Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Improving performance and efficiency of Graph Neural Networks by injective aggregation

Wei Dong[a], Junsheng Wu[b], Xinwan Zhang[b], Zongwen Bai[c], Peng Wang[d,*], Marcin Woźniak[e,*]

[a] School of Computer Science and Engineering, Northwestern Polytechnical University, China
[b] School of Software, Northwestern Polytechnical University, China
[c] Shaanxi Key Laboratory of Intelligent Processing of Big Energy Data, School of Physics and Electronic Information, Yanan University, China
[d] School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia
[e] Faculty of Applied Mathematics, Silesian University of Technology, Kaszubsa 23, 44100 Gliwice, Poland

A R T I C L E I N F O

Article history:
Received 29 October 2021
Received in revised form 30 July 2022
Accepted 3 August 2022
Available online 13 August 2022

Keywords:
Graph Neural Networks
Aggregation function
Aggregation matrix
Injectivity
Traffic state prediction

A B S T R A C T

Aggregation functions are regarded as the multiplication between an aggregation matrix and node embeddings, based on which a full rank matrix can enhance representation capacity of Graph Neural Networks (GNNs). In this work, we fill this research gap based on the full rank aggregation matrix and its functional form, *i.e.*, the injective aggregation function, and state that injectivity is necessary to guarantee the rich representation capacity to GNNs. To this end, we conduct theoretical injectivity analysis for the typical feature aggregation methods and provide inspiring solutions on turning the non-injective aggregation functions into injective versions. Based on our injective aggregation functions, we create various GNN structures by combining the aggregation functions with the other ingredient of GNNs, node feature encoding, in different ways. Following these structures, we highlight that by using our injective aggregation function entirely as a pre-processing step before applying independent node feature learning, we can simultaneously achieve satisfactory performance and computational efficiency on the large-scale graph-based traffic data for traffic state prediction tasks. Through comprehensive experiments on standard node classification benchmarks and practical traffic state data (for Chengdu and Xi'an cities), we show that the representation capacity of GNNs can be improved by using our injective aggregation functions just by changing the model in simple operations.

1. Introduction

Graph Neural Networks (GNNs) [1,2] have revolutionized graph representation learning as a special type of neural network structure extended from the application of neural networks on speech recognition [3], natural language processing [4], and other applications in networking systems [5]. Given the promising features GNNs have shown thereon, recent years have witnessed a surge of interest in proposing GNN approaches to model graph-structured data. Some successful application areas of GNNs include, but are not limited to, social networks [6], recommender system [7–9], natural language processing [10], and computer vision [11,12].

Essentially, two operations underpinning the representation capacity of GNNs are neighborhood aggregation and node feature encoding. Neighborhood aggregation, *a.k.a.* message passing, updates the features of a node by aggregating the features of the neighboring nodes. Node feature encoding learns a parameterized mapping function to encode the node features. Based on these two operations, various GNN approaches are proposed, and normally these two operations are recursively iterated to model high-order contextual information from the neighborhood [13]. Motivated differently, these feature aggregation functions have different forms, and the performance of the GNNs can vary with applications. In terms of the node feature encoding, common options are single or Multiple Layer Perceptron (MLP). The difference between many GNN variants lies in the design of the feature aggregation functions [14]. These feature aggregation functions can be essentially formalized as the multiplication of the aggregation matrix and its independent variables, that is, the node feature embeddings. Hence, aggregation functions not only aggregate the node features from the neighborhood but also project the node feature embeddings into the space spanned

* Corresponding authors.
E-mail addresses: dw156@mail.nwpu.edu.cn (W. Dong), wujunsheng@nwpu.edu.cn (J. Wu), zhangxinwan@mail.nwpu.edu.cn (X. Zhang), ydbzw@yau.edu.cn (Z. Bai), pengw@uow.edu.au (P. Wang), marcin.wozniak@polsl.pl (M. Woźniak).

https://doi.org/10.1016/j.knosys.2022.109616
0950-7051/© 2022 Elsevier B.V. All rights reserved.

by its base vectors through multiplication operation. The result of the projection is generated by the linear combination of the base vectors. Generally, if the rank of the aggregation matrix is larger, the linear combination is more diverse, thus enriching the representation capacity for GNNs. Because the rank of a matrix is equal to the number of base vectors, a full rank aggregation matrix can enhance the representation capacity of GNNs maximally. Some latest works [9,15,16] focus on the using the co-embedding strategies, fusing these strategies and GNNs to improve the representation learning for various tasks.

However, theoretically and practically, the lack of a unified guidance on how to improve the proposed aggregation functions to the full rank version as well as a systematical comparison between the existing aggregation schemes with and without full rank not only makes it challenging to select the appropriate means to aggregate neighboring features but also hinders the design of new feature aggregation functions.

In this work, we fill the above research gap by providing a theoretical and practical guidance for analyzing and improving the effectiveness of aggregation functions to enhance the representation capacity of the GNNs. Our study is based on the design of the full rank aggregation matrix, which is equivalent to how to construct an injective feature aggregation function from a mathematical perspective. This implies injectivity of the aggregation function is a necessary property in guaranteeing the GNNs to rich its representation capacity. To this end, we conduct theoretical injectivity analysis for a set of popular feature aggregation functions in existing GNN models and provide inspiring solutions on how to turn the non-injective aggregation functions into injective versions. To verify the effectiveness of the injective feature aggregation functions in this work, we create various GNN structures by combining the aggregation functions with the node feature encoding in three different ways. One way follows the recursive aggregation and encoding schemes [13]. The other two ways completely decouple the feature aggregation and feature learning by using a two-stage pipeline: firstly aggregating the features and then applying the feature encoding for each node or firstly encoding the node features and then fusing the neighborhood information for node classification. Among these variants, we highlight that: to perform the tasks of traffic state prediction from large-scale graph-based traffic data, the combination of prepositive aggregation and postpositive encoding mode not only achieves satisfactory performance but also alleviates the computational burden on GPUs, as it can put the feature aggregation in CPUs as a pre-processing step. We conduct comprehensive experiments on eight node classification tasks and practical traffic state data (for Chengdu and Xi'an cities) [17,18]. The results demonstrate that the graph representation capacity can be improved by using our injective aggregation functions, which can be easily realized by just changing minor operation in the model. The contributions of this work can be summarized as follows:

- We present theoretical and practical guidance on how to derive injective feature aggregation functions from graph-structured data to enhance the representation capacity of GNNs. The injective aggregation functions in our work can be used as plug-ins to improve the effectiveness of the representation capacity in GNNs by simply changing simple operations.
- We provide a systematical comparison between some typical feature aggregation functions in the context of various GNN structures derived by combining feature aggregation and node encoding in different ways. To our knowledge, this is the first work which conducts a comprehensive experimental study on the feature aggregation functions in GNNs in this way.
- We highlight that the combination of prepositive aggregation and postpositive encoding mode has advantages in satisfactory performance and low dependency on the large-scale graph-based traffic data [17,18], verified by the experiments on the practical traffic state data for Chengdu and Xi'an cities.

The rest of the paper is organized as follows. Section 2 briefly presents some preliminary knowledge about GNNs and their applications in traffic domain. Section 3 elaborately describes how to design novel injective aggregation functions and turn the non-injective aggregation functions into injective versions. Section 4 creates various GNN structures by combining the aggregation functions with the node feature encoding in three different ways, verifying the effectiveness of the injective feature aggregation functions in this work. Section 5 highlights and analyzes the time and memory complexities of the combination of prepositive aggregation and postpositive encoding mode, and proceeds to utilize this combination scheme to perform the task of the large-scale traffic state prediction. Section 6 shows the competitive results of our models on the eight benchmark datasets and the efficient model performance on the large-scale traffic state prediction. Section 7 concludes the paper and proposes the future work.

2. Preliminary knowledge

Let $G = (\mathcal{V}, \mathbf{A})$ denotes an undirected graph with node feature vector $\mathbf{X}_v \in \mathbb{R}^{D^{(0)}}$ for $v \in \mathcal{V}$ and adjacency (typically symmetric) matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, where a_{ij} denotes the edge weight between nodes v_i and v_j. GNNs aim to learn node representation or graph representation based on such graph structure. Two elementary operations in GNNs are neighboring feature aggregation and node feature encoding. While the former updates the representation of a node by aggregating the feature vectors of the neighbors, the latter learns a parameterized mapping function to encode the node features. Different variants of GNNs vary with the forms of these two components. In this work, we focus mainly on the neighborhood aggregation part.

Graph Convolutional Networks (GCNs) [13] are a prevailing variant of GNN models for node classification tasks. It consists of a stack of learned first-order spectral filters followed by nonlinear activation function to learn node representations. Mathematically, GCNs can be written as:

$$\mathbf{H}^{(l)} = \sigma(\hat{\mathbf{A}}_{\text{sn}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}), \tag{1}$$

with $\hat{\mathbf{A}}_{\text{sn}}$ being the neighboring feature aggregation matrix, $\mathbf{W}^{(l)}$ being the feature encoding matrix in layer l, and $\sigma(\cdot)$ being a non-linear activation function. The feature aggregation matrix is a symmetrically normalized Laplacian matrix with the form:

$$\hat{\mathbf{A}}_{\text{sn}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}, \tag{2}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ denotes the adjacency matrix with self loops and $\tilde{\mathbf{D}} = \sum_j \tilde{\mathbf{A}}_{ij}$ denotes the diagonal degree matrix. $\mathbf{A}$ is the original adjacency matrix with element $a_{ij} = 1$ if nodes v_i and v_j are connected and otherwise $a_{ij} = 0$. In the aggregation step, GCN utilizes an aggregator $\hat{\mathbf{H}}^{(l-1)} = \hat{\mathbf{A}}_{\text{sn}}\mathbf{H}^{(l-1)}$ to gather the neighboring features into each node representation. In the following encoding step, an updater $\mathbf{H}^{(l)} = \hat{\mathbf{H}}^{(l-1)}\mathbf{W}^{(l)}$ is used to encode the representation for each node. These two operations are recursively iterated. After k iterations, the representation of a node captures k-hop contextual information from the neighborhood. This recursive process is adopted in most GNN models [13,19,20], which can be written mathematically as:

$$\mathcal{A}_{\text{rec}} = \mathcal{E}^{(L)} \circ \mathcal{F} \circ \cdots \circ \mathcal{E}^{(1)} \circ \mathcal{F}, \tag{3}$$

where $\mathcal{A}_{\text{rec}} : \mathbb{R}^{N\times D^{(0)}} \rightarrow \mathbb{R}^{N\times D^{(L)}}$ can be regarded as a recursive GNN with input node features $\mathbf{X} = \mathbf{H}^{(0)} \in \mathbb{R}^{N\times D^{(0)}}$. $\mathcal{F} : \mathbb{R}^{N\times D^{(l)}} \rightarrow \mathbb{R}^{N\times D^{(l)}}$ is an aggregation function to propagate neighboring features on the hidden representations $\mathbf{H}^{(l)}$ based on the aggregation matrix. $\mathcal{E}^{(l)} : \mathbb{R}^{N\times D^{(l)}} \rightarrow \mathbb{R}^{N\times D^{(l+1)}}$ is an encoding mapping to independently update each node representation in $\hat{\mathbf{H}}^{(l)}$.

Following GCNs, a subsequent of GNN variants have been proposed with various forms of feature aggregation matrices. In this paper, apart from the aggregation function in GCNs [13], we also explore some other popular feature aggregators, including GraphSAGE-Mean (Mean) [19], original PageRank (OPR) [21], and the personal PageRank (PPR) [21] for analysis. Mean aggregator [19] employs the mean of the neighboring features of each node as an aggregation scheme. Compared with GCNs [13], the benefit of Mean aggregator is that it can randomly sample a fixed number of neighbors per node for aggregation, thus reducing the computational cost when the neighborhood size is large [19]. The definition of the aggregation matrix in Mean aggregator is:

$$\hat{\mathbf{A}}_{\text{mean}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}, \tag{4}$$

where $\tilde{\mathbf{D}}$ and $\tilde{\mathbf{A}}$ have the same definitions as in Eq. (2).

Original PageRank [22] is an algorithm used to rank web pages. It updates the ranking of a page by weighted sum of its linked pages' rankings, where the weight for a neighboring page is defined as the reciprocal of its degree. This idea was extended in general graph-structured data for neighborhood propagation. This results in an OPR aggregation matrix [21], which is mathematically defined as:

$$\hat{\mathbf{A}}_{\text{opr}} = \tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1}. \tag{5}$$

The neighborhood aggregation in original GCN [13,19] is essentially Laplacian smoothing and stack of too many such aggregation layers can lead to over-smoothing. To enlarge the neighborhood size and overcome the over-smoothing issue, two aggregation models based on personalized PageRank's propagation scheme [22], termed personalized propagation of neural predictions (PPNP) and approximate personalized propagation of neural predictions (APPNP), are proposed [21].

