PersonaSAGE: A Multi-Persona Graph Neural Network

Gautam Choudhary
Adobe Research
gc.iitr@gmail.com

Iftikhar Ahamath Burhanuddin
Adobe Research
burhanud@adobe.com

Eunyee Koh
Adobe Research
eunyee@adobe.com

Fan Du
Adobe Research
fdu@adobe.com

Ryan A. Rossi
Adobe Research
ryrossi@adobe.com

ABSTRACT

Graph Neural Networks (GNNs) have become increasingly important in recent years due to their state-of-the-art performance on many important downstream applications. Existing GNNs have mostly focused on learning a single node representation, despite that a node often exhibits polysemous behavior in different contexts. In this work, we develop a persona-based graph neural network framework called PersonaSAGE that learns multiple persona-based embeddings for each node in the graph. Such disentangled representations are more interpretable and useful than a single embedding. Furthermore, PersonaSAGE learns the appropriate set of persona embeddings for each node in the graph, and every node can have a different number of assigned persona embeddings. The framework is flexible enough and the general design helps in the wide applicability of the learned embeddings to suit the domain. We utilize publicly available benchmark datasets to evaluate our approach and against a variety of baselines. The experiments demonstrate the effectiveness of PersonaSAGE for a variety of important tasks including link prediction where we achieve an average gain of 15% while remaining competitive for node classification. Finally, we also demonstrate the utility of PersonaSAGE with a case study for personalized recommendation of different entity types in a data management platform.

KEYWORDS

Persona, Graph Neural Networks, Node Embeddings, Disentangled Representation Learning

1 INTRODUCTION

In recent years, Graph Neural Networks (GNNs) have become increasingly important due to their state-of-the-art performance on many important downstream applications including link prediction and node classification [5, 15, 23, 24, 27, 31, 32]. Existing GNNs have mostly focused on learning a single node embedding (or representation) [23, 32], despite that a node often exhibits polysemous behavior in different contexts [3]. For instance, an individual may have many different personas, e.g., a user may be a researcher, father, coach, and activist [14, 22]. These personas may be fundamentally different or even impossible for other individuals. Each one of these general personas may also have sub-personas that capture specific behaviors and characteristics of each, which we call sub-personas. However, existing methods are unable to learn such sets of persona embeddings for each node in the graph.

To address this limitation, we develop a persona-based graph neural network framework called PersonaSAGE that learns a set of

Figure 1: Overview of the learning objectives of PersonaSAGE and prior work. While previous work has mainly focused on learning single embedding per node, a few recent works learn multiple embeddings per node. In contrast, PersonaSAGE not only learns multiple persona embeddings but also learns the appropriate number of persona embeddings per node.

embeddings for each node in the graph. Furthermore, PersonaSAGE learns the appropriate set of persona embeddings for each node in the graph, and every node can have a different number of assigned personas. In Figure 1, we provide an intuitive overview comparing the learning objective of PersonaSAGE to previous work. Consider the real-world scenario of roles in a data management platform. The central node is that of a data science manager and is represented by two embeddings: blue and green. The green embedding denotes the role of the manager (and their sub-roles). The other nodes being individual contributors such as a data analyst, data engineer or data scientist are represented by embeddings other than the green embedding based on their respective roles. The manager doubles as one of these roles, for instance, data scientist, and is additionally represented by a blue embedding. This scenario illustrates the need for different nodes in the graph to be represented by a varying number of embeddings depending on their role and community membership. Each set of embeddings collectively represents the node. Notably, as shown in Figure 1, every node can have a different set of embeddings along with a different number of assigned embeddings as well. For instance, one node in Figure 1 has 3 embedding vectors compared to the other nodes that have only 2 embeddings. Furthermore, the nodes with 2 embedding vectors are also distinct,

representing different personas (behaviors and structural characteristics) altogether, which is shown by the color of the embedding vectors in Figure 1.

