mathbf{e}_v^{(k-1)}, \sum_{u\in\mathcal{N}(v)} \mathbf{e}_u^{(k)}\right) \tag{18}$$

because MLPs have the universal approximation ability and meanwhile can represent the composition of functions $f^{(k)} \circ \phi^{(k)}$. Another key conclusion is that both the mean and max-pooling aggregators are not injective, and they are less as powerful as the sum aggregator. However, SGC, LightGCN, and PPRGo use the (weighted) mean aggregator, which loses a certain expressive power to avoid the scale of embeddings explosion with stacking graph convolution operations. From this point of view, our scaling network combines the strengths of GIN and simplified GNNs like SGC/LightGCN/PPRGo: firstly, according to the empirical conclusions of simplified GNNs, MLPs in GCNs inherit unnecessary complexity and even make models hard to train. In our framework, MLPs only act on the learning of scaling factors, leaving the backbone of aggregation and transformation for node representation remaining as their original neat architectures. Secondly, according to the theoretical conclusions of GIN, mean aggregator is not injective, which cannot represent repeating neighbor features. Our

scaling factor can make up for this shortcoming by modulating the magnitude of embedding vectors to a proper degree, so that repeating neighbor features will not diminish and meanwhile the scale of vectors will not explode.

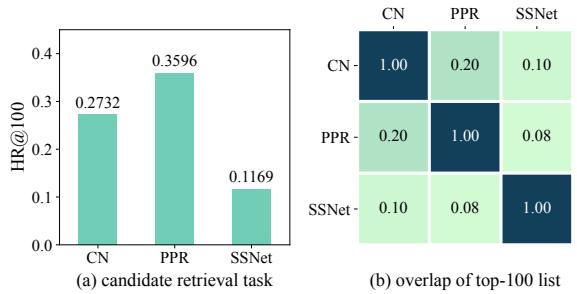


Figure 3: Comparisons of different retrieval methods. The dataset is Xbox.

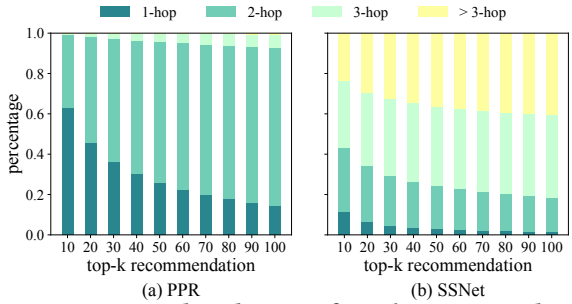


Figure 4: Distance distribution of top-k recommendations relative to the source node. The dataset is Xbox.

4.2 Empirical Findings

4.2.1 Dense Retrieval vs. Rule-Based Retrieval. Embedding based models, equipped with ANN search toolkit such as Faiss [13], are widely used as a dense retrieval approach for candidate recall. Although graph embeddings possess many advantages such as automatically graph context learning, we find that for the candidate recall stage, dense retrieval is a complement, rather than a replacement, for traditional rule-based retrieval such as CN-based and PPR-based methods. From Figure 3(a) we can observe that dense retrieval along still underperforms CN and PPR, however, as revealed in Figure 3(b), different retrieval methods catch different groups of candidates, e.g., the overlapping ratio of PPR and PPRGo(SSNet)'s top-100 list are merely 8%. In this regard, different retrieval methods are complemented to each other and they together comprise a comprehensive and diverse retrieval mechanism.

4.2.2 Break the Filter Bubble. Rule-based retrieval methods can easily cause the *filter bubble* or *echo chamber* problem [21, 25], which means that users' social circles become increasingly homogeneous and narrow, the majority information outside user's ego-network remains untouched. This is obviously undesirable and is bad for a social network's long-term development. Figure 4 compares the hop distribution in top-k recommendations of PPR and PPRGo(SSNet). We omit the CN because it can only retrieve the 2-hop neighborhood. As in expectation, most of PPR's recommended nodes are located in the source node's small ego-network, such as 1-hop and 2-hop neighborhood. In contrast, graph embedding method's results have a broader coverage of the social network, e.g., in top-100 recommendations, 40.8% of recommendations locate in the 3-hop and 40.5% come from higher-order neighborhood. Actually, among the newly added social links (u, v) of the Xbox platform, for 27.2% cases, v is in the 3-hop neighborhood of u and for 21.7% cases, v is even in the higher-hop, which demonstrates that we indeed need the friend finding algorithms to break the filter bubble.