PPNP utilizes the propagation scheme of $\pi_{\text{ppr}}(\vec{\boldsymbol{i}}_x) = (1-\alpha)\hat{\mathbf{A}}_{\text{sn}}\pi_{\text{ppr}}(\vec{\boldsymbol{i}}_x)+\alpha\vec{\boldsymbol{i}}_x$ to design its aggregation pipeline. The teleport vector $\boldsymbol{i}_x$ defining the root node x is a one-hot indicator vector. $\alpha \in (0, 1]$ is the teleport (restart) probability. By solving this propagation scheme, we get $\pi_{\text{ppr}}(\vec{\boldsymbol{i}}_x) = \alpha(\mathbf{I}_N - (1-\alpha)\hat{\mathbf{A}}_{\text{sn}})^{-1}\vec{\boldsymbol{i}}_x$. Substituting the unit matrix $\mathbf{I}_N$ for the indicator vector $\vec{\boldsymbol{i}}_x$, we obtain PPNP aggregation matrix $\hat{\mathbf{A}}_{\text{ppnp}} = \alpha(\mathbf{I}_N - (1-\alpha)\hat{\mathbf{A}}_{\text{sn}})^{-1}$. PPNP follows a predict-then-propagate pipeline, which can be expressed as:

$$\begin{aligned} &\mathbf{H}^{(l)} = \sigma(\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}), \quad l \in \mathbb{N} \\ &\mathbf{Z} = \mathit{softmax}(\alpha(\mathbf{I}_N - (1-\alpha)\hat{\mathbf{A}}_{\text{sn}})^{-1}\mathbf{H}^{(L-1)}). \end{aligned} \tag{6}$$

However, directly calculating the inverse operation in PPNP aggregator is expensive. To solve this issue, an approximate PPNP model (APPNP) is proposed [21], which uses topic-sensitive PageRank [23] instead of the fully personalized PageRank [22] as the aggregation scheme. Specifically, APPNP adopts the iterative computation manner from topic-sensitive PageRank, which avoids the calculation of matrix inversion. Mathematically, APPNP is written as:

$$\begin{aligned} &\mathbf{Z}^{(0)} = \mathbf{H}^{(l)} = \sigma(\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}), \quad l \in \mathbb{N} \\ &\mathbf{Z}^{(k+1)} = (1-\alpha)\hat{\mathbf{A}}_{\text{sn}}\mathbf{Z}^{(k)} + \alpha\mathbf{Z}^{(0)}, \\ &\mathbf{Z}^{(K)} = \mathit{softmax}((1-\alpha)\hat{\mathbf{A}}_{\text{sn}}\mathbf{Z}^{(K-1)} + \alpha\mathbf{Z}^{(0)}). \end{aligned} \tag{7}$$

After the (k)th aggregation step, the result of $\mathbf{Z}^{(k)}$ is rewritten as $\mathbf{Z}^{(k)} = \hat{\mathbf{A}}_{\text{appnp}}\mathbf{H}^{(0)}$, where $\hat{\mathbf{A}}_{\text{appnp}} = (1-\alpha)^k\hat{\mathbf{A}}_{\text{sn}}^k + \alpha\sum_{i=0}^{k-1}(1-\alpha)^i\hat{\mathbf{A}}_{\text{sn}}^i$ is APPNP aggregation matrix. When taking the limit $k \rightarrow +\infty$, $\mathbf{Z}^{(k)} \rightarrow \alpha(\mathbf{I}_N - (1-\alpha)\hat{\mathbf{A}}_{\text{sn}})^{-1}\mathbf{H}^{(0)}$, which is the PPNP's aggregation. Therefore, PPNP and APPNP aggregators are equivalent in theory when $k \rightarrow +\infty$.

In this section we show the underpinnings for prevailing aggregation functions of GNNs. Following this way, we will state how to turn the non-injective aggregation functions into injective versions (such as $\hat{\mathbf{A}}_{\text{sn}}$, $\hat{\mathbf{A}}_{\text{mean}}$, and $\hat{\mathbf{A}}_{\text{opr}}$), and how to prove the injectivity of PPNP and APPNP aggregators given specific factors.

3. New injective feature aggregation functions

The above-mentioned feature aggregators can be essentially formalized as the multiplication of the aggregation matrix and the node feature embeddings. Aggregators project the node feature embeddings into the space spanned by its base vectors through multiplication operation. Generally, if the rank of the aggregation matrix is larger, the linear combination is more diverse. Because the rank of matrix is equal to the number of base vectors, a full rank aggregation matrix (non-singular matrix) can enhance the representation capacity of GNNs maximally, *i.e.*, mathematically, the multiplication of the full rank aggregation matrix and the node feature embeddings can maintain the invariance of the rank of the node feature embeddings:

$$\text{rank}(\mathbf{PH}) = \min(\text{rank}(\mathbf{P}), \text{rank}(\mathbf{H})), \tag{8}$$

where $\mathbf{P}$ is any full rank aggregation matrix and $\mathbf{H}$ is the node feature embeddings. Following this way, we theoretically propose Lemma 1 (proved in Appendix A) to describe the difference between the representation capacity of the non-singular aggregations and singular ones by using the theory of Von Neumann Entropy [24].

Lemma 1. *Non-singular aggregations may have more representation capacity than singular ones.*

However, it leaves how to theoretically and practically improve the representation capacity of a general GNN model unanswered. To provide a necessary and important complement to GNNs, we resort to matrix theories to analyze the injectivity nature of the above aggregation functions and provide solutions on how to turn the non-injective aggregators into injective versions in order to enhance the representation capacity for the GNNs resulted.

Lemma 2 (*[25]*). *Let* $\mathbf{B}$ *be* $M \times N$ *matrix in general:*

- *If the matrix has full rank,* i.e., $\text{rank}(\mathbf{B}) = \min\{M, N\}$, *then the aggregator induced from* $\mathbf{B}$ *is: (1) injective if* $M \geq N = \text{rank}(\mathbf{B})$*; (2) surjective if* $N \geq M = \text{rank}(\mathbf{B})$*; (3) bijective if* $M = N = \text{rank}(\mathbf{B})$.
- *If the matrix* $\mathbf{B}$ *does not have full rank,* i.e., $\text{rank}(\mathbf{B}) < \min\{M, N\}$, *the aggregator derived from* $\mathbf{B}$ *is neither injective nor surjective.*

According to Lemma 2, we can associate the injectivity property of an aggregation function with the rank of its aggregation matrix. Because the aggregators studied in this work, *i.e.*, SN, Mean, and OPR, are symmetric matrices, changing the eigenvalues of these matrices can make them full rank based on Lemma 3:

Lemma 3 (*[25]*). *The number of non-zero eigenvalues of a symmetric matrix equals to its rank since symmetric matrices can be diagonalized.*

Furthermore, we add new Lemma 4 to facilitate our analysis on aggregation injectivity. The proof is presented in Appendix B by using diagonalization, which is the precondition for proving the injectivity of aggregators.

Lemma 4. *The three symmetric matrices, $\hat{\mathbf{A}}_{\rm sn}$, $\hat{\mathbf{A}}_{\rm mean}$, and $\hat{\mathbf{A}}_{\rm opr}$ in Eqs. (2), (4), (5), have the same eigenvalues $\lambda_1, \ldots, \lambda_N$ with $\lambda_n \le 1$ for $n \in [1, 2, \ldots, N]$.*

Not all aggregators can be designed to injective functions. For instance, Graph ATtention network (GAT) is a prevalent GNN model leveraging masked self-attentional layers to address the shortcomings of GNNs by heuristic design. Specifically, by stacking layers in which nodes are able to attend over their neighboring features, GAT implicitly enables specifying different weights to different nodes in a neighborhood, without requiring any kind of costly matrix operation or depending on knowing the graph structure upfront. The definition of GAT is:

$$\begin{aligned}\alpha_{i,j} &= \exp(\phi^*(\vec{\boldsymbol{a}}^{(l)\top}[\vec{\boldsymbol{h}}_i^{(l-1)} \parallel \vec{\boldsymbol{h}}_j^{(l-1)}])),\\ \hat{\mathbf{A}}^{(l)}_{\text{gat }[i,j]} &= \frac{\alpha_{i,j}}{\sum_{k\in\mathcal{N}_i}\alpha_{i,k}},\end{aligned} \tag{9}$$

where $\hat{\mathbf{A}}^{(l)}_{\rm gat}$ is the masked self-attentional aggregation matrix, the attention weights are $\vec{\boldsymbol{a}}^{(l)} \in \mathbb{R}^{2D^{(l)}}$, $\parallel$ is the concatenation operation, the representation of the node v_i is $\vec{\boldsymbol{h}}_i^{(l-1)} \in \mathbf{H}^{(l-1)}$, and the nonlinear activation function ϕ^* is LeakyReLU.

However, $\hat{\mathbf{A}}^{(l)}_{\rm gat}$ is asymmetric, thus, the conclusion that $\hat{\mathbf{A}}^{(l)}_{\rm gat}$ can be injective cannot be proved by diagonalization. This conclusion is ambiguous.

3.1. Building injective aggregation functions

The aggregators derived from $\hat{\mathbf{A}}_{\rm sn}$, $\hat{\mathbf{A}}_{\rm mean}$, and $\hat{\mathbf{A}}_{\rm opr}$ cannot be guaranteed to be injective because their eigenvalues can be 0, according to Lemma 4. To make them injective, based on Lemmas 2 and 3, one way is to change the eigenvalues of such symmetric matrices in order that all of their eigenvalues are non-zero, what makes these matrices full rank and the associated aggregation functions bijective (injective and surjective). To achieve this goal, we propose the following solution:

$$\hat{\mathbf{A}}_{\rm isn} = \beta\mathbf{I}_N + \hat{\mathbf{A}}^{2k}_{\rm sn}, \tag{10}$$

where the $\beta \in (0, 1]$ is a hyper-parameter and $\hat{\mathbf{A}}^{2k}_{\rm sn}$ denotes the matrix $\hat{\mathbf{A}}_{\rm sn}$ to the power of $2k$. The reason why we define the even power $2k$ is because of transforming the eigenvalue range of $\hat{\mathbf{A}}_{\rm sn}$ from $\lambda_n \le 1$ to $0 \le \lambda_n \le 1$ is conducive to designing injective functions. Intuitively, $\hat{\mathbf{A}}^{2k}_{\rm sn}$ can be understood as applying the original feature aggregation based on $\hat{\mathbf{A}}_{\rm sn}$ for an even number of times, which essentially models $2k$-hop contextual information. The eigenvalues of $\hat{\mathbf{A}}^{2k}_{\rm sn}$ fall in the range of $0 \le \lambda^{2k}_{\rm sn} \le 1$ with $k \in \mathbb{N}$. Adding $\beta\mathbf{I}_N$ on $\hat{\mathbf{A}}^{2k}_{\rm sn}$ ensures the eigenvalues of $\hat{\mathbf{A}}_{\rm isn}$ to fall in the range of $\beta \le \lambda_{\rm isn} \le 1+\beta$. Therefore, the symmetric matrix $\hat{\mathbf{A}}_{\rm isn}$ has full rank and the corresponding aggregator is bijective (injective and surjective) according to Lemmas 2 and 3. Similarly, injective Mean and OPR aggregators define their associated aggregation matrices as:

$$\hat{\mathbf{A}}_{\rm imean} = \beta\mathbf{I}_N + \hat{\mathbf{A}}^{2k}_{\rm mean}, \tag{11}$$

$$\hat{\mathbf{A}}_{\rm iopr} = \beta\mathbf{I}_N + \hat{\mathbf{A}}^{2k}_{\rm opr}. \tag{12}$$

Although our injective aggregators and the aggregators of residual GNNs [26,27] have similar forms, the injective aggregators use the residual connection to construct the injective function. In contrast, the aggregators of residual GNNs employ the residual connection to prevent the gradient vanishing or exploding problem in the node feature encoding. Moreover, Graph Isomorphism Network (GIN) [28] used the injective hash mapping inferred from Weisfeiler–Lehman graph isomorphism test (WL-Test) [29,30] as the aggregation function to capture the graph structure for enhancing the performance of graph classification tasks. The difference against GIN is that our injective aggregation function focuses on to enlarge the linear combination diversity of the base vectors in the aggregation matrix to enrich the representation capacity for GNNs. To summarize, we define Lemma 5 and present its detailed proof in Appendix C. We also compare the representation capacity gap between non-injective and injective aggregators in Table 1, implying that the original aggregation functions are close to be injective ones.

Table 1
The comparison of the representation capacity between non-injective and injective aggregators. The representation capacity is described by Von Neumann Entropy [24].

Aggregator	$\hat{\mathbf{A}}_{\rm sn}$	$\hat{\mathbf{A}}_{\rm isn}$
Representation Capacity	7.22998500122903	7.88019742347323
Aggregator	$\hat{\mathbf{A}}_{\rm mean}$	$\hat{\mathbf{A}}_{\rm imean}$
Representation Capacity	7.22998499824809	7.88019742125691
Aggregator	$\hat{\mathbf{A}}_{\rm opr}$	$\hat{\mathbf{A}}_{\rm iopr}$
Representation Capacity	7.22998499825765	7.88019742112355

Lemma 5. *The aggregators based on the three aggregation matrices $\hat{\mathbf{A}}_{\rm isn}$, $\hat{\mathbf{A}}_{\rm imean}$, and $\hat{\mathbf{A}}_{\rm iopr}$ defined in Eqs. (10), (11), (12) are injective, when $k \in \mathbb{N}$ and $\beta \in (0, 1]$.*

3.2. Injectivity analysis for personalized PageRank's propagation schemes

We claim that PPNP aggregator is injective intrinsically and APPNP aggregator is injective under some additional constraints. Formally, we propose Lemmas 6 and 7:

Lemma 6. *PPNP aggregator with its aggregation matrix $\hat{\mathbf{A}}_{\rm ppnp} = \alpha(\mathbf{I}_N - (1-\alpha)\hat{\mathbf{A}}_{\rm sn})^{-1}$ is injective.*

Lemma 7. *APPNP aggregator with its aggregation matrix $\hat{\mathbf{A}}_{\rm appnp} = (1-\alpha)^k\hat{\mathbf{A}}^k_{\rm sn} + \alpha\sum_{i=0}^{k-1}(1-\alpha)^i\hat{\mathbf{A}}^i_{\rm sn}$ is injective if the symmetrical matrix $\hat{\mathbf{A}}_{\rm sn}$ is irreducible, $k = 2u$ ($u \in \mathbb{N}$), and $\alpha \in (0, 1)$.*

The invertibility of the square matrix $\mathbf{I}_N - (1-\alpha)\hat{\mathbf{A}}_{\rm sn}$ in PPNP aggregation matrix has been proved in Klicpera et al. work [21], which implies that $\hat{\mathbf{A}}_{\rm ppnp}$ is full rank and PPNP aggregator is injective based on Lemma 2. Following this clue, we prove the injectivity of PPNP aggregator in Appendix D.

In Lemma 7, an irreducible aggregation matrix $\hat{\mathbf{A}}_{\rm sn}$ means that its corresponding graph is a connected graph, *i.e.*, a node in the graph must have a path to arrive at any other node in the graph. The irreducible property also indicates the eigenvalues of matrix $\hat{\mathbf{A}}_{\rm sn}$ to be $-1 \le \lambda_{\rm sn} \le 1$ based on Perron–Frobenius theorem [25]. If $k = 2u$ ($u \in \mathbb{N}$), we have $\lambda_{\rm appnp} = (1-\alpha)^{2u}\lambda^{2u}_{\rm sn} + \alpha\sum_{i=0}^{u-1}(1-\alpha)^{2i}\lambda^{2i}_{\rm sn}(1+(1-\alpha)\lambda_{\rm sn})$, which is proved in Appendix E. Hence, by virtue of Lemmas 2 and 3, $\lambda_{\rm appnp} > 0$ with $-1 \le \lambda_{\rm sn} \le 1$ ensures $\hat{\mathbf{A}}_{\rm appnp}$ to be full rank and APPNP aggregator to be bijective.