The experiments demonstrate the effectiveness of our approach for link prediction and node classification. Overall, we find that PersonaSAGE always outperforms the other methods across all graphs and prediction tasks. For link prediction, PersonaSAGE achieves a mean gain of 15% over the best baseline method across all graphs. PersonaSAGE also performs well for node classification achieving a mean gain of 1% and up to 17% over the other methods. Furthermore, we also conducted a case study in Section 4 where PersonaSAGE is applied for personalized recommendation of queries, attributes, and datasets. In this case study, PersonaSAGE learns multiple embeddings per node from a large heterogeneous graph derived from the usage logs of users (from a data management platform). Notably, PersonaSAGE significantly outperforms the other methods, achieving a gain of 19.2%, 15.7%, and 21.7% in AUC over the best baseline method for query, attribute, and dataset recommendation tasks, respectively. These results demonstrate the effectiveness of PersonaSAGE and its ability to learn sets of embeddings for every node in the graph that appropriately capture the contextual behavior and personas of the nodes.

The main contributions are as follows:

- **Problem Formulation:** We introduce the problem of automatically learning sets of embeddings for each node in the graph, which may also be of different sizes depending on the structural characteristics surrounding the node.
- **Novel Framework:** This work develops PersonaSAGE, a novel graph neural network framework that learns multiple embeddings for every node, which are also flexible in size. Hence, PersonaSAGE may learn 2 embeddings for one node and 3 embeddings for another, thus, every node is automatically assigned the appropriate set of embeddings.
- **Effectiveness:** Through comprehensive experiments, PersonaSAGE is shown to be extremely effective for a wide variety of application tasks including link prediction and node classification. Notably, PersonaSAGE outperforms the other methods across a wide variety of graphs. Finally, we also demonstrate the utility of PersonaSAGE on a case study where we apply these techniques for personalized recommendation of queries, datasets, and attributes in a data management platform.

2 PersonaSAGE FRAMEWORK

In this section, we describe the proposed approach called PersonaSAGE. We first provide an overview and then detail the algorithm for the generation of multiple sets of persona embeddings. Later, we describe specific choices involved in our approach such as persona assignments and the kind of aggregations involved. An end-to-end optimization problem is designed to learn the parameters for downstream applications such as link prediction and node classification.

2.1 Problem Formulation

In this work, we investigate the new problem formally described below. Given an arbitrary graph $G = (\mathcal{V}, \mathcal{E})$ along with its adjacency matrix $\mathbf{A}$, the problem is to learn multiple embeddings $\mathcal{X}_v = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$ for every node $v \in \mathcal{V}$ where (i) for any two different nodes $u, v \in \mathcal{V}$, $|\mathcal{X}_u| \neq |\mathcal{X}_v|$ may hold and (ii) $|\mathcal{X}_u \cup \mathcal{X}_v| = |\mathcal{X}_u| + |\mathcal{X}_v|$. Intuitively, (i) implies that different nodes can have different number of embeddings whereas (ii) implies that the embeddings learned for any node in the graph are unique (non-identical), otherwise $\exists\, u, v \in \mathcal{V}, |\mathcal{X}_u \cup \mathcal{X}_v| < |\mathcal{X}_u| + |\mathcal{X}_v|$ would hold. Let $K = \max_{v \in \mathcal{V}} |\mathcal{X}_v|$ denote the maximum number of embeddings per node in G such that $K \ll n$, where n is the number of nodes in G. Without loss of generality, all embeddings are assumed to be of fixed size D, *i.e.*, $(\mathbf{x}_i \in \mathcal{X}_v) \in \mathbb{R}^D$. Intuitively, each learned embedding $\mathbf{x}_i \in \mathcal{X}_v$ represents a set of "sub-personas" for a given "structural context", which are also automatically learned from the data.[1] As an aside, in this work, uppercase letters in calligraphic bold font denote sets, $\mathcal{X}$.

2.2 Overview

Figure 2 highlights the given inputs and desired outputs of the system and an overview of our proposed model, PersonaSAGE. As shown, given a sample graph comprising $n = 6$ nodes and their node embeddings, we compute a reference persona label for each node through a clustering algorithm by assuming K sets of personas in the input graph data. Suppose that $K = 4$ personas are considered and each persona is represented by a distinct color. To visualize, assume the output cluster labels as highlighted in *Step 1* of the figure. We can convert them into one-hot characteristic encodings, $\mathbf{c}_v \in \mathbb{R}^K$ for each node v in the graph. We refer to these encodings as initial persona membership vectors whose each element defines the degree of association of every persona to the given node. For each node, by using an initial node embedding and persona membership vector as input, each layer of PersonaSAGE iteratively updates them based on the neighboring nodes as highlighted in *Step 2* in the figure. The model finally yields a set of persona embeddings and a corresponding updated persona membership vector for each node. The desired output for this example is represented in the last block of the figure where node A has a higher membership for the *red* persona than the *violet* and has 2 corresponding persona embeddings as intuitive from its neighborhood. Similarly, node B has a persona embedding for each of the 3 persona memberships, namely *red*, *violet*, and *blue* with proportions relative to its neighborhood. Our model is capable of finding the appropriate number of personas for each node. In the next section, we describe the update procedures for the persona membership vector as well as their corresponding persona embeddings for each node.