4.2.3 Online Rankers. Our production ranking model is based on *Decision Tree* (DT). Features include a few very simple network metrics which at most require 2-order network context, such as indegree/outdegree of the source/target nodes and the number of *common neighbors* (CN), to make the online inference lightweight. Graph embedding information will turn into a scalar feature, which is the dot product score of the corresponding vectors. The advantages of feature-based rankers like DT are two-fold: (1) both simple features (such as degree and CN) and advance features (such as embedding similarity) are included. When a user has added a new friend, his ego-network change can be reflected by the simple features, so the ranker can perceive the changes in real-time without waiting until the graph embeddings got updated; (2) DT is explainable, we can print the decision path to see why the model recommends some users as potential friends. Besides, it is very convenient to conduct feature importance test in DT. In our experiments, embedding scores account for 65% of overall feature importance, and by adding embedding scores as features we observe a 20.7% gain in offline NDCG. This demonstrates that graph embedding signal plays a key role in friend recommendations.

5 RELATED WORK

5.1 Social Link Prediction

Social link prediction is an important task for online social networks. Typical applications include *People You May Know* on Facebook/LinkedIn and *Who to Follow* on Twitter. In the past, researchers manually extract some metrics [18, 19, 28], such as CN, AA, and BCN, to represent the proximity between two nodes in the social network. In recent years, graph embedding techniques, such as DeepWalk [22], Node2vec [10], and LINE [24], provide a different view of solving the social link prediction problem. Each node is represented as an embedding vector, which can potentially encode comprehensive information for the node on the graph implicitly and automatically. The proximity between two nodes can be measured as the interaction between the corresponding embedding vectors. In this paper, we focus on the graph embedding manner.

5.2 Simplified GNNs

Graph neural networks have achieved great success in representation learning for graph structure data. Recently, some researches show that simplifying GNN models could bring the model improvement on both efficiency and effectiveness. To be specific, this line of works could be mainly categorized into two groups:

1) Simplifying GNN parameters. SGC [29] proves that removing intermediate non-linear activation during aggregation could not only reduce computation complexity but also achieve competitive performance. Similarly, after removing the feature transformation and non-linear activation components in GCN, LightGCN [12] successfully makes the model easier to optimize and achieves better performance in link prediction tasks.

2) Simplifying the network structure. SIGN [9] decouples the GNN model into two parts, including pre-processing and post-classification, and utilizes the inception mechanism during pre-processing to enhance the expression power. SAGN [23] further introduces an attention mechanism to the post-classification component, which helps the model to adaptively gather neighborhood information among different hops. PPRGo [5] simplifies the network architecture by utilizing a propagation scheme derived from PPR, which could generate a small neighbor set for each node.

6 CONCLUSION

In this paper, we discuss friend recommendations with GNNs. We observe that in many GNNs' architectures, the normalizer of neighborhood aggregator is set by heuristic, kept fixed, and is decoupled from the representation learning, which may hurt the quality of representation vectors. Thus, we propose the *self-rescaling network* (SSNet), which rescales the representation in the last step, to bridge the neighborhood normalizer and the representation learning in an end-to-end framework. SSNet is a model-agnostic mechanism, and it can be applied to various GNNs. We conduct extensive experiments on three large-scale datasets and two different tasks, with seven base GNN models. Results demonstrate that SSNet significantly improves the quality of various GNNs' representations.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (No.62172174, 62127808).