To ensure the injectivity of APPNP even with reducible matrix $\hat{\mathbf{A}}_{\rm sn}$, we further propose the injective Even-Power (EP) APPNP:

$$\hat{\mathbf{A}}_{\rm eappnp} = \beta\mathbf{I}_N + (1-\alpha)^k\hat{\mathbf{A}}^{2k}_{\rm sn} + \alpha\sum_{i=1}^{k-1}(1-\alpha)^i\hat{\mathbf{A}}^{2i}_{\rm sn}, \tag{13}$$

where $k \in \mathbb{N}$, β is a hyper-parameter as in Eq. (10), and α is a teleport probability as in Eq. (6). Similar to the injectivity proof of APPNP, we can have $\lambda_{\rm eappnp} = \beta + (1-\alpha)^k\lambda^{2k}_{\rm sn} + \alpha\sum_{i=1}^{k-1}(1-\alpha)^i\lambda^{2i}_{\rm sn}$. When $\lambda_{\rm sn} \le 1$, we have $\lambda_{\rm eappnp} > 0$, which indicates

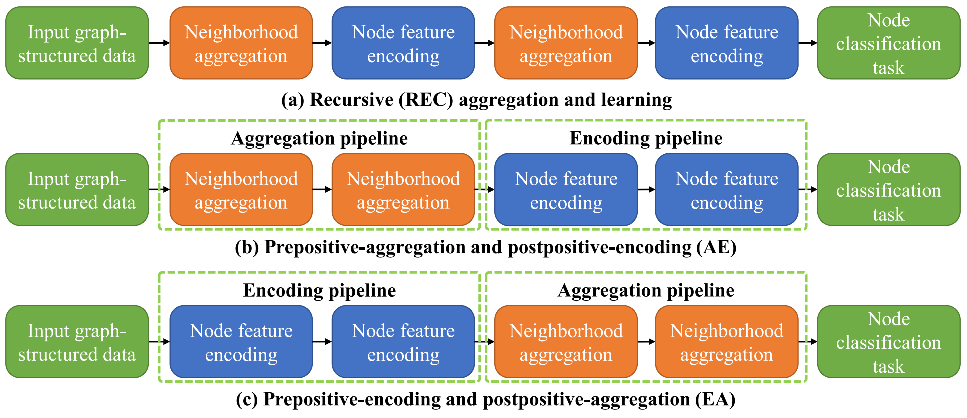

Fig. 1. Illustration of three GNN frameworks which are derived by combining aggregation functions and feature encoding functions through different ways: (a) recursive (REC) aggregation and learning; (b) prepositive-aggregation and postpositive-encoding (AE); (c) prepositive-encoding and postpositive-aggregation (EA).

the injectivity property of EP APPNP based on Lemmas 2 and 3. In addition, since the odd-power terms are removed in EP APPNP, it can save computation comparing to the original APPNP when performing feature aggregation. In summary, we propose Lemma 8, with the detailed proof in Appendix F:

Lemma 8. *The EP APPNP aggregator with the aggregation matrix* $\hat{\mathbf{A}}_{\mathrm{eappnp}} = \beta\mathbf{I}_N + (1-\alpha)^k\hat{\mathbf{A}}_{\mathrm{sn}}^{2k} + \alpha\sum_{i=1}^{k-1}(1-\alpha)^i\hat{\mathbf{A}}_{\mathrm{sn}}^{2i}$ *is injective when* $k \in \mathbb{N}$, $\alpha \in (0, 1]$, *and* $\beta \in (0, 1]$.

4. Composing GNNs by integrating injective aggregation functions and node encodings in different ways

With the injective aggregation functions in Section 3, we can compose various GNN models that enrich the representation capacity as much as possible by combining these aggregation functions with the node feature encoding functions. In this work, we choose non-linear MLP as node feature encoding function because, by the universal approximation theorem [31], the non-linear MLP is easier to fit injective functions than 1-layer perceptron [28].

To substantially verify the effectiveness of the proposed injective aggregators and provide a comprehensive test for typical aggregation functions, we go beyond the commonly adopted recursive framework and create different GNN structures by integrating the aggregators and feature encoding functions in three different ways: (1) recursive (REC) aggregation and encoding as in most GNNs [13,19,20,32]; (2) prepositive-aggregation and postpositive-encoding (AE) [33,34]; and (3) prepositive-encoding and postpositive-aggregation (EA) [21]. These three frameworks are shown in Fig. 1.

Under these three types of GNN frameworks, one common question we aim to answer is: *do the GNN models resulted from the injective aggregation functions observe advantage over the GNNs built from the non-injective counterparts?* To answer this question, we begin by defining the representation capacity of the aggregation function for GNN models, and theoretically comparing the capacity of the GNN models with injective or non-injective aggregation functions. The representation capacity of the aggregation function is defined as:

Definition 1. Let a mapping $\mathcal{A}$ be a GNN. The representation capacity of the aggregation mapping $\mathcal{F}$ for $\mathcal{A}$ is defined as:

$$C_{\mathcal{A}} \equiv \mathrm{rank}(\mathcal{F}), \tag{14}$$

where $\mathcal{F}$ could be any aggregation function represented by an aggregation matrix.

The representation capacity $C_{\mathcal{A}}$ can be effectively improved if $\mathcal{F}$ is an injective mapping. Before diving into the discussion of the representation capacity of the three different GNN frameworks, we need one additional lemma from discrete mathematics.

Lemma 9 ([35]). *If f_1 and f_2 are both injective functions, then the composite function $f_1 \circ f_2$ is injective.*

4.1. Composing GNNs under REC framework

REC framework expressed in Eq. (3) indicates aggregation and encoding steps are performed alternately. To enhance the representation capacity of existing recursive GNNs, we can substitute $\mathcal{F}$ with an injective aggregator to make $\mathcal{A}_{\mathrm{rec}}$ have more representation capacity. Hence, we propose:

Lemma 10. *Let the mapping $\mathcal{A}_{\mathrm{rec}}$ be a recursively trained GNN. We have:*

$$C_{\mathcal{A}_{\mathrm{rec}}|\mathcal{F}\text{ is injective}} \geq C_{\mathcal{A}_{\mathrm{rec}}}. \tag{15}$$

Relying on Lemma 2, each GNN layer in the recursive process, $\mathcal{E}^{(l)} \circ \mathcal{F}$, has more representation capacity if $\mathcal{F}$ is injective. The proof of Lemma 10 is shown in Appendix G.

4.2. Composing GNNs under AE framework

The aggregation and encoding are closely coupled in the REC framework, which results in time and memory complexity growing exponentially with the increasing number of model layers. This poses a challenge to the training of GNNs on large graphs [20]. To address this bottleneck, AE framework completely decouples the aggregation and encoding into two independent stages, as shown in Fig. 1. The advantage by doing this is that the aggregation can be fully removed from the training process as a pre-processing step, which can be performed in less expensive CPU and host memory once for all. Then in the feature encoding state, the encoding function applies to each node independently, which can be implemented in mini-batch mode without the need to loading the features of all the nodes into GPU memory. The AE framework is defined as:

$$\mathcal{A}_{\mathrm{ae}} = \mathcal{E}^{(L)} \circ \cdots \circ \mathcal{E}^{(1)} \circ \mathcal{F} \circ \cdots \circ \mathcal{F}. \tag{16}$$

The AE framework is relatively a new GNN framework. The design of the aggregation functions in the existing work [33,34] are still based on empirical intuition and heuristics. In this work, we use our theoretically-sound injective aggregators in this framework to improve the representation capacity of the GNNs. Hence, we propose:

Table 2
GNN structures based on various aggregation functions under three GNN frameworks.

Model	AE	EA	REC
SN	$\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{sn}}^2\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$	$\hat{\mathbf{A}}_{\mathrm{sn}}^2(\mathrm{ReLu}(\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)})$	$\hat{\mathbf{A}}_{\mathrm{sn}}^2\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{sn}}^2\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$
Injective SN	$\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{isn}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$	$\hat{\mathbf{A}}_{\mathrm{isn}}(\mathrm{ReLu}(\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)})$	$\hat{\mathbf{A}}_{\mathrm{isn}}\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{isn}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$
Mean	$\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{mean}}^2\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$	$\hat{\mathbf{A}}_{\mathrm{mean}}^2(\mathrm{ReLu}(\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)})$	$\hat{\mathbf{A}}_{\mathrm{mean}}^2\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{mean}}^2\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$
Injective Mean	$\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{imean}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$	$\hat{\mathbf{A}}_{\mathrm{imean}}(\mathrm{ReLu}(\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)})$	$\hat{\mathbf{A}}_{\mathrm{imean}}\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{imean}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$
OPR	$\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{opr}}^2\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$	$\hat{\mathbf{A}}_{\mathrm{opr}}^2(\mathrm{ReLu}(\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)})$	$\hat{\mathbf{A}}_{\mathrm{opr}}^2\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{opr}}^2\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$
Injective OPR	$\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{iopr}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$	$\hat{\mathbf{A}}_{\mathrm{iopr}}(\mathrm{ReLu}(\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)})$	$\hat{\mathbf{A}}_{\mathrm{iopr}}\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{iopr}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$
APPNP	$\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{appnp}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$	$\hat{\mathbf{A}}_{\mathrm{appnp}}(\mathrm{ReLu}(\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)})$	$\hat{\mathbf{A}}_{\mathrm{appnp}}\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{appnp}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$
EP APPNP	$\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{eappnp}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$	$\hat{\mathbf{A}}_{\mathrm{eappnp}}(\mathrm{ReLu}(\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)})$	$\hat{\mathbf{A}}_{\mathrm{eappnp}}\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{eappnp}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$
PPNP	$\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{ppnp}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$	$\hat{\mathbf{A}}_{\mathrm{ppnp}}(\mathrm{ReLu}(\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)})$	$\hat{\mathbf{A}}_{\mathrm{ppnp}}\mathrm{ReLu}(\hat{\mathbf{A}}_{\mathrm{ppnp}}\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}$

Lemma 11. *Let the mapping $\mathcal{A}_{\mathrm{ae}}$ be a prepositive-aggregation and postpositive-encoding framework. We have:*

$$C_{\mathcal{A}_{\mathrm{ae}}|\mathcal{F} \text{ is injective}} \geq C_{\mathcal{A}_{\mathrm{ae}}}. \tag{17}$$

The proof of Lemma 11 is provided in Appendix H.

4.3. Composing GNNs under EA framework

EA is another framework that decouples aggregation and encoding in GNN, which encodes the node features to obtain the node-wise predictions first and then neighboring predictions are propagated in the second aggregation stage to update the node predictions. In this sense, the GNNs under EA framework must be trained in an end-to-end fashion, which requires all the node features to be loaded into GPU memory. This framework is originally proposed in the work [21], which can be defined as:

$$\mathcal{A}_{\mathrm{ea}} = \mathcal{F} \circ \cdots \circ \mathcal{F} \circ \mathcal{E}^{(L)} \circ \cdots \circ \mathcal{E}^{(1)}. \tag{18}$$

Again, we replace the aggregation mapping $\mathcal{F}$ with our injective aggregators to create the injective GNN models under the EA models. We propose:

Lemma 12. *Let the mapping $\mathcal{A}_{\mathrm{ea}}$ be a pre-encoding and post-aggregation GNN framework. We have:*

$$C_{\mathcal{A}_{\mathrm{ea}}|\mathcal{F} \text{ s injective}} \geq C_{\mathcal{A}_{\mathrm{ea}}}. \tag{19}$$

The detailed proof is given in Appendix I.

In summary, Lemmas. 10, 11, and 12 prove the advantage of the GNN models based on the injective aggregation functions under three frameworks, which theoretically answer the question raised at the beginning of this section.

5. Complexity analysis for AE framework and performing it on the task of traffic state prediction

This section focuses on analyzing the complexity of AE framework and comparing it to state-of-the-art GNNs. Based on the complexity analysis, we utilize the advantage of AE framework performed on large-scale graph-based data to predict the traffic state.

5.1. Complexity analysis for GNNs under AE framework and the complexity comparison to state-of-the-art GNNs

We compare the time as well as memory complexity between REC and AE frameworks by employing the mini-batch mode. We define $D^{(1)} = \cdots = D^{(l)} = \cdots = D^{(L)} = D$ to simplify the description of time and memory complexities. The time and memory complexities of all state-of-the-art models in Section 6.1 are shown in Table 3.

In Eq. (1), the aggregator $\hat{\mathbf{H}}^{(l-1)} = \hat{\mathbf{A}}_{\mathrm{sn}}\mathbf{H}^{(l-1)}$ costs $O(\|\hat{\mathbf{A}}_{\mathrm{sn}}\|_0 D)$ and the updater $\mathbf{H}^{(l)} = \hat{\mathbf{H}}^{(l-1)}\mathbf{W}^{(l)}$ costs $O(ND^2)$. Given the time costs we have shown thereon, an L-layer GCN [13] consumes $O(L\|\hat{\mathbf{A}}_{\mathrm{sn}}\|_0 D + LND^2)$ time complexity by using full-batch mode, with requiring $O(LND)$ memory complexity. Both time and memory complexities of full-batch training GCN are proportional to N or L, which results in poor scalability.

Vanilla GCN is a variant of GCN [13] that uses mini-batch mode for GCN. It is defined as:

$$\begin{aligned} \omega_{ij} &= \frac{1}{(\delta_i+1)^{\frac{1}{2}}(\delta_j+1)^{\frac{1}{2}}}, \\ \vec{\boldsymbol{h}}_i^{(l)} &= \sigma((\mathbf{W}^{(l)})^T(\omega_{ii}\vec{\boldsymbol{h}}_i^{(l-1)} + \sum_{j\in\mathcal{N}_i}\omega_{ij}\vec{\boldsymbol{h}}_j^{(l-1)})), \end{aligned} \tag{20}$$

where $\vec{\boldsymbol{h}}_i^{(l)}$ is a feature vector in (l)th layer for node i. δ_i is node i's degree. $\mathcal{N}_i$ is the neighborhood of node i. $(\mathbf{W}^{(l)})^T$ is the transpose of a matrix $\mathbf{W}^{(l)}$. We suppose B to be the mini-batch samples and S_{NEI} to be the neighborhood size, then the training time complexity is $O(S_{\mathrm{NEI}}^L BD^2)$ and the memory complexity is $O(S_{\mathrm{NEI}}^L BD)$.