2.3 Algorithm

The core idea behind the PersonaSAGE algorithm is to represent each node as a set of persona embeddings where the cardinality of this set may differ for different nodes. Let K denote the maximum number of persona embeddings (or embedding vectors) per node. For instance, $K = 4$ in Figure 1 as a user may have four embedding

---

[1]This is in contrast to recent work [8] that assumes the different contexts are given as input and simply learns an embedding for each one.

Figure 2: Overview of our proposed approach, PersonaSAGE.

vectors since there are four different colors denoting the various persona embedding vectors. Furthermore, let D denote the maximum embedding size for the K different persona embeddings that a node in G can be assigned. Our approach models the persona representation of a node v as a combination of two aspects: (i) a persona membership vector $\mathbf{c}_v \in \mathbb{R}^K$ of unit length ($\|\mathbf{c}_v\|_1 = 1$) denoting proportionate degree of membership for each persona, and (ii) the corresponding persona embeddings for non-zero persona memberships, $\boldsymbol{\mathcal{X}}_v = \{\mathbf{x}_i \mid \mathbf{c}_{v,i} > 0, \mathbf{x}_i \in \mathbb{R}^D\}$. We represent $\mathbf{c}_{v,i}$ to denote the degree of engagement or association of a particular node v to a persona $i \in 1, \ldots, K$ while $\mathbf{x}_i$ denotes the corresponding persona embedding in a D dimensional vector space.

The forward propagation algorithm for PersonaSAGE is formally defined in Algorithm 1. Given a graph, $G(\mathcal{V}, \mathcal{E})$, its initial set of node embeddings $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$ and the number of personas K to be computed, along with the given set of initial membership vectors $\{\mathbf{c}_v \mid v \in \mathcal{V}\}$ as prior knowledge, the algorithm finds multiple (and different) sets of persona embeddings $\boldsymbol{\mathcal{X}}_v$ for each node v in the graph. We also provide a strategy to learn membership vectors in the next section, in case the prior knowledge is not known.

The model initializes each persona embedding for a given node $v \in \mathcal{V}$ from the same input node features of that node to avoid any bias in Line 1. For each layer l of the network, the algorithm performs two updates. The first update happens for persona membership vector $\mathbf{C}_v^l$ by aggregating information from its neighbors $\mathcal{N}(v)$ and then applying normalization in Line 5 and 6 respectively. We assume that the persona membership vector of a node is likely to be influenced by that of its neighbors and we update it at each layer of the network. This update strategy is based on the intuition that the *persona* of a node is a function of itself and that of its neighbors where the neighborhood of a node is defined as its directly connected nodes. The next update happens separately for each persona, where the updated membership vector is used to condition the neighboring persona embeddings of the previous layer $\{\mathbf{X}_{u,i}^{l-1}, u \in \mathcal{N}(v)\}$ for a reference node v before applying a suitable aggregator function as described in Line 10. Intuitively, for the i-th persona embedding of a node is updated based on i-th persona embeddings of its neighbors conditioned upon their persona memberships $\mathbf{C}_{u,i}^l$. We discuss various alternatives for aggregation functions in a subsequent section. The aggregated information from neighbors' i-th persona $\mathbf{h}_{\mathcal{N}(v),i}^l$ is then concatenated with the node's own i-th persona embedding $\{\mathbf{X}_{v,i}^{l-1}\}$ and passed through a neural network (affine transformations followed by a non-linear activation function $\sigma(\cdot)$ such as Sigmoid) where $\mathbf{W}^l$ denote learnable weight matrices of this neural network. Note that a single neural

network is trained per layer and is the same for each persona. Both the update steps could be individually parallelized at the node level for efficient computation. Finally, the set of embeddings for which persona memberships are non-zero is returned. To reiterate, we aim to disentangle the node embeddings as a set of multiple persona embeddings conditioned by the persona membership vector.