REFERENCES

[1] Lada A. Adamic and Eytan Adar. 2003. Friends and Neighbors on the Web. *Social Networks* 25, 3 (2003), 211–230.
[2] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. 475–486.
[3] Ziv Bar-Yossef and Li-Tal Mashiach. 2008. Local Approximation of Pagerank and Reverse Pagerank. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)*. 279–288.
[4] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. 2021. Beyond Low-frequency Information in Graph Convolutional Networks. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI'21)*. 3950–3957.
[5] Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling Graph Neural Networks with Approximate PageRank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'20)*. 2464–2473.
[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR'14)*. 1–14.
[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proceedings of the Annual Conference on Neural Information Processing Systems 2016 (NIPS'16)*. 3837–3845.
[8] Xiaowen Dong, Dorina Thanou, Laura Toni, Michael M. Bronstein, and Pascal Frossard. 2020. Graph Signal Processing for Machine Learning: A Review and New Perspectives. *IEEE Signal Processing Magazine* 37, 6 (2020), 117–127.
[9] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. 2020. SIGN: Scalable Inception Graph Neural Networks. In *Proceedings of Graph Representation Learning and Beyond Workshop at the 37th International Conference on Machine Learning (GRL+@ICML'20)*. 1–9.
[10] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. 855–864.
[11] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. 1025–1035.
[12] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'20)*. 639–648.
[13] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547.
[14] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR'15)*. 1–15.
[15] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR'17)*. 1–14.
[16] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *Proceedings of the 7th International Conference on Learning Representations (ICLR'19)*. 1–15.
[17] Walid Krichene and Steffen Rendle. 2020. On Sampled Metrics for Item Recommendation. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'20)*. 1748–1757.
[18] Zhepeng Li, Xiao Fang, and Olivia Sheng. 2018. A Survey of Link Recommendation for Social Networks: Methods, Theoretical Foundations, and Future Research Directions. *ACM Transactions on Management Information Systems* 9, 1 (2018), 1–26.
[19] Qingyun Liu, Shiliang Tang, Xinyi Zhang, Xiaohan Zhao, Ben Y. Zhao, and Haitao Zheng. 2016. Network Growth and Link Prediction Through an Empirical Lens. In *Proceedings of the 2016 Internet Measurement Conference (IMC'16)*. 1–15.
[20] Zhen Liu, Qian-Ming Zhang, Linyuan Lu, and Tao Zhou. 2011. Link prediction in complex networks: a local naïve Bayes model. *Europhysics Letters* 96, 4 (2011), 48007.
[21] Farzan Masrour, Tyler Wilson, Heng Yan, Pang-Ning Tan, and Abdol-Hossein Esfahanian. 2020. Bursting the Filter Bubble: Fairness-Aware Network Link Prediction. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI'20)*. 841–848.
[22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14)*. 701–710.
[23] Chuxiong Sun, Hongming Gu, and Jie Hu. 2021. Scalable and Adaptive Graph Neural Networks with Self-Label-Enhanced Training. arXiv:2104.09376
[24] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-Scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web (WWW'15)*. 1067–1077.
[25] Antonela Tommasel, Juan Manuel Rodriguez, and Daniela Godoy. 2021. I Want to Break Free! Recommending Friends from Outside the Echo Chamber. In *Proceedings of the Fifteenth ACM Conference on Recommender Systems (RecSys'21)*. 23–33.
[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Proceedings of the Annual Conference on Neural Information Processing Systems 2017 (NIPS'17)*. 5998–6008.
[27] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR'18)*. 1–12.
[28] Zhibo Wang, Jilong Liao, Qing Cao, Hairong Qi, and Zhi Wang. 2015. Friendbook: A Semantic-Based Friend Recommendation System for Social Networks. *IEEE Transactions on Mobile Computing* 14, 3 (2015), 538–551.
[29] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*. 6861–6871.
[30] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *Proceedings of the 7th International Conference on Learning Representations (ICLR'19)*. 1–17.

A APPENDIX

A.1 Introduction of Base GNN Models

To verify the generalization ability of SSNet, we experiment with a variety of GNNs as base models and test if SSNet can improve their performance, including:

• **GraphSAGE** [11] is a scalable GNN by performing neighborhood sampling during the layer-wise message aggregation. We test two aggregation types in it: pooling aggregator and LSTM aggregator.
• **GIN** [30]. Considering that some GNN models such as GraphSAGE cannot distinguish certain simple graph structures, Keyulu et al. proposed the *Graph Isomorphism Network* (GIN) which is as powerful as the Weisfeiler-Lehman graph isomorphism test.
• **GAT** [27] leverages attention layers on graph-structured data, which allows for assigning different importance scores to different neighbor nodes.
• **SAGN** [23] replaces the concatenation operations in SIGN [9] with graph structure-aware attention operations, so that it can adaptively gather neighborhood information among different hops.
• **FAGCN** [4] uses a frequency adaptation graph convolutional network to adaptively combine the low-frequency and high-frequency signals on the graph.
• **LightGCN** [12] simplifies GCNs by removing the linear transformation and non-linear activation function.
• **PPRGo** [5]. Based on APPNP [16], PPRGo uses an efficient Personalized PageRank algorithm for information diffusion and only retains top-k most important neighbors for each root node.