Several mini-batch variants of the vanilla GCN have been proposed to address the bottleneck of poor scalability. GraphSAGE [19] uses fixed-size sampling to reduce the computing range of the neighborhood in each layer, but it still faces the same bottleneck of vanilla GCN in essence. By defining S as the sampling size and $S < S_{\mathrm{NEI}}$, the training time complexity is $O(S^L BD^2)$ and the memory complexity is $O(S^L BD)$ for GraphSAGE.

FastGCN [36] leverages global importance sampling instead of fixed-size sampling to reduce the growth of the sampling size S from the exponential to linear level. However, FastGCN [36] must require the extra complexity for the global importance sampling, so its complexities are $> O(SLBD^2)$ for time and $> O(SLBD)$ for memory.

VRGCN [37] reduces the sample size in each layer by using the variance reduction. Its time complexity is $O(S_{\mathrm{VR}}^L BD^2)$ with the reduced sample size $S_{\mathrm{VR}} \leq S$, and its memory complexity is $O(LND)$.

Cluster-GCN [38] first utilizes the graph clustering to partition the whole graph to several subgraphs, employing the vanilla GCN [13] performed over each subgraph. Its performance depends on the chosen graph clustering methods heavily to difficultly ensure training stability. It also uses stochastic multiple partitions to enhance its performance to make its model structure sophisticated excessively.

Up to now, L-GCN [20] learns graph structured data by a layer-wise training manner. It performs aggregation step once for the $(l-1)$th node representations across each layer trained, feeding the node representations into a single layer perceptron in the encoding step by using mini-batch mode. Training (l)th layer of L-GCN by the layer-wise strategy detailed as:

$$\begin{aligned} &(\mathbf{W}^{(l)*}, \boldsymbol{\Theta}^*) = \\ &min_{\mathbf{W}^{(l)},\boldsymbol{\Theta}} \quad Loss(\sigma(\hat{\mathbf{A}}_{\mathrm{sn}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}), \boldsymbol{\Theta}, \mathbf{Y}). \end{aligned} \tag{21}$$

Table 3
In the training process, You et al. [20] summarized the time complexity of state-of-the-art GCNs with REC framework for feature aggregation, and the memory complexity of them for storing node representations. We add the time and memory complexities of AE framework for the training process to compare to these GCNs. For the weight storage, the memory costs are the same for all compared methods; hence, we ignore them in this table. L is the layer number, D is the feature dimension, N is the node number, S_{NEI} is the neighborhood size, B is the mini-batch size, S is the training sample size, S_{VR} is the reduced sample size, and N_{BAT} is the mini-batch number.

Complexity	GCN [13]	Vanilla GCN [13]	GraphSAGE [19]	FastGCN [36]	VRGCN [37]	L^2-GCN [20]	AE Framework
Training time	$O(L\|\hat{\mathbf{A}}_{\text{sn}}\|_0 D + LND^2)$	$O(S_{\text{NEI}}^L BD^2)$	$O(S^L BD^2)$	$> O(SLBD^2)$	$O(S_{\text{VR}}^L BD^2)$	$> O(L\frac{\|\hat{\mathbf{A}}_{\text{sn}}\|_0}{N_{\text{BAT}}} D + BD^2)$	$O(LBD^2)$
Training memory	$O(LND)$	$O(S_{\text{NEI}}^L BD)$	$O(S^L BD)$	$> O(SLBD)$	$O(LND)$	$> O(BD)$	$O(LBD)$

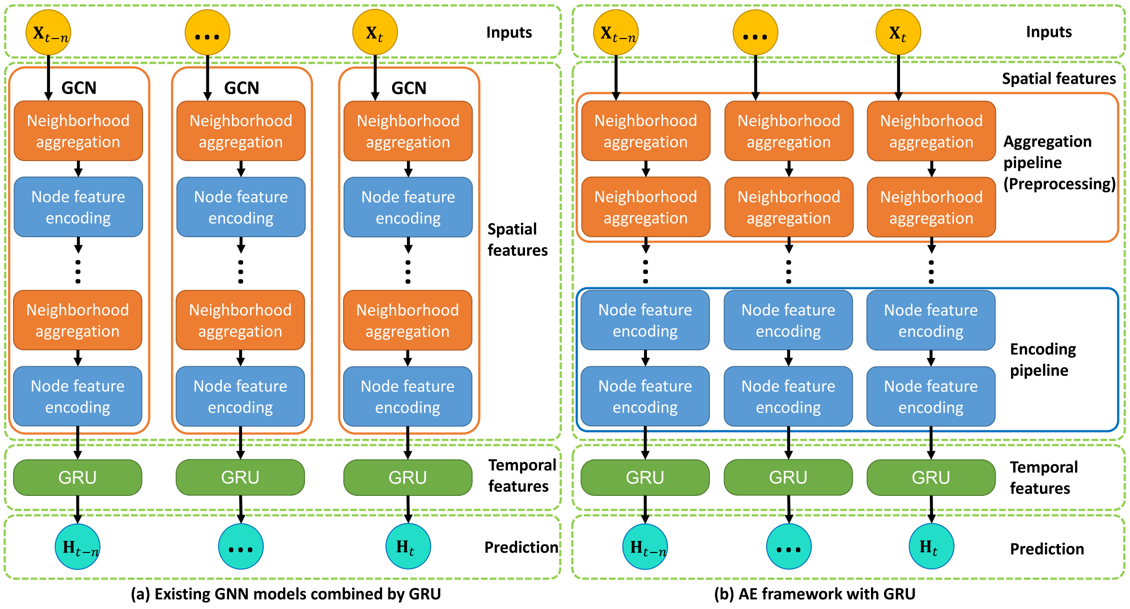

Fig. 2. The comparison between AE framework and existing GNN models associated with GRU.

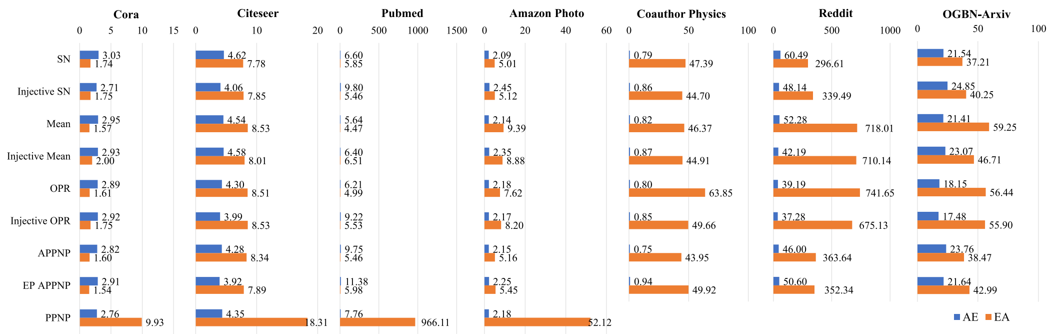

Fig. 3. The training time (seconds) comparison between AE and EA frameworks.

After finishing the (l)th layer training, L-GCN [20] saves the weight matrix $\mathbf{W}^{(l)*}$ between the current input layer and the hidden layer, and drops the weight matrices $\mathbf{\Theta}^*$ between this hidden layer and output layer unless $l = L$, then calculating the (l)th layer representations for next layer-wise training weights, *i.e.*, $(\mathbf{W}^{(l+1)*}, \mathbf{\Theta}^*)$. This process is repeated until all layers are trained. Moreover, You et al. [20] further propose L^2-GCN by using a learned RNN controller to decide when to stop in each layer's training [39] via policy-based Reinforcement Learning [40], leading to that the time complexity is $> O(L\frac{\|\hat{\mathbf{A}}_{\text{sn}}\|_0}{N_{\text{BAT}}} D + BD^2)$ and the memory complexity is $> O(BD)$. However, the layer-wise training strategy and the learned RNN controller are too sophisticated to guarantee the model's initial performance.

For the training time complexity, AE framework only conducts aggregation results once in the preprocessing stage, and encodes each node representation independently in the training process. Therefore, the time complexity of aggregation pipeline is $O(k\|\hat{\mathbf{A}}_{\text{sn}}\|_0 D)$ and the time complexity of encoding pipeline is $O(LBD^2)$, where k is the k-hop neighborhood. Because the aggregation pipeline is not involved in the training stage, the training time complexity of AE framework is $O(LBD^2)$, and training memory complexity for all L layers is $O(LBD)$.

L^2-GCN [20] runs the aggregating preprocessing stage before training each layer, thereby performing L times aggregation operation with L layers. By contrast, the aggregation pipeline of AE framework only performs neighborhood aggregation once for all during the whole preprocessing stage and training process. To compute and store the large symmetrically normalized matrix $\hat{\mathbf{A}}_{\text{sn}}$ and its k times aggregation with the feature matrix $\hat{\mathbf{A}}_{\text{sn}}^k \mathbf{H}^{(0)}$, we could take advantage of distributed computing and storage systems to finish these costly aggregating propagation before starting the training process. Moreover, unlike L^2-GCN [20]

Table 4
The comparison between the GNN models constructed from injective and non-injective aggregation functions under AE framework and the state-of-the-art methods, in terms of micro-f1, training time (seconds), and GPU memory usage (MBs). OOM denotes out of memory. The best results for each dataset are highlighted in (**bold**), and the best between injective aggregation function and its corresponding non-injective version is highlighted in **bold**.

Model	Cora			Citeseer			Pubmed			Amazon Photo		
	Micro-F1	Time	Memory	Micro-F1	Time	Memory	Micro-F1	Time	Memory	Micro-F1	Time	Memory
Vanilla Mini-Batch [13]	84.60 ± 0.15	24.13 s	459M	76.58 ± 0.28	46.21 s	584M	86.60 ± 0.10	89.52 s	926M	90.25 ± 0.22	56.92 s	706M
GraphSAGE-Mean [19]	85.10 ± 0.10	25.40 s	655M	77.82 ± 0.15	68.20 s	660M	87.15 ± 0.05	502.20 s	678M	90.26 ± 0.26	89.10 s	652M
FastGCN [36]	85.40 ± 0.10	8.21 s	659M	78.24 ± 0.30	26.06 s	696M	88.00 ± 0.10	41.05 s	862M	90.42 ± 0.23	26.30 s	853M
VRGCN [37]	85.30 ± 0.20	8.09 s	(**256M**)	78.06 ± 0.23	20.60 s	(**315M**)	87.10 ± 0.15	165.23 s	(**386M**)	90.68 ± 0.52	65.03 s	(**362M**)
Cluster-GCN [38]	84.50 ± 0.15	15.50 s	372M	76.93 ± 0.60	46.35 s	386M	86.85 ± 0.15	208.25 s	465M	89.68 ± 0.20	86.36 s	380M
GIN [28]	81.40 ± 0.22	8.28 s	680M	73.20 ± 0.25	22.20 s	690M	85.80 ± 0.22	113.86 s	884M	84.27 ± 0.36	89.42 s	864M
L-GCN [20]	84.60 ± 0.10	4.62 s	619M	77.06 ± 0.18	4.52 s	620M	87.40 ± 0.15	7.25 s	620M	89.82 ± 0.40	4.20 s	420M
L^2-GCN [20]	84.00 ± 0.15	3.56 s	619M	77.00 ± 0.12	4.08 s	618M	86.35 ± 0.10	6.21 s	634M	89.60 ± 0.26	3.64 s	422M
SN	87.64 ± 0.30	3.03 s	358M	78.52 ± 0.25	4.62 s	364M	88.94 ± 0.05	6.60 s	414M	90.60 ± 0.24	(**2.09 s**)	382M
Injective SN	**88.02 ± 0.50**	(**2.71 s**)	358M	(**79.89 ± 0.24**)	4.06 s	364M	**90.72 ± 0.16**	9.80 s	414M	**92.14 ± 0.47**	2.45 s	382M
Mean	87.50 ± 0.29	2.95 s	358M	78.28 ± 0.20	4.54 s	364M	87.32 ± 0.12	(**5.64 s**)	414M	89.54 ± 0.37	2.14 s	382M
Injective Mean	(**88.42 ± 0.26**)	2.93 s	358M	**79.45 ± 0.16**	4.58 s	364M	**89.26 ± 0.42**	6.40 s	414M	**91.26 ± 0.21**	2.35 s	382M
OPR	87.34 ± 0.27	2.89 s	358M	77.89 ± 0.16	4.30 s	364M	89.48 ± 0.25	6.21 s	414M	90.78 ± 0.05	2.18 s	382M
Injective OPR	**88.00 ± 0.32**	2.92 s	358M	**79.65 ± 0.22**	3.99 s	364M	**90.40 ± 0.34**	9.22 s	414M	**91.60 ± 0.54**	2.17 s	382M
APPNP	87.42 ± 0.16	2.82 s	358M	78.27 ± 0.26	4.28 s	364M	89.98 ± 0.18	9.75 s	414M	(**92.26 ± 0.29**)	2.15 s	382M
EP APPNP	**88.35 ± 0.27**	2.91 s	358M	**79.78 ± 0.25**	(**3.92 s**)	364M	(**91.09 ± 0.20**)	11.38 s	414M	91.83 ± 0.58	2.25 s	382M
PPNP	88.42 ± 0.21	2.76 s	358M	79.60 ± 0.63	4.35 s	364M	90.72 ± 0.39	7.76 s	414M	91.89 ± 0.29	2.18 s	382M