The time complexity of the forward propagation phase of Algorithm 1 is $O(L \cdot n \cdot K \cdot d)$, where n is the cardinality of $\mathcal{V}$, and d is an upper bound of the maximum degree of the nodes. This time complexity can be significantly reduced by accounting for matrix computation and parallelism. We examine various clustering algorithms (*e.g.*, KMeans, Ward) along with a choice of aggregation functions in an ablation study in Section 3.

Algorithm 1: PersonaSAGE Forward Propagation

Input : Graph $G(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; number of clusters K; membership vectors $\{\mathbf{c}_v \in \mathbb{R}^K, \forall v \in \mathcal{V}\}$; number of layers L

Output: Persona Embeddings $\boldsymbol{\mathcal{X}}_v, \forall v \in \mathcal{V}$

    /* Initialisation   */
1 $\mathbf{X}_{v,i}^0 = \mathbf{x}_v, \forall i \in \{1, \ldots, K\}, \forall v \in \mathcal{V}$;
2 $\mathbf{C}_v^0 = \mathbf{c}_v, \forall v \in \mathcal{V}$;
3 **for** $l \leftarrow 1$ **to** L **do**
      /* Persona Membership Update   */
4   **for** $v \in \mathcal{V}$ **do**
5          $\mathbf{C}_v^l = \mathbf{C}_v^{l-1} + \sum_{u \in \mathcal{N}(v)} \mathbf{C}_u^{l-1}$;
6          $\mathbf{C}_v^l = \mathbf{C}_v^l / \|\mathbf{C}_v^l\|_1$;
7   **end**
      /* Persona Embeddings Update   */
8   **for** $v \in \mathcal{V}$ **do**
9   **for** $i \leftarrow 1$ **to** K **do**
10          $\mathbf{h}_{\mathcal{N}(v),i}^l = f_{aggregate}^l \left(\left\{ (\mathbf{C}_{u,i}^l, \mathbf{X}_{u,i}^{l-1}) \mid u \in \mathcal{N}(v) \right\} \right)$;
11          $\mathbf{X}_{v,i}^l = \sigma \left(\mathbf{W}^l \cdot CONCAT \left[\mathbf{X}_{v,i}^{l-1}, \mathbf{h}_{\mathcal{N}(v),i}^l \right] \right)$;
12   **end**
13   **end**
14 **end**
15 $\boldsymbol{\mathcal{X}}_v = \left\{ \mathbf{X}_{v,i} \mid \mathbf{C}_{v,i} > 0, \forall i \in \{1, \ldots, K\} \right\}$;
16 **return** $\boldsymbol{\mathcal{X}}_v$;

2.4 Persona Assignment

We now describe how to obtain initial persona membership vectors which our algorithm relies on as prior knowledge. We pose it as a clustering problem by partitioning the nodes in a graph G into K mutually exclusive sets. More formally, a node v belonging to cluster i will have a persona membership vector $\mathbf{c}_v = \mathbf{e}_i \in \{0, 1\}^K$, *i.e.*, a one-hot encoded vector. We investigate various clustering algorithms such as KMeans, Spectral Clustering, Ward's Hierarchical Clustering, etc. for partitioning nodes in a graph. The clustering outputs a label denoting a hard assignment of a persona to each node. Our algorithm leverages this static assignment and automatically learns new personas for every node based on their neighborhood.

2.5 Aggregator Functions

An aggregator function is used to collect and transmit important and combined information from neighboring nodes $\mathcal{N}(v)$ to a given node v. Since this information aggregation is not tied to a specific order, the aggregator functions are to be order invariant. We investigate three kinds of aggregator functions that are used to combine the persona information from neighborhood nodes $\mathcal{N}(v)$ to a reference node v. Each describes how the persona membership value $c \in \mathbb{R}_{\geq 0}$ combine with its corresponding persona embedding $\mathbf{x}_u \in \mathbb{R}^D$ and interact in the neighborhood $\mathcal{N}(v)$ of node v.