A.2 Datasets

We use three real-world social network datasets for experiments. The **Xbox** dataset is constructed from Xbox gaming social network. The forming of an edge from u to v may be caused by multiple reasons, e.g., u and v are friends in the real world; or, u and v share similar gaming interests, they just like to play games together, without knowing the real identity of each other; or, v is a star player who attracts many fans. The other two datasets – **Pokec** and **LiveJournal** – are from the SNAP public datasets (https://snap.stanford.edu/data/index.html#socnets). Pokec is the most popular online social network in Slovakia. LiveJournal is a free online community where users can keep a blog, journal, or diary. Basic statistics of the three datasets are listed in Table A1. For all these three datasets, edges are directed, which means that a user can link to another user without reciprocation. To construct the validation set and test set, a small portion of edges are sampled and removed from each dataset, and the remaining edges are treated as the training set. To make the discussions general to all types of social networks, we do not involve side information such as node attributes feature in experiments. Still, they can be easily added to the input side of GNNs. Each node on the graph will be associated with a learnable embedding vector.

Table A1: Basic statistics of the three datasets

Dataset	#.nodes	#.edges	Avg #.degree
Pokec	1,632,803	27,560,308	16.88
LiveJournal	4,847,571	62,094,395	12.81
Xbox	3,000,000	80,194,576	26.73

A.3 Evaluation Metrics

For the candidate retrieval task, we use Hit-Rate (HR@100, HR@300) and NDCG@300 to evaluate models' performance, because in this task, the main objective is to propose a small set of candidates to downstream rankers, and the relative order of the top-k candidates is less important. For the friend ranking task, since we have a small size of candidate samples, we care more about the ranking order of positive samples against negative samples. So, we use *Normalized Discounted Cumulative Gain* (NDCG) and *Area Under the Curve* (AUC) as the benchmark metrics, which are commonly adopted in the literature.

A.4 Training Settings

For all the models, the embedding dimension is fixed to 64. We optimize the models with Adam [14] through the BPR [12] loss function. For GraphSAGE, GIN, GAT, and SAGN, their embeddings are initialized by the output embeddings of Node2vec, since we find them hard to train from scratch. The best setting of the GNN layer is summarized in Table A2. We empirically find that all GNNs need at most 2 layers (which means 2-hop neighborhood) in our scenario. For GraphSAGE, the neighbor sampling size is set to 20. For GIN, we use the sum aggregator and a 2-layer MLP for feature transformation. For GAT, we set the number of attention heads to 4 and take the mean of the outputs of different heads. For SAGN, we use the symmetrically normalized adjacency matrix, like in LightGCN, as the transition matrix, and we use 2-layer MLPs for all the MLPs in the model. For FAGCN, we omit the weight matrices and activation functions. The only parameters are embedding vectors and the weights in the self-gating mechanism. For LightGCN, we find it better not to stack the embeddings of different layers in our cases. For PPRGo, we use the top-32 neighbors, and the feature transformation MLP is omitted. We try the normalization method in neighborhood aggregation described in the original paper but

Table A2: Best number of stacking layers for GNNs

	SAGE	GIN	GAT	SAGN	FAGCN	LightGCN
Pokec	1	1	1	2	1	2
LiveJournal	2	1	2	2	1	2
Xbox	1	2	1	1	1	1

Table A3: Comparisons of SSNet's variants. The task is friend ranking.

	Pokec		LiveJournal		Xbox	
	AUC	NDCG	AUC	NDCG	AUC	NDCG
LightGCN						
Base	0.7752	0.3327	0.8110	0.3768	0.7569	0.3588
L2-norm	0.7548	0.3325	0.7771	0.3967	0.7212	0.3383
FFN	0.7789	0.3443	0.8116	0.4180	0.7586	0.3634
Additive	0.7741	0.3312	0.8095	0.3760	0.7508	0.3458
SSNet	**0.8169**	**0.3922**	**0.8501**	**0.4535**	**0.7808**	**0.4003**
PPRGo						
Base	0.8087	0.3840	0.8340	0.4174	0.7781	0.3966
L2-norm	0.7653	0.3511	0.7787	0.3860	0.7048	0.3394
FFN	0.7953	0.3666	0.8246	0.4039	0.7541	0.3740
Additive	0.8117	0.3876	0.8293	0.4082	0.7645	0.3819
SSNet	**0.8193**	**0.3988**	**0.8395**	**0.4229**	**0.7825**	**0.4056**

find that uniform weights perform better, so we adopt the latter. The number of MLP layers in SSNet and FFN is set to 2, with layer sizes being *[64, 32, 1]* for SSNet and *[64, 64, 64]* for FFN. We omit the scalar bias in the last layer of the SSNet's MLP. We use the $Tanh$ function as the non-linear activation for both of SSNet and FFN.

A.5 Additional Experimental Results

Table A2 lists the best number of stacking layers for GNNs. Table A3 reports the comparisons among vector transformation variants on the friend ranking task.