Model	Coauthor Physics			Reddit			OGBN-Arxiv			OGBN-Products		
	Micro-F1	Time	Memory	Micro-F1	Time	Memory	Micro-F1	Time	Memory	Micro-F1	Time	Memory
Vanilla Mini-Batch [13]	92.05 ± 0.28	386.51 s	2548M	92.96 ± 0.17	697.46 s	5621M	70.15 ± 0.14	602.18 s	1328M	OOM		
GraphSAGE-Mean [19]	92.20 ± 0.30	361.28 s	2208M	93.50 ± 0.10	1120.05 s	4352M	70.20 ± 0.10	925.16 s	2053M	OOM		
FastGCN [36]	92.58 ± 0.10	25.00 s	2588M	92.63 ± 0.16	922.18 s	4432M	69.45 ± 0.06	321.30 s	2015M	OOM		
VRGCN [37]	92.35 ± 0.10	265.23 s	1680M	94.35 ± 0.04	263.24 s	878M	69.95 ± 0.12	152.20 s	492M	75.10 ± 0.20	1256.58 s	890M
Cluster-GCN [38]	92.30 ± 0.18	320.25 s	1745M	94.66 ± 0.12	524.24 s	585M	68.82 ± 0.09	356.85 s	516M	74.50 ± 0.24	1754.20 s	733M
GIN [28]	86.85 ± 0.38	284.06 s	2028M	90.58 ± 0.35	696.28 s	4860M	64.38 ± 0.10	462.08 s	2100M	OOM		
L-GCN [20]	91.90 ± 0.25	6.25 s	720M	94.30 ± 0.16	48.21 s	622M	69.23 ± 0.14	29.54 s	486M	75.04 ± 0.10	82.14 s	518M
L^2-GCN [20]	91.85 ± 0.10	4.21 s	720M	94.15 ± 0.08	49.21 s	638M	69.90 ± 0.14	26.42 s	488M	75.20 ± 0.16	65.42 s	536M
SN	92.67 ± 0.21	0.79 s	(**608M**)	94.62 ± 0.06	60.49 s	(**414M**)	70.10 ± 0.10	21.54 s	(**382M**)	74.11 ± 0.09	(**26.87 s**)	(**382M**)
Injective SN	**93.69 ± 0.12**	0.86 s	(**608M**)	**95.43 ± 0.08**	48.14 s	(**414M**)	**71.28 ± 0.15**	24.85 s	(**382M**)	**75.55 ± 0.24**	33.60 s	(**382M**)
Mean	92.66 ± 0.05	0.82 s	(**608M**)	94.86 ± 0.05	52.28 s	(**414M**)	70.63 ± 0.12	21.41 s	(**382M**)	77.76 ± 0.12	44.59 s	(**382M**)
Injective Mean	**93.45 ± 0.06**	0.87 s	(**608M**)	(**96.45 ± 0.03**)	42.79 s	(**414M**)	(**71.58 ± 0.09**)	23.07 s	(**382M**)	(**78.80 ± 0.18**)	42.76 s	(**382M**)
OPR	92.84 ± 0.08	0.80 s	(**608M**)	91.46 ± 0.09	39.19 s	(**414M**)	68.36 ± 0.25	18.15 s	(**382M**)	67.82 ± 0.24	36.07 s	(**382M**)
Injective OPR	(**93.88 ± 0.03**)	0.85 s	(**608M**)	**92.84 ± 0.14**	(**37.28 s**)	(**414M**)	**68.95 ± 0.16**	(**17.48 s**)	(**382M**)	**70.65 ± 0.22**	38.24 s	(**382M**)
APPNP	92.40 ± 0.06	(**0.75 s**)	(**608M**)	94.22 ± 0.08	46.00 s	(**414M**)	69.82 ± 0.06	23.76 s	(**382M**)	74.28 ± 0.15	40.37 s	(**382M**)
EP APPNP	**93.68 ± 0.02**	0.94 s	(**608M**)	**94.95 ± 0.06**	50.60 s	(**414M**)	**69.99 ± 0.16**	21.64 s	(**382M**)	**74.96 ± 0.17**	42.04 s	(**382M**)
PPNP	OOM			OOM			OOM			OOM		

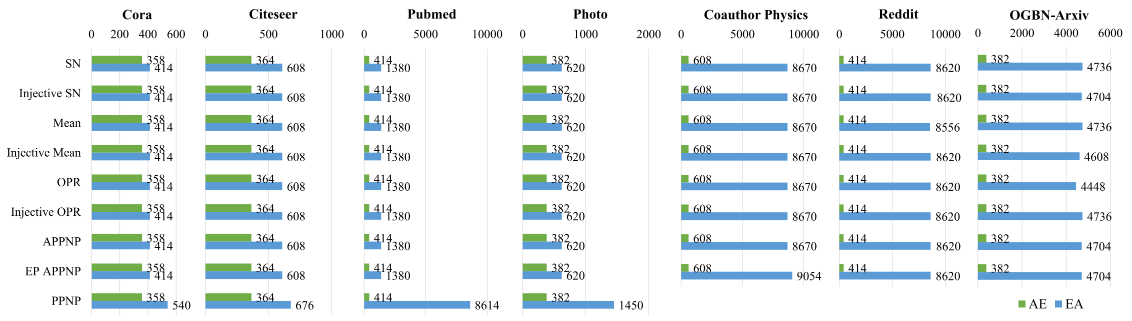

Fig. 4. Comparison of the GPU memory (MBs) usage (GPU memory usage during training) between AE and EA frameworks.

and Cluster-GCN [38], which require the complicatedly learned RNN controller and the sophisticated graph clustering algorithm to optimize their performance, respectively, AE framework is straightforward without any additional techniques or tricks.

5.2. Performing AE framework on the task of traffic state prediction

Based on the above analysis, as our AE framework alleviates the computational cost, performing it on the tremendous traffic graph-based data is efficient [17]. By decoupling two running-alternated and indispensable steps in GNNs, which are, respectively, neighborhood aggregation and node feature encoding functions, we arrange the aggregation part as a pre-processing pipeline before applying independent node feature learning as an encoding pipeline. The benefit of the decoupled structure is that we can deploy the intensive-consumed former on inexpensive computing and storage devices like Central Processing Units (CPUs) and memory, and the weights-learnable latter on costly but efficient parallel-computing resources (Graphics Processing Units, GPUs). To perform the tasks of traffic state prediction from graph-based traffic data implying spatial and temporal features [17], we elaborately design AE framework based on combining a temporal model, *i.e.*, Gate Recurrent Unit (GRU). In this regard, our AE framework can aggregate spatial features, and GRU can learn temporal features. The comparison between AE framework and existing GNN models with GRU are shown in Fig. 2.

6. Experiments

In this section, we firstly introduce the datasets we used for our experiments. Then we present some key implementation details of our methods. After that, we compare the GNNs derived from our injective aggregation functions to those built from non-injective aggregators as well as the state-of-the-art GNN models, in terms of both node classification performance and training efficiency.

Datasets: We conducted experiments on eight node classification datasets, which are increasingly larger. These datasets

Table 5
The CPU time consumption of the pre-processing step in AE framework.

Model	Cora	Citeseer	Pubmed	Amazon photo
SN	**(0.06 s)**	**0.20 s**	**0.19 s**	**(0.30 s)**
Injective SN	0.09 s	0.32 s	0.29 s	0.34s
Mean	**0.07 s**	**(0.19 s)**	**0.19 s**	**(0.30 s)**
Injective Mean	0.09 s	0.30 s	0.29 s	0.36s
OPR	**(0.06 s)**	**(0.19 s)**	**(0.18 s)**	**(0.30 s)**
Injective OPR	0.09 s	0.31 s	0.28 s	0.38s
APPNP	0.12 s	0.39 s	0.36 s	0.49s
EP APPNP	**0.10 s**	**0.32 s**	**0.29 s**	**0.37 s**
PPNP	8.14 s	14.76 s	490.81 s	221.36s
Model	Coauthor Physics	Reddit	OGBN-Arxiv	OGBN-Products
SN	**8.45 s**	**25.78 s**	**1.25 s**	**74.97 s**
Injective SN	11.18 s	27.02 s	1.36 s	102.29s
Mean	**(8.34 s)**	**(25.31 s)**	**(1.04 s)**	**(29.78 s)**
Injective Mean	11.00 s	26.88 s	1.27 s	56.03s
OPR	**8.35 s**	**25.64 s**	**1.06 s**	**71.60 s**
Injective OPR	11.33 s	26.68 s	1.30 s	97.98s
APPNP	14.68 s	38.95 s	1.75 s	123.38s
EP APPNP	**11.32 s**	**27.86 s**	**1.36 s**	**102.66 s**
PPNP	OOM	OOM	OOM	OOM

include two small-scale datasets: Cora [13] and Citeseer [13]; three medium-scale datasets: Pubmed [13], Amazon Photo [43], and Coauthor Physics [43]; and three large-scale datasets: Reddit [19], OGBN-Arxiv [44], and OGBN-Products [44]. Among these datasets, the graph representation of Citeseer is not connected and its corresponding adjacency matrix is reducible. Besides the eight datasets, we also introduce two large-scale graph-based traffic data: the traffic networks for Chengdu and Xi'an cities. The detailed dataset statistics are summarized in Appendix J.

Implementation details: We present the detailed network structures of our GNN models in Table 2, where *SN, Mean*, and *OPR* indicate the aggregation functions used, are the original non-injective versions introduced in Section 2 and *injective x* indicates the injective version of aggregation function x is used. To ensure the same 2-order neighborhood size in AE and EA frameworks, we set $k = 2$ in APPNP and $k = 1$ in the other models. For REC in Table 2, all 2-layer models have 4-order neighborhood since each of our injective aggregation function has minimum order of 2. The teleport probability α is set to be 0.2 for PPNP, APPNP, and EP APPNP; the bias β is 0.2 for injective SN, injective Mean, injective OPR, and EP APPNP under the AE framework; and β is set to be 0.1 for these injective models under REC and EA frameworks. We use Adam optimizer with the learning rate of 0.001 for AE and 0.0045 for EA and REC. The weight decay is set to be 0.005. A dropout with ratio 0.2 is used. We train the models for 1000 epochs, with the epoch number for early stopping set to be 10 for Cora and Citeseer, 30 for Amazon Photo, and 5 for the other datasets. We implement our models by Tensorflow 1.5. All of the experiments are performed on a machine with Intel 8-core i7-8700K CPU (3.70 GHz), 64 GB CPU memory, and one GeForce RTX 2080Ti card (11 GB GPU's memory). For each dataset, we run a model five times and the mean and standard variation of the micro-f1 score, and the average training time and GPU consumption are used as evaluation metrics. For data normalization, we employ the row normalized approach [13] for Cora, Citeseer, Pubmed, Amazon Photo, and Coauthor Physics, and use the standard scalar method [43] for the others.

6.1. The experiments under AE framework

Considering the GNN models derived from our injective aggregation functions under AE framework, they can simultaneously achieve satisfactory classification performance and computational efficiency, we compare such injective AE models with eight state-of-the-art GNN models, including Vanilla Mini-Batch [13], GraphSAGE-Mean [19], FastGCN [36], VRGCN [37], Cluster-GCN [38], GIN [28], L-GCN [20], and L^2-GCN [20]. The results are shown in Table 4. The numbers reported for the comparing methods are obtained as the best performance by running the code publicly released from the original papers five times. For fair comparisons, all the comparing methods use the same settings, including the same batch size, the same feature dimensionality and the same number of model layers. Concretely, we set the batch size to be 256 for Cora and Citeseer, and 1024 for the other datasets; The channel dimensionality for the hidden layers are set to be 16 for Cora and Citeseer, and 512 for the remaining datasets. All the models have 2 layers.

From the table we can see, the GNN models based on our injective aggregation functions outperform the state-of-the-art models in terms of all the three metrics, *i.e.*, micro-f1, training time, and GPU memory consumption. The reason for the high training and memory efficiency of our models lies in that we remove the neighborhood aggregation from the training pipeline as a pre-processing step. The CPU time consumption of the pre-processing step in AE framework is shown in Table 5. The independent node-wise encoding in the training pipeline can be conveniently implemented using mini-batch mode, which alleviates the GPU burden. Although the comparing methods also adopt mini-batch training, they employ the recursive training structure, which results in expensive computation.

We also compare our GNN models with those built from the original non-injective aggregation functions. As can be seen, the injective versions obtain superior performance over the non-injective baselines on almost all the datasets. This observation empirically verifies the effectiveness of our injective aggregation functions, formally claimed in Lemma 11. PPNP is a powerful aggregation scheme, which results in stable performance. But since it requires the calculation of matrix inverse, it leads to enormous memory consumption especially on large graph datasets. This explains why one can observe "out of memory" issues.

6.2. The experiments under EA framework

In this section, we evaluate the performance of GNNs built from our injective aggregation functions under EA framework. Firstly, we compare our injective models to some existing GNN models in Table 6. Following [43], we choose MLP and another two state-of-the-art models, recursively full-batch trained GAT [41] and MoNet [42] for comparison, where MLP means the models do not have the neighboring feature aggregation operation. The experimental setup is the same as that under the AE framework except that the learning rate under EA framework is set to be 0.0045. From the table we can see that our injective models outperform such comparing methods on all the datasets.

Note that PPNP and APPNP in their original work [21] are under EA framework. As analyzed in Lemmas 6 and 7, PPNP aggregator is injective and APPNP aggregator is injective when the power k is even and the graph is not reducible. One special case for APPNP is the Citeseer dataset, which is not connected and thus cannot guarantee the injectivity of the APPNP aggregation function. The proposed EP APPNP performs on par with or sometimes better than these two PPR variants but with higher aggregation efficiency as it removes the odd-power terms in the APPNP formulation.

In the comparison between the injective and non-injective models under EA frameworks, the injective models again observe better performance on most of the datasets. The results are consistent with Lemma 12.

Furthermore, we compare AE and EA frameworks in terms of GPU time in Fig. 3 and GPU memory consumption in Fig. 4

Table 6
The comparison between the GNN models constructed from injective and non-injective aggregation functions under EA framework and the state-of-the-art methods. The results on OGBN-Products are removed from the table due to OOM.