2.5.1 Mean. This aggregation performs an average operation of the information collected from neighbors, i.e.,

$$f_{aggregate}\left(\{(c, \mathbf{x}_u) \mid u \in \mathcal{N}(v)\}\right) = \frac{\sum_{u \in \mathcal{N}(v)} c \cdot \mathbf{x}_u}{|\mathcal{N}(v)|}$$

2.5.2 Sum. This aggregation sums up the information collected from neighbors, i.e.,

$$f_{aggregate}\left(\{(c, \mathbf{x}_u) \mid u \in \mathcal{N}(v)\}\right) = \sum_{u \in \mathcal{N}(v)} c \cdot \mathbf{x}_u$$

2.5.3 Max. This aggregation performs the element-wise max operation of the neighboring node vectors, i.e.,

$$f_{aggregate}\left(\{(c, \mathbf{x}_u) \mid u \in \mathcal{N}(v)\}\right) = [\max_{u \in \mathcal{N}(v)} (c \cdot \mathbf{x}_{u,1}), ...]$$

2.6 Training and Optimization

Until now, we have discussed sets of persona embeddings with different cardinalities. For sake of practical usage in downstream tasks, we may need to obtain a single embedding of fixed length for each node. We can use multiple aggregation strategies. A straightforward option is an orderly concatenation of all K persona embeddings:

$$\tilde{\mathbf{X}}_v = \mathbf{X}_{v,1} \oplus \mathbf{X}_{v,2} \oplus \ldots \oplus \mathbf{X}_{v,K}$$

where we do not condition them with persona memberships. Another way is to explicitly condition them by scaling the embeddings with persona membership vector before concatenation:

$$\tilde{\mathbf{X}}_v = (\mathbf{C}_{v,1} \cdot \mathbf{X}_{v,1}) \oplus (\mathbf{C}_{v,2} \cdot \mathbf{X}_{v,2}) \oplus \ldots \oplus (\mathbf{C}_{v,K} \cdot \mathbf{X}_{v,K})$$

and return this embedding than the set of persona embeddings which is shown in Line 15 in Algorithm 1. We adopt, the above formulation for computing final node embeddings in the experiments. Here, $\oplus$ denotes the concatenation operation.

Now, this vector can be easily used for downstream tasks. We describe two benchmark tasks: link prediction and node classification. For link prediction, the similarity score between two nodes u, v is defined as the inner vector product: $sim(u, v) = \tilde{\mathbf{X}}_u \odot \tilde{\mathbf{X}}_v$. For node classification, the node embeddings $\tilde{\mathbf{X}}_v$ can be fed to a neural network to learn node labels. In both cases, a standard loss function, such as cross-entropy loss, can be readily used in backpropagation to learn the weight parameters of the PersonaSAGE model.

3 EXPERIMENTS

In this section, we design experiments to investigate the effectiveness of PersonaSAGE for link prediction (Section 3.2) and node classification (Section 3.3). We then conduct an ablation study in Section 3.4.

Table 1: Dataset Statistics.

Dataset	#Classes	#Nodes	#Edges	#Features
Citeseer	6	3,327	4,732	3703
Cora	7	2,708	5,429	1433
PubMed	3	19,717	44,338	500

3.1 Experimental setup

3.1.1 Data. We consider the following citation network datasets for experiments: Cora, Citeseer, PubMed [25]. These are datasets are a network of scientific publications (nodes), connected via a citation relationship, *i.e.*, one publication cites another. Each node in the datasets is represented by a fixed-length vector derived using a bag-of-words representation of the document. The count of nodes, edges, and features is shown in Table 1. The edge weights, if any, are not considered and we assume the graphs as undirected for sake of simplicity. The usage of these datasets is for two downstream tasks explained in the subsequent sections. Given two publications (an edge), the model could learn to predict if one cites another (link prediction) or given a publication (node), the model could learn to predict their classes as described in Appendix A. These features are normalized (min-max) to obtain a range between 0 and 1. For our approach, we initialize the persona membership vector as the one-hot encoding of cluster labels obtained by applying a clustering algorithm on the normalized features. Since the clustering algorithm usually requires the number of clusters (K) to be supplied as input, we assume the number of personas is equivalent to the number of classes found in the data for sake of simplicity. This could be set to any other value as well.

3.1.2 Baselines. We use the following methods for comparison: ChebNet (Cheb) [6], Graph Convolution Network (GCN) [13], Graph-SAGE [10], Graph Attention Networks (GAT) [27], Topology Adaptive Graph convolutional networks (TAG) [7], EdgeConv (Edge) [29], Simple Graph convolution (SGN) [30]. See Appendinx A for further details. We use a standard implementation of the baselines, as provided by Deep Graph Library (DGL) [28], with default hyperparameters wherever required and keep them constant throughout. Analogous to a simple Neural Network (NN) with a hidden layer, we build a Graph Neural Network (GNN) model with 2 same convolution layers (hidden and output) for each baseline with a ReLU activation function in between. The hidden and final embedding dimensions are kept constant for each baseline model and that of PersonaSAGE for a fair comparison. Though ours is a multi embedding approach, we keep the final embedding size equal to that obtained from baselines by adjusting the output dimension (D) and the number of personas (K) such that their concatenation yields a match.

3.1.3 Training and Evaluation. Given a graph dataset, we hold out 15% of data for testing model performance while another 15% to select the best model learned during training as part of the validation set. All models are trained for 100 epochs using Adam Optimizer [12] with a learning rate of 0.01. The loss function used for optimization is cross-entropy between predictions and ground truth values. All experiments are seeded for reproducibility and

Table 2: Link Prediction Results.

	Citeseer		Cora		PubMed	
Model	**Raw Feat**	**Random Feat**	**Raw Feat**	**Random Feat**	**Raw Feat**	**Random Feat**
Cheb	0.733 ± 0.03	0.563 ± 0.09	0.686 ± 0.06	0.603 ± 0.10	0.803 ± 0.03	0.778 ± 0.01
Edge	0.639 ± 0.03	0.576 ± 0.02	0.515 ± 0.02	0.475 ± 0.04	0.80 ± 0.00	0.732 ± 0.01
GAT	0.734 ± 0.02	0.661 ± 0.02	0.678 ± 0.01	0.661 ± 0.01	0.743 ± 0.01	0.699 ± 0.01
GCN	0.711 ± 0.01	0.682 ± 0.01	0.682 ± 0.01	0.676 ± 0.01	0.791 ± 0.00	0.726 ± 0.01
GraphSAGE	0.816 ± 0.01	0.802 ± 0.01	0.667 ± 0.02	0.618 ± 0.02	0.747 ± 0.01	0.713 ± 0.01
SG	0.705 ± 0.01	0.671 ± 0.01	0.681 ± 0.01	0.674 ± 0.02	0.788 ± 0.01	0.723 ± 0.00
TAG	0.822 ± 0.02	0.798 ± 0.02	0.718 ± 0.00	0.717 ± 0.00	0.829 ± 0.00	0.774 ± 0.01
PersonaSAGE	**0.872 ± 0.02**	**0.907 ± 0.00**	**0.828 ± 0.01**	**0.911 ± 0.01**	**0.950 ± 0.00**	**0.871 ± 0.02**

Table 3: Ablation study link prediction results investigating different variants of our PersonaSAGE framework.

	Citeseer		Cora		PubMed	
Model	**Raw Feat**	**Random Feat**	**Raw Feat**	**Random Feat**	**Raw Feat**	**Random Feat**
PersonaSAGE-KM (K = 1)	0.753 ± 0.00	0.747 ± 0.00	0.740 ± 0.02	0.709 ± 0.01	0.939 ± 0.00	0.843 ± 0.01
PersonaSAGE-KM-max	0.823 ± 0.05	0.863 ± 0.00	0.748 ± 0.04	0.869 ± 0.00	0.880 ± 0.01	0.822 ± 0.00
PersonaSAGE-KM-sum	0.789 ± 0.05	0.885 ± 0.01	0.763 ± 0.03	0.884 ± 0.01	0.881 ± 0.01	0.834 ± 0.01
PersonaSAGE-KM	**0.881 ± 0.02**	**0.916 ± 0.01**	**0.857 ± 0.01**	0.908 ± 0.01	**0.951 ± 0.00**	**0.885 ± 0.00**
PersonaSAGE-Birch	0.872 ± 0.02	0.908 ± 0.00	0.833 ± 0.01	0.909 ± 0.01	**0.951 ± 0.00**	0.871 ± 0.02
PersonaSAGE-Spec	0.867 ± 0.02	0.902 ± 0.02	0.828 ± 0.01	0.878 ± 0.03	-	0.819 ± 0.01
PersonaSAGE	0.872 ± 0.02	0.907 ± 0.00	0.828 ± 0.01	**0.911 ± 0.01**	0.950 ± 0.00	0.871 ± 0.02