Model	Cora	Citeseer	Pubmed	Amazon Photo
MLP	74.46 ± 4.77	74.94 ± 0.54	86.46 ± 0.21	81.72 ± 0.57
GAT [41]	87.02 ± 0.12	78.38 ± 0.10	87.75 ± 0.22	85.78 ± 3.08
MoNet [42]	86.95 ± 0.15	77.89 ± 0.25	88.06 ± 0.08	91.00 ± 0.28
SN	87.20 ± 0.01	79.02 ± 0.32	88.20 ± 0.12	92.09 ± 0.18
Injective SN	**87.95 ± 0.22**	**79.78 ± 0.12**	**89.30 ± 0.16**	**92.85 ± 0.12**
Mean	87.06 ± 0.18	78.96 ± 0.21	86.94 ± 0.16	91.99 ± 0.15
Injective Mean	**87.84 ± 0.03**	**79.58 ± 0.18**	**87.86 ± 0.23**	**92.68 ± 0.25**
OPR	87.12 ± 0.23	79.16 ± 0.33	88.02 ± 0.40	91.76 ± 0.30
Injective OPR	**87.68 ± 0.10**	**79.88 ± 0.20**	**88.84 ± 0.25**	**(92.87 ± 0.20)**
APPNP [21]	87.15 ± 0.12	80.04 ± 0.29	**(89.78 ± 0.30)**	92.49 ± 0.36
EP APPNP	**87.72 ± 0.10**	**(80.24 ± 0.22)**	89.24 ± 0.33	**92.87 ± 0.19**
PPNP [21]	**(88.26 ± 0.37)**	79.76 ± 0.21	89.74 ± 0.39	92.25 ± 0.12
Model	Coauthor Physics	Reddit	OGBN-Arxiv	
MLP	87.89 ± 0.55	73.87 ± 0.09	55.15 ± 0.24	
GAT [41]	92.55 ± 0.38	OOM	OOM	
MoNet [42]	92.62 ± 0.10	94.60 ± 0.08	69.52 ± 0.24	
SN	92.85 ± 0.12	95.17 ± 0.06	70.06 ± 0.18	
Injective SN	**(93.72 ± 0.09)**	**(96.07 ± 0.07)**	**(70.85 ± 0.32)**	
Mean	92.04 ± 0.08	94.92 ± 0.03	70.14 ± 0.22	
Injective Mean	**93.67 ± 0.03**	**95.99 ± 0.06**	**70.73 ± 0.12**	
OPR	92.71 ± 0.08	94.22 ± 0.06	69.13 ± 0.30	
Injective OPR	**93.56 ± 0.03**	**95.02 ± 0.08**	**69.95 ± 0.15**	
APPNP [21]	93.34 ± 0.05	95.58 ± 0.11	69.13 ± 0.34	
EP APPNP	**93.56 ± 0.04**	**96.06 ± 0.05**	**70.39 ± 0.29**	
PPNP [21]	OOM	OOM	OOM	

Table 7
The comparison between GNN models constructed from non-injective and injective aggregation functions under REC framework. The results on Coauthor Physics, Reddit, and OGBN-Products are removed from this table due to OOM.

Model	Cora	Citeseer	Pubmed	Amazon Photo	OGBN-Arxiv
SN	87.05 ± 0.18	78.38 ± 0.44	86.30 ± 0.18	89.02 ± 0.28	70.46 ± 0.20
Injective SN	**87.58 ± 0.32**	**78.54 ± 0.14**	**87.06 ± 0.15**	**89.78 ± 0.38**	**70.74 ± 0.26**
Mean	87.48 ± 0.19	77.70 ± 0.70	85.24 ± 0.22	89.63 ± 0.35	70.41 ± 0.43
Injective Mean	**87.64 ± 0.29**	**(78.62 ± 0.44)**	**85.68 ± 0.07**	**(90.42 ± 0.10)**	**70.50 ± 0.21**
OPR	87.32 ± 0.23	78.18 ± 0.27	86.46 ± 0.10	89.21 ± 0.38	69.79 ± 0.10
Injective OPR	**87.58 ± 0.20**	**78.42 ± 0.64**	**86.94 ± 0.24**	**89.48 ± 0.51**	**69.94 ± 0.15**
APPNP	87.28 ± 0.13	78.10 ± 1.09	**(88.06 ± 0.14)**	**90.34 ± 0.37**	70.09 ± 0.10
EP APPNP	**87.42 ± 0.34**	**78.24 ± 0.44**	86.90 ± 0.55	89.70 ± 0.27	**(70.79 ± 0.10)**
PPNP	**(87.86 ± 0.44)**	77.98 ± 0.48	86.80 ± 0.14	89.35 ± 0.84	OOM

Table 8
The comparison between the GNN models constructed from injective and non-injective aggregation functions under AE framework, in terms of micro-f1, training time (seconds), and GPU memory usage (MBs). OOM denotes out of memory. The best results for each dataset are highlighted in **(bold)**, and the best between injective aggregation function and its corresponding non-injective version is highlighted in **bold**.

Model	Chengdu			Xi'an		
	Micro-F1	Time	Memory	Micro-F1	Time	Memory
SN	92.23 ± 0.12	68.67 s	426M	90.06 ± 0.22	55.52 s	414M
Injective SN	**93.28 ± 0.20**	62.56 s	426M	**92.58 ± 0.18**	58.20 s	414M
Mean	91.45 ± 0.10	60.62 s	426M	91.68 ± 0.23	56.02 s	414M
Injective Mean	**93.58 ± 0.14**	**(60.35 s)**	426M	**92.05 ± 0.15**	54.25 s	414M
OPR	92.25 ± 0.18	65.38 s	426M	91.14 ± 0.52	**(51.45 s)**	414M
Injective OPR	**(93.65 ± 0.25)**	62.65 s	426M	**(92.84 ± 0.14)**	55.28 s	414M
APPNP	93.38 ± 0.25	68.56 s	426M	92.15 ± 0.30	60.28 s	414M
EP APPNP	**93.40 ± 0.18**	65.39 s	426M	**92.82 ± 0.26**	62.65 s	414M
PPNP	OOM			OOM		

on all datasets except OGBN-Products. As EA framework needs to perform the feature encoding and feature aggregation on the whole graph online, its training follows the full-batch mode, which loads all the node features into the GPU memory. This

Table 9
The CPU time consumption of the pre-processing step in AE framework for the Chengdu and Xi'an datasets.

Model	Chengdu	Xi'an
SN	235.21 s	205.68 s
Injective SN	**(218.08 s)**	211.07 s
Mean	256.12 s	218.53 s
Injective Mean	268.05 s	225.40 s
OPR	276.10 s	**(196.62 s)**
Injective OPR	269.08 s	206.85 s
APPNP	251.39 s	268.69 s
EP APPNP	263.12 s	263.38 s
PPNP	OOM	OOM

leads to more expensive computation and memory consumption comparing to the models under AE framework, where the whole aggregation pipeline is removed from the training process.

6.3. *The experiments under REC framework*

The last set of experiments in Table 7 are conducted to verify Lemma 10, *i.e.*, injective REC models have better representation capacity than its non-injective versions and are thus expected to obtain higher classification accuracy. From the table, we can see the GNN models constructed from the injective *SN, Mean, OPR* outperform the models from the non-injective aggregators on all the datasets. The three PPR variants, *i.e.*, PPNP, APPNP, EP APPNP, observe comparable performance as the aggregation functions in all these three cases are injective, where one special case is Citeseer which cannot guarantee the injectivity of APPNP.

6.4. *The experiments of AE framework on the task of traffic state prediction*

Besides the experiments mentioned above, we also employ AE framework to predict the traffic state based on the large-scale graph-based traffic data. Table 8 shows the comparison between the GNN models constructed from injective and non-injective aggregation functions under AE framework. Note that the state-of-the-art GNN models mentioned above are Out-Of-Memory (OOM) on the two large-scale graph-based traffic data. This experiment implies the outstanding efficiency of AE framework on the task of traffic state prediction. Table 9 indicates that: on the two traffic datasets, the CPU time consumption of the pre-processing step in AE framework is acceptable and competitive.

7. Conclusions

In this work, we improved the representation capacity of GNN models by proposing injective aggregation functions. To this end, we analyzed the injectivity for the typical aggregation functions and proposed solutions on turning the non-injective functions into injective versions. Our solutions can not only improve the representation capacity of the GNNs but also provide theoretical and practical guidance on the design of new feature aggregation functions in GNNs. We have evaluated the effectiveness of the injective functions under various GNN frameworks. The experimental results demonstrated the advantages of our proposals, especially showing the state-of-the-art results of AE framework on the task of traffic state prediction.

In future work, we will explore more injective aggregation functions theoretically and practically, generalizing these functions performed on other tasks like graph classification and link prediction. We also hope to extend the applicability of such aggregation functions to other large-scale graph-structured applications, such as social networks and recommendation networks, widely verifying the effectiveness and efficiency of our proposals.

CRediT authorship contribution statement

Wei Dong: Conceptualization, Software, Investigation, Writing – original draft. **Junsheng Wu:** Conceptualization, Software, Investigation, Writing – original draft. **Xinwan Zhang:** Conceptualization, Software, Investigation, Writing – original draft. **Zongwen Bai:** Conceptualization, Software, Validation, Investigation, Visualization, Writing – original draft. **Peng Wang:** Conceptualization, Methodology, Validation, Investigation, Writing – original draft, Supervision. **Marcin Woźniak:** Conceptualization, Methodology, Validation, Investigation, Writing – original draft, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

Authors would like to acknowledge contribution to this research from the Rector of the Silesian University of Technology, Gliwice, Poland under pro-quality Grant No. 09/010/RGJ22/0068. Our work has also been supported by these following projects: Natural Science Basic Research Program of Shaanxi (Program No. 2021JLM-16) and National Natural Science Foundation of China (Grant No. 61872284).

Appendix A. Proof of Lemma 1

Proof. Given an aggregation matrix $\mathbf{P} \in \mathbb{R}^{N\times N}$, the Von Neumann Entropy [24] defined thereon is described as:

$$H(\mathbf{P}) = \sum_{n=1}^{N} -\frac{\lambda_n}{\sum_{n=1}^{N}\lambda_n}\log\frac{\lambda_n}{\sum_{n=1}^{N}\lambda_n}, \tag{A.1}$$

where λ_n is the eigenvalue derived from the aggregation matrix $\mathbf{P}$. The work [24] states that large $H(\mathbf{P})$ implies more information capacity, enhancing its corresponding representation ability in the basis space spanned by the aggregation matrix $\mathbf{P}$. Generally, non-singular aggregation matrix has non-zero eigenvalues, implying its larger range of the probability distribution $\frac{\lambda_n}{\sum_{n=1}^{N}\lambda_n}$ than that of singular ones. In term of Eq. (A.1) the larger range $\frac{\lambda_n}{\sum_{n=1}^{N}\lambda_n}$ indicates higher $H(\mathbf{P})$, thus non-singular aggregations may have more representation capacity than singular ones. □

Appendix B. Proof of Lemma 4

Proof. Mean aggregation matrix $\hat{\mathbf{A}}_{\text{mean}}$ defined in Eq. (4) is a right stochastic square matrix with each row summing to 1. According to Gershgorin circle theorem [25], any eigenvalue λ_{mean} derived from this matrix is $\lambda_{\text{mean}} \le 1$.

Let λ_{sn} be any eigenvalue and $\vec{\boldsymbol{v}}_{\text{sn}}$ be the associated eigenvector for the symmetrically normalized Laplacian matrix $\hat{\mathbf{A}}_{\text{sn}}$ described in Eq. (2). There is:

$$\hat{\mathbf{A}}_{\text{sn}}\vec{\boldsymbol{v}}_{\text{sn}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\vec{\boldsymbol{v}}_{\text{sn}} = \lambda_{\text{sn}}\vec{\boldsymbol{v}}_{\text{sn}} \tag{B.1}$$

By multiplying Eq. (B.1) with the diagonal degree matrix $\tilde{\mathbf{D}}^{-\frac{1}{2}}$ from left, Eq. (B.1) can be rewritten as:

$$\begin{aligned}\tilde{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}_{\text{sn}}\vec{\boldsymbol{v}}_{\text{sn}} &= \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\vec{\boldsymbol{v}}_{\text{sn}},\\ &= \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\vec{\boldsymbol{v}}_{\text{sn}} = \hat{\mathbf{A}}_{\text{mean}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\vec{\boldsymbol{v}}_{\text{sn}}.\end{aligned} \tag{B.2}$$

Due to Eq. (B.1), there is:

$$\tilde{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}_{\text{sn}}\vec{\boldsymbol{v}}_{\text{sn}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\lambda_{\text{sn}}\vec{\boldsymbol{v}}_{\text{sn}} = \lambda_{\text{sn}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\vec{\boldsymbol{v}}_{\text{sn}}. \tag{B.3}$$

Let combining Eqs. (B.2) and (B.3), we obtain:

$$\begin{aligned}\hat{\mathbf{A}}_{\text{mean}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\vec{\boldsymbol{v}}_{\text{sn}} &= \lambda_{\text{sn}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\vec{\boldsymbol{v}}_{\text{sn}},\\ &= \lambda_{\text{mean}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\vec{\boldsymbol{v}}_{\text{sn}} = \lambda_{\text{mean}}\vec{\boldsymbol{v}}_{\text{mean}}.\end{aligned} \tag{B.4}$$

By virtue of Eq. (B.4), $\lambda_{\text{sn}} = \lambda_{\text{mean}}$ and each eigenvector of the symmetrically normalized Laplacian matrix $\hat{\mathbf{A}}_{\text{sn}}$ is the corresponding eigenvector of the right stochastic matrix $\hat{\mathbf{A}}_{\text{mean}}$ scaled by $\tilde{\mathbf{D}}^{-\frac{1}{2}}$.

Relying on the deducing process we presented above and multiplying the following equation:

$$\begin{aligned}\hat{\mathbf{A}}_{\text{opr}}\vec{\boldsymbol{v}}_{\text{opr}} &= \tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1}\vec{\boldsymbol{v}}_{\text{opr}},\\ &= \lambda_{\text{opr}}\vec{\boldsymbol{v}}_{\text{opr}},\end{aligned} \tag{B.5}$$

with $\tilde{\mathbf{D}}^{-1}$ from left, there is:

$$\begin{aligned}\tilde{\mathbf{D}}^{-1}\hat{\mathbf{A}}_{\text{opr}}\vec{\boldsymbol{v}}_{\text{opr}} &= \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1}\vec{\boldsymbol{v}}_{\text{opr}},\\ &= \hat{\mathbf{A}}_{\text{mean}}\tilde{\mathbf{D}}^{-1}\vec{\boldsymbol{v}}_{\text{opr}}.\end{aligned} \tag{B.6}$$

As Eq. (B.5), we obtain:

$$\begin{aligned}\tilde{\mathbf{D}}^{-1}\hat{\mathbf{A}}_{\text{opr}}\vec{\boldsymbol{v}}_{\text{opr}} &= \tilde{\mathbf{D}}^{-1}\lambda_{\text{opr}}\vec{\boldsymbol{v}}_{\text{opr}},\\ &= \lambda_{\text{opr}}\tilde{\mathbf{D}}^{-1}\vec{\boldsymbol{v}}_{\text{opr}}.\end{aligned} \tag{B.7}$$

From Eqs. (B.6) and (B.7) we can get $\lambda_{\text{opr}} = \lambda_{\text{mean}}$ by:

$$\begin{aligned}\hat{\mathbf{A}}_{\text{mean}}\tilde{\mathbf{D}}^{-1}\vec{\boldsymbol{v}}_{\text{opr}} &= \lambda_{\text{opr}}\tilde{\mathbf{D}}^{-1}\vec{\boldsymbol{v}}_{\text{opr}},\\ &= \lambda_{\text{mean}}\tilde{\mathbf{D}}^{-1}\vec{\boldsymbol{v}}_{\text{opr}} = \lambda_{\text{mean}}\vec{\boldsymbol{v}}_{\text{mean}}.\end{aligned} \tag{B.8}$$

Each eigenvector of the matrix $\hat{\mathbf{A}}_{\text{mean}}$ scaled by $\tilde{\mathbf{D}}^{-1}$ is the related eigenvector of the OPR matrix $\hat{\mathbf{A}}_{\text{opr}}$. Therefore, these three symmetric matrices $\hat{\mathbf{A}}_{\text{sn}}$, $\hat{\mathbf{A}}_{\text{mean}}$, and $\hat{\mathbf{A}}_{\text{opr}}$ have the same eigenvalues and thereby $\lambda_{\text{sn}} = \lambda_{\text{opr}} = \lambda_{\text{mean}} \leq 1$. □

Appendix C. Proof of Lemma 5

Proof. As any real symmetric matrix can be diagonalized [25], the real symmetric matrix $\hat{\mathbf{A}}_{\text{sn}}$ is diagonalizable. Diagonalizing $\hat{\mathbf{A}}_{\text{sn}}$ as $\mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}\mathbf{P}^{-1}$, there is:

$$\begin{aligned}\hat{\mathbf{A}}_{\text{sn}}^{2k} &= \hat{\mathbf{A}}_{\text{sn}}\cdots\hat{\mathbf{A}}_{\text{sn}},\\ &= \mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}\mathbf{P}^{-1}\cdots\mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}\mathbf{P}^{-1} = \mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}^{2k}\mathbf{P}^{-1},\end{aligned} \tag{C.1}$$

where $\boldsymbol{\Lambda}_{\text{sn}}$ is a diagonal matrix with the eigenvalues of $\hat{\mathbf{A}}_{\text{sn}}$. Based on Eq. (C.1), the aggregation matrix $\hat{\mathbf{A}}_{\text{isn}}$ of the injective SN aggregator can be diagonalized as:

$$\begin{aligned}\hat{\mathbf{A}}_{\text{isn}} &= \beta\mathbf{I}_N + \hat{\mathbf{A}}_{\text{sn}}^{2k},\\ &= \beta\mathbf{I}_N + \mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}^{2k}\mathbf{P}^{-1},\\ &= \beta\mathbf{P}\mathbf{P}^{-1} + \mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}^{2k}\mathbf{P}^{-1},\\ &= \mathbf{P}(\beta\mathbf{P}^{-1} + \boldsymbol{\Lambda}_{\text{sn}}^{2k}\mathbf{P}^{-1}),\\ &= \mathbf{P}(\beta\mathbf{I}_N + \boldsymbol{\Lambda}_{\text{sn}}^{2k})\mathbf{P}^{-1}.\end{aligned} \tag{C.2}$$

From Eq. (C.2) we can obtain the eigenvalue of $\hat{\mathbf{A}}_{\text{isn}}$ to be $\lambda_{\text{isn}} = \beta + \lambda_{\text{sn}}^{2k}$. According to $\lambda_{\text{sn}} \leq 1$ in Lemma 4, the eigenvalues of $\hat{\mathbf{A}}_{\text{isn}}$ fall in the range of $\beta \leq \lambda_{\text{isn}} \leq 1 + \beta$ to avoid the numerical value 0, making $\hat{\mathbf{A}}_{\text{isn}}$ full rank and its associated aggregator to be an injective function based on Lemmas 2 and 3.

In the same way, diagonalizing the real symmetric matrices $\hat{\mathbf{A}}_{\text{imean}}$ and $\hat{\mathbf{A}}_{\text{iopr}}$ can deduce their corresponding aggregators to be injective. □

Appendix D. Proof of Lemma 6

Proof. The symmetric matrix $\hat{\mathbf{A}}_{\text{ppnp}}$ can be diagonalized with the symmetrically normalized matrix $\hat{\mathbf{A}}_{\text{sn}} = \mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}\mathbf{P}^{-1}$:

$$\begin{aligned}\hat{\mathbf{A}}_{\text{ppnp}} &= \alpha(\mathbf{I}_N - (1-\alpha)\hat{\mathbf{A}}_{\text{sn}})^{-1},\\ &= \alpha(\mathbf{I}_N - (1-\alpha)\mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}\mathbf{P}^{-1})^{-1},\\ &= \alpha(\mathbf{P}\mathbf{P}^{-1} - (1-\alpha)\mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}\mathbf{P}^{-1})^{-1},\\ &= \alpha(\mathbf{P}(\mathbf{I}_N - (1-\alpha)\boldsymbol{\Lambda}_{\text{sn}})\mathbf{P}^{-1})^{-1},\\ &= \alpha((\mathbf{I}_N - (1-\alpha)\boldsymbol{\Lambda}_{\text{sn}})\mathbf{P}^{-1})^{-1}\mathbf{P}^{-1},\\ &= \alpha\mathbf{P}(\mathbf{I}_N - (1-\alpha)\boldsymbol{\Lambda}_{\text{sn}})^{-1}\mathbf{P}^{-1},\\ &= \mathbf{P}\alpha(\mathbf{I}_N - (1-\alpha)\boldsymbol{\Lambda}_{\text{sn}})^{-1}\mathbf{P}^{-1}.\end{aligned} \tag{D.1}$$

Because $\alpha(\mathbf{I}_N - (1-\alpha)\boldsymbol{\Lambda}_{\text{sn}})^{-1}$ is a diagonal matrix, so $\lambda_{\text{ppnp}} = \frac{\alpha}{1-(1-\alpha)\lambda_{\text{sn}}}$. Due to $\lambda_{\text{sn}} \leq 1$ (shown in Lemma 4) and $0 < \alpha \leq 1$ (shown in Eq. (6)), λ_{ppnp} is always greater than 0. As a result, $\hat{\mathbf{A}}_{\text{ppnp}}$ is full rank and its associated aggregator is bijective (injective and surjective) by virtue of Lemmas 2 and 3. □

Appendix E. Proof of Lemma 7

Proof. Being similar to the injectivity proof of PPNP aggregator represented in Appendix D, we use the diagonalizable property of the symmetrically normalized Laplacian matrix $\hat{\mathbf{A}}_{\text{sn}} = \mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}\mathbf{P}^{-1}$ to demonstrate the injectivity of APPNP aggregator. Due to Eq. (C.1), there is:

$$\begin{aligned}&\hat{\mathbf{A}}_{\text{appnp}}\\ &=(1-\alpha)^k\hat{\mathbf{A}}_{\text{sn}}^k + \alpha\sum_{i=0}^{k-1}(1-\alpha)^i\hat{\mathbf{A}}_{\text{sn}}^i,\\ &=(1-\alpha)^k(\mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}\mathbf{P}^{-1})^k + \alpha\sum_{i=0}^{k-1}(1-\alpha)^i(\mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}\mathbf{P}^{-1})^i,\\ &=(1-\alpha)^k\mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}^k\mathbf{P}^{-1} + \alpha\sum_{i=0}^{k-1}(1-\alpha)^i\mathbf{P}\boldsymbol{\Lambda}_{\text{sn}}^i\mathbf{P}^{-1},\\ &=\mathbf{P}((1-\alpha)^k\boldsymbol{\Lambda}_{\text{sn}}^k\mathbf{P}^{-1} + \alpha\sum_{i=0}^{k-1}(1-\alpha)^i\boldsymbol{\Lambda}_{\text{sn}}^i\mathbf{P}^{-1}),\\ &=\mathbf{P}((1-\alpha)^k\boldsymbol{\Lambda}_{\text{sn}}^k + \alpha\sum_{i=0}^{k-1}(1-\alpha)^i\boldsymbol{\Lambda}_{\text{sn}}^i)\mathbf{P}^{-1}.\end{aligned} \tag{E.1}$$

From Eq. (E.1) we can conclude, APPNP aggregation matrix $\hat{\mathbf{A}}_{\text{appnp}}$ can be diagonalized by the matrix $\mathbf{P}$ and is a symmetric matrix. If k is a even integer $2u$ with the integer $u \in \mathbb{N}$, then:

$$\begin{aligned}&\hat{\mathbf{A}}_{\text{appnp}}\\ &=\mathbf{P}((1-\alpha)^k\boldsymbol{\Lambda}_{\text{sn}}^k + \alpha\sum_{i=0}^{k-1}(1-\alpha)^i\boldsymbol{\Lambda}_{\text{sn}}^i)\mathbf{P}^{-1},\\ &=\mathbf{P}((1-\alpha)^{2u}\boldsymbol{\Lambda}_{\text{sn}}^{2u} + \alpha\sum_{i=0}^{u-1}((1-\alpha)^{2i}\boldsymbol{\Lambda}_{\text{sn}}^{2i}+\\ &\quad(1-\alpha)^{2i+1}\boldsymbol{\Lambda}_{\text{sn}}^{2i+1}))\mathbf{P}^{-1},\\ &=\mathbf{P}((1-\alpha)^{2u}\boldsymbol{\Lambda}_{\text{sn}}^{2u} + \alpha\sum_{i=0}^{u-1}(1-\alpha)^{2i}\boldsymbol{\Lambda}_{\text{sn}}^{2i}(\mathbf{I}_N+\\ &\quad(1-\alpha)\boldsymbol{\Lambda}_{\text{sn}}))\mathbf{P}^{-1}.\end{aligned} \tag{E.2}$$

Table J.10
The statistics of datasets.

Dataset	Nodes	Edges	Feature	Classes	Train/Val/Test
Cora	2,708	5,429	1433	7	1208/500/1000
Citeseer	3,327	4,732	3703	6	1827/500/1000
Pubmed	19,717	44,338	500	3	17,717/500/1000
Amazon Photo	7,650	119,043	745	8	160/240/7250
Coauthor Physics	34,493	247,962	8415	5	100/150/34,243
Reddit	232,965	11,606,919	602	41	153,932/23,699/55,334
OGBN-Arxiv	169,343	1,166,243	128	40	90,941/29,799/48,603
OGBN-Products	2,449,029	61,859,140	100	47	196,615/39,323/2,213,091

Table J.11
The statistics of two large-scale graph-based traffic datasets.

Dataset	Length of time series	Nodes	Edges	Feature	Classes	Train/Val/Test
Chengdu	1,756,800	4,266,815	82,320,569	20	3	853,363/426,682/2,986,770
Xi'an	1,756,800	3,852,855	65,209,168	20	3	770,571/385,285/2,696,999

Following Eq. (E.2), the eigenvalue λ_{appnp} of APPNP aggregation matrix $\hat{\mathbf{A}}_{\text{appnp}}$ is:

$$\begin{aligned}\lambda_{\text{appnp}} =&(1-\alpha)^{2u}\lambda_{\text{sn}}^{2u} + \alpha\sum_{i=0}^{u-1}(1-\alpha)^{2i}\lambda_{\text{sn}}^{2i}(1+ \\ &(1-\alpha)\lambda_{\text{sn}}), \\ =&(1-\alpha)^{2u}\lambda_{\text{sn}}^{2u} + \alpha(1+(1-\alpha)\lambda_{\text{sn}})+ \\ &\alpha\sum_{i=1}^{u-1}(1-\alpha)^{2i}\lambda_{\text{sn}}^{2i}(1+(1-\alpha)\lambda_{\text{sn}}).\end{aligned} \tag{E.3}$$

Given $\alpha \in (0, 1)$, and $|\lambda_{\text{sn}}| \le 1$ based on Perron–Frobenius theorem [25] that requires the irreducible matrix $\hat{\mathbf{A}}_{\text{sn}}$, the eigenvalue λ_{appnp} is always greater than 0, *i.e.*, $\lambda_{\text{appnp}} > 0$. Therefore, the symmetric matrix $\hat{\mathbf{A}}_{\text{appnp}}$ is full rank and its associated aggregator is bijective (injective and surjective) based on Lemmas 2 and 3.

On the other hand, if k is a odd integer $2u + 1$, then:

$$\begin{aligned}&\hat{\mathbf{A}}_{\text{appnp}} \\ &=\mathbf{P}((1-\alpha)^{2u+1}\mathbf{\Lambda}_{\text{sn}}^{2u+1} + \alpha(1-\alpha)^{2u}\mathbf{\Lambda}_{\text{sn}}^{2u}+ \\ &\quad\alpha\sum_{i=0}^{u-1}(1-\alpha)^{2i}\mathbf{\Lambda}_{\text{sn}}^{2i}(\mathbf{I}_N+(1-\alpha)\mathbf{\Lambda}_{\text{sn}}))\mathbf{P}^{-1}, \\ &=\mathbf{P}((1-\alpha)^{2u}\mathbf{\Lambda}_{\text{sn}}^{2u}((1-\alpha)\mathbf{\Lambda}_{\text{sn}}+\alpha\mathbf{I}_N)+ \\ &\quad\alpha\sum_{i=0}^{u-1}(1-\alpha)^{2i}\mathbf{\Lambda}_{\text{sn}}^{2i}(\mathbf{I}_N+(1-\alpha)\mathbf{\Lambda}_{\text{sn}}))\mathbf{P}^{-1}.\end{aligned} \tag{E.4}$$

According to Eq. (E.4), the eigenvalue λ_{appnp} of APPNP aggregation matrix $\hat{\mathbf{A}}_{\text{appnp}}$ is:

$$\begin{aligned}\lambda_{\text{appnp}} =&(1-\alpha)^{2u}\lambda_{\text{sn}}^{2u}((1-\alpha)\lambda_{\text{sn}}+\alpha)+ \\ &\alpha\sum_{i=0}^{u-1}(1-\alpha)^{2i}\lambda_{\text{sn}}^{2i}(1+(1-\alpha)\lambda_{\text{sn}}), \\ =&(1-\alpha)^{2u}\lambda_{\text{sn}}^{2u}((1-\alpha)\lambda_{\text{sn}}+\alpha)+\alpha(1+ \\ &(1-\alpha)\lambda_{\text{sn}})+\alpha\sum_{i=1}^{u-1}(1-\alpha)^{2i}\lambda_{\text{sn}}^{2i}(1+ \\ &(1-\alpha)\lambda_{\text{sn}}),\end{aligned} \tag{E.5}$$

where $(1-\alpha)\lambda_{\text{sn}}+\alpha > -1$ and $0 < 1+(1-\alpha)\lambda_{\text{sn}} < 2$, due to $\alpha \in (0, 1)$ and $|\lambda_{\text{sn}}| \le 1$. Hence, λ_{appnp} may be equal to 0 to result in the non-full rank matrix $\hat{\mathbf{A}}_{\text{appnp}}$, which does not ensure APPNP aggregator to be injective.

Besides, if the aggregation matrix $\hat{\mathbf{A}}_{\text{sn}}$ is not irreducible, we can only get $\lambda_{\text{sn}} \le 1$ based on Gershgorin circle theorem [25] and cannot ensure $\lambda_{\text{appnp}} > 0$, thereby, APPNP aggregator with the reducible $\hat{\mathbf{A}}_{\text{sn}}$ is not necessarily injective. □

Appendix F. Proof of Lemma 8

Proof. We use Eq. (C.1) to rewrite Eq. (13) as:

$$\begin{aligned}&\hat{\mathbf{A}}_{\text{eappnp}} \\ &=\beta\mathbf{I}_N+(1-\alpha)^k\hat{\mathbf{A}}_{\text{sn}}^{2k}+\alpha\sum_{i=1}^{k-1}(1-\alpha)^i\hat{\mathbf{A}}_{\text{sn}}^{2i}, \\ &=\beta\mathbf{P}\mathbf{P}^{-1}+(1-\alpha)^k(\mathbf{P}\mathbf{\Lambda}_{\text{sn}}\mathbf{P}^{-1})^{2k}+ \\ &\quad\alpha\sum_{i=1}^{k-1}(1-\alpha)^i(\mathbf{P}\mathbf{\Lambda}_{\text{sn}}\mathbf{P}^{-1})^{2i}, \\ &=\beta\mathbf{P}\mathbf{P}^{-1}+(1-\alpha)^k\mathbf{P}\mathbf{\Lambda}_{\text{sn}}^{2k}\mathbf{P}^{-1}+ \\ &\quad\alpha\sum_{i=1}^{k-1}(1-\alpha)^i\mathbf{P}\mathbf{\Lambda}_{\text{sn}}^{2i}\mathbf{P}^{-1}, \\ &=\mathbf{P}(\beta\mathbf{P}^{-1}+(1-\alpha)^k\mathbf{\Lambda}_{\text{sn}}^{2k}\mathbf{P}^{-1}+ \\ &\quad\alpha\sum_{i=1}^{k-1}(1-\alpha)^i\mathbf{\Lambda}_{\text{sn}}^{2i}\mathbf{P}^{-1}), \\ &=\mathbf{P}(\beta\mathbf{I}_N+(1-\alpha)^k\mathbf{\Lambda}_{\text{sn}}^{2k}+\alpha\sum_{i=1}^{k-1}(1-\alpha)^i\mathbf{\Lambda}_{\text{sn}}^{2i})\mathbf{P}^{-1}.\end{aligned} \tag{F.1}$$

By virtue of Eq. (F.1), we obtain the eigenvalues of EP APPNP aggregation matrix:

$$\lambda_{\text{eappnp}} = \beta+(1-\alpha)^k\lambda_{\text{sn}}^{2k}+\alpha\sum_{i=1}^{k-1}(1-\alpha)^i\lambda_{\text{sn}}^{2i}. \tag{F.2}$$

When $\alpha \in (0, 1]$, $\lambda_{\text{sn}} \le 1$ confirmed in Lemma 4, and $\beta \in (0, 1]$, then $\lambda_{\text{eappnp}} > 0$. Hence, the aggregation matrix $\hat{\mathbf{A}}_{\text{eappnp}}$ is full rank and its associated aggregator is injective according to Lemmas 2 and 3. □

Appendix G. Proof of Lemma 10

Proof. Let each GNN layer in the recursive process be a mapping $\mathcal{L}^{(l)} = \mathcal{E}^{(l)} \circ \mathcal{F}$. According to Definition 1 and Lemma 2, we get:

$$C_{\mathcal{E}^{(l)}\circ\mathcal{F}|\mathcal{F}\text{ is injective}} > C_{\mathcal{E}^{(l)}\circ\mathcal{F}|\mathcal{F}\text{ is not injective}}. \tag{G.1}$$

Eq. (G.1) states that each aggregation layer of $\{\mathcal{A}_{\text{rec}}|\mathcal{F}\text{ is injective}\}$ can capture more representation capacity than its non-injective version $\{\mathcal{A}_{\text{rec}}|\mathcal{F}$ is not injective$\}$, *i.e.*,

$$C_{\mathcal{A}_{\text{rec}}|\mathcal{F} \text{ is injective}} > C_{\mathcal{A}_{\text{rec}}|\mathcal{F} \text{ is not injective}}. \tag{G.2}$$

Eq. (G.2) hence means:

$$C_{\mathcal{A}_{\text{rec}}|\mathcal{F} \text{ is injective}} \geq C_{\mathcal{A}_{\text{rec}}}. \quad \square \tag{G.3}$$

Appendix H. Proof of Lemma 11

Proof. Let the aggregation pipeline in AE framework be the mapping $\mathcal{F}_{\text{agg}} = \mathcal{F} \circ \cdots \circ \mathcal{F}$ and the associated encoding pipeline be the mapping $\mathcal{E}_{\text{enc}} = \mathcal{E}^{(L)} \circ \cdots \circ \mathcal{E}^{(1)}$. Depended on Lemma 9, $\mathcal{F}_{\text{agg}}$ is injective if $\mathcal{F}$ is injective, and $\mathcal{F}_{\text{agg}}$ is non-injective if $\mathcal{F}$ is non-injective, *i.e.*, $C_{\mathcal{F}_{\text{agg}}|\mathcal{F} \text{ is injective}} > C_{\mathcal{F}_{\text{agg}}|\mathcal{F} \text{ is not injective}}$ by virtue of Definition 1 and Lemma 2. Hence, there is:

$$C_{\mathcal{A}_{\text{ae}}|\mathcal{F} \text{ is injective}} \geq C_{\mathcal{A}_{\text{ae}}}. \quad \square \tag{H.1}$$

Appendix I. Proof of Lemma 12

Proof. Similar to proving Lemma 11, we get $C_{\mathcal{F}_{\text{agg}}|\mathcal{F} \text{ is injective}} > C_{\mathcal{F}_{\text{agg}}|\mathcal{F} \text{ is not injective}}$ by virtue of Definition 1, Lemmas 2 and 9. We hence obtain:

$$C_{\mathcal{A}_{\text{ea}}|\mathcal{F} \text{ is injective}} \geq C_{\mathcal{A}_{\text{ea}}}. \quad \square \tag{I.1}$$

Appendix J. Dataset statistics

All dataset statistics are shown in Tables J.10 and J.11. Note that the train/val/test splits of Cora, Citeseer, and Pubmed use the splits of these corresponding datasets in Paper [20], in order to run mini-batch experiments for AE framework conveniently.

References

[1] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, S Yu Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. 32 (1) (2020) 4–24.

[2] Guotong Xue, Ming Zhong, Jianxin Li, Jia Chen, Chengshuai Zhai, Ruochen Kong, Dynamic network embedding survey, Neurocomputing 472 (2022) 212–223.

[3] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al., Deep speech 2: End-to-end speech recognition in english and mandarin, in: International Conference on Machine Learning, PMLR, 2016, pp. 173–182.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 2019, pp. 4171–4186.

[5] Marcin Woźniak, Jakub Siłka, Michał Wieczorek, Mubarak Alrashoud, Recurrent neural network model for IoT and networking malware threat detection, IEEE Trans. Ind. Inform. 17 (8) (2020) 5583–5594.

[6] Shengjie Min, Zhan Gao, Jing Peng, Liang Wang, Ke Qin, Bo Fang, STGSN—A spatial–temporal graph neural network framework for time-evolving social networks, Knowl. Based Syst. 214 (2021) 106746.

[7] Ruiping Yin, Kan Li, Guangquan Zhang, Jie Lu, A deeper graph neural network for recommender systems, Knowl. Based Syst. 185 (2019) 105020.

[8] Xiangyu Song, Jianxin Li, Yifu Tang, Taige Zhao, Yunliang Chen, Ziyu Guan, Jkt: A joint graph convolutional network based deep knowledge tracing, Inform. Sci. 580 (2021) 510–523.

[9] Xiangyu Song, Jianxin Li, Qi Lei, Wei Zhao, Yunliang Chen, Ajmal Mian, Bi-CLKT: Bi-graph contrastive learning based knowledge tracing, 2022, arXiv preprint arXiv:2201.09020.

[10] Dan Jiang, Ronggui Wang, Juan Yang, Lixia Xue, Kernel multi-attention neural network for knowledge graph embedding, Knowl.-Based Syst. (2021) 107188.

[11] Pengshuai Yin, Jiayong Ye, Guoshen Lin, Qingyao Wu, Graph neural network for 6D object pose estimation, Knowl. Based Syst. 218 (2021) 106839.

[12] Zhao-Min Chen, Xiu-Shen Wei, Peng Wang, Yanwen Guo, Multi-label image recognition with graph convolutional networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 5177–5186.

[13] Thomas Kipf, Max Welling, Semi-supervised classification with graph convolutional networks, in: International Conference on Learning Representations, 2017.

[14] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, Maosong Sun, Graph neural networks: A review of methods and applications, AI Open 1 (2020) 57–81.

[15] Shuiqiao Yang, Sunny Verma, Borui Cai, Jiaojiao Jiang, Kun Yu, Fang Chen, Shui Yu, Variational co-embedding learning for attributed network clustering, 2021, arXiv preprint arXiv:2104.07295.

[16] Hui Yin, Shuiqiao Yang, Xiangyu Song, Wei Liu, Jianxin Li, Deep fusion of multimodal features for social media retweet time prediction, World Wide Web 24 (4) (2021) 1027–1044.

[17] Jiexia Ye, Juanjuan Zhao, Kejiang Ye, Chengzhong Xu, How to build a graph-based deep learning architecture in traffic domain: A survey, IEEE Trans. Intell. Transp. Syst. (2020).

[18] Fan Zhou, Qing Yang, Ting Zhong, Dajiang Chen, Ning Zhang, Variational graph neural networks for road traffic prediction in intelligent transportation systems, IEEE Trans. Ind. Inform. 17 (4) (2020) 2802–2812.

[19] William L. Hamilton, Rex Ying, Jure Leskovec, Inductive representation learning on large graphs, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 1025–1035.

[20] Yuning You, Tianlong Chen, Zhangyang Wang, Yang Shen, L2-gcn: Layer-wise and learned efficient training of graph convolutional networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 2127–2135.

[21] Johannes Klicpera, Aleksandar Bojchevski, Stephan Günnemann, Predict then propagate: Graph neural networks meet personalized pagerank, in: International Conference on Learning Representations, 2019.

[22] Sergey Brin, The PageRank citation ranking: bringing order to the web, Proc. ASIS 98 (1998) 161–172.

[23] Taher H. Haveliwala, Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search, IEEE Trans. Knowl. Data Eng. 15 (4) (2003) 784–796.

[24] Dénes Petz, Entropy, von Neumann and the von Neumann entropy, Springer, 2001.

[25] Roger A. Horn, Charles R. Johnson, Matrix Analysis, Cambridge University Press, 2012.

[26] Xavier Bresson, Thomas Laurent, Residual gated graph convnets, 2017, ArXiv Preprint arXiv:1711.07553.

[27] Guohao Li, Matthias Muller, Ali Thabet, Bernard Ghanem, Deepgcns: Can gcns go as deep as cnns? in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 9267–9276.

[28] Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka, How powerful are graph neural networks? in: International Conference on Learning Representations, 2018.

[29] A.A. Leman, B. Weisfeiler, A reduction of a graph to a canonical form and an algebra arising during this reduction, Nauchno-Techn. Inform. 2 (9) (1968) 12–16.

[30] László Babai, Ludik Kucera, Canonical labelling of graphs in linear average time, in: 20th Annual Symposium on Foundations of Computer Science (Sfcs 1979), IEEE, 1979, pp. 39–46.

[31] Roger A. Horn, Charles R. Johnson, Matrix Analysis, Cambridge University Press, 2012.

[32] Michaël Defferrard, Xavier Bresson, Pierre Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, Adv. Neural Inform. Process. Syst. 29 (2016) 3844–3852.

[33] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, Kilian Weinberger, Simplifying graph convolutional networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 6861–6871.

[34] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, Federico Monti, Sign: Scalable inception graph neural networks, 2020, ArXiv Preprint arXiv:2004.11198.

[35] David A. Sprecher, Elements of Real Analysis, Courier Corporation, 2012.

[36] Jie Chen, Tengfei Ma, Cao Xiao, FastGCN: Fast learning with graph convolutional networks via importance sampling, in: International Conference on Learning Representations, 2018.

[37] Jianfei Chen, Jun Zhu, Le Song, Stochastic training of graph convolutional networks with variance reduction, in: International Conference on Machine Learning, PMLR, 2018, pp. 942–950.

[38] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, Cho-Jui Hsieh, Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 257–266.

[39] Yue Cao, Tianlong Chen, Zhangyang Wang, Yang Shen, Learning to optimize in swarms, Adv. Neural Inform. Process. Syst. 32 (2019) 15044.

[40] Ronald J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (3) (1992) 229–256.

[41] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, Yoshua Bengio, Graph attention networks, in: International Conference on Learning Representations, 2019.

[42] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, Michael M Bronstein, Geometric deep learning on graphs and manifolds using mixture model cnns, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5115–5124.

[43] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, Stephan Günnemann, Pitfalls of graph neural network evaluation, 2018, ArXiv Preprint arXiv:1811.05868.

[44] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, Jure Leskovec, Open graph benchmark: Datasets for machine learning on graphs, 2020, ArXiv Preprint arXiv:2005.00687.