run with 5 different seeds resulting in different parameter initializations of weights along with a different data split. The mean and standard deviation of best scores are reported in tables. All other hyperparameters (hidden size, output size, etc.) are kept constant throughout. The embedding size for the first GNN layer is set to 128. For Cora, the embedding size of second layer D is set to 70 for all baseline models while for PersonaSAGE it is 10, since we assume $K = 7$ personas in the data and each persona embedding is of size 10, the concatenated embedding size ($7x10 = 70$) is unchanged. Similarly, for Citeseer we assume $K = 6$ and for PubMed $K = 3$ with $D = 10$. Results with '−' indicate lack of completion. Unless otherwise mentioned, the PersonaSAGE variants use the mean aggregator. Further, PersonaSAGE refers to the default approach that uses the mean aggregator with Ward clustering.

Figure 3: Comparing link prediction test performance (AUC) as a function of the number of epochs used for training on the PubMed dataset. Strikingly, PersonaSAGE achieves significantly better performance across all training epochs.

3.2 Link Prediction

We first investigate the effectiveness of PersonaSAGE for link prediction. In particular, the goal of the link prediction task is to accurately predict the likelihood of the formation of new edges in the graph. Given a pair of nodes, the likelihood can be computed in terms of similarity between the two nodes as the inner product of the two node embeddings followed by a Sigmoid activation. This task measures the capability of the embeddings to capture the structure and topology of the graph. The edge set is randomly split into train and test sets, thus serving as positive samples for each set. An equal size set of remaining unconnected edges is constructed and split in the same proportions as the train and test set. This randomly sampled set serves as negative samples. This setup is adopted in many works, see [1, 9, 24].

In Table 2, we compare PersonaSAGE to a wide variety of other graph representation learning methods. For link prediction, we investigate using both the raw input features that are typically used for node classification (due to their correlation with the class labels of the nodes) and we also study using random features since there is no guarantee that the raw input features would be useful for the link prediction task. Notably, we observe that PersonaSAGE outperforms the other methods across all graph datasets and across both feature settings including the setting where raw input features are used and another where we instead leverage random features (Table 2). Interestingly, PersonaSAGE performs best on Citeseer and Cora when the input features are randomly initialized as opposed to using the standard input features that are typically used for node classification. For instance, we achieve 0.907 on Citeseer when random features are used compared to only 0.872 when using the raw input features as shown in Table 2. PersonaSAGE always

Table 4: Node Classification Results.

Model	Citeseer	Cora	PubMed
Cheb	0.707 ± 0.04	0.763 ± 0.04	0.876 ± 0.0
Edge	0.629 ± 0.08	0.719 ± 0.02	0.862 ± 0.0
GAT	0.665 ± 0.03	0.781 ± 0.02	0.830 ± 0.0
GCN	0.721 ± 0.04	0.851 ± 0.02	0.843 ± 0.01
GraphSAGE	0.714 ± 0.03	0.781 ± 0.04	0.866 ± 0.01
SG	0.709 ± 0.04	0.853 ± 0.02	0.842 ± 0.01
TAG	0.616 ± 0.10	0.833 ± 0.04	0.872 ± 0.0
PersonaSAGE	**0.722 ± 0.02**	**0.859 ± 0.01**	**0.877 ± 0.01**

outperforms the best baseline method with a gain of 6%, 15%, and 14% using raw input features and 13%, 27%, and 12% when random features are used for Citeseer, Cora, and Pubmed, respectively. Nevertheless, in all cases, PersonaSAGE achieves significantly better predictive performance compared to the wide range of baseline methods. These results indicate the advantage and utility of PersonaSAGE over the other baselines for this prediction task.

In Figure 3, we investigate the performance of the different methods as the number of training epochs varies. Strikingly, PersonaSAGE achieves significantly better performance across all number of training epochs. This holds true for training with very few epochs as observed in Figure 3. Furthermore, the best performance of PersonaSAGE is achieved when using only 20 training epochs, and slightly decreases with additional epochs. In contrast, many of the other methods achieve their best performance using a far larger number of epochs compared to PersonaSAGE. For instance, Graph-SAGE achieves an AUC of 0.70 when using 100 training epochs whereas PersonaSAGE achieves a significantly better AUC of 0.90 using only 10 epochs. This is a 10x difference in the number of epochs, while significantly outperforming GraphSAGE in terms of AUC (0.90 compared to 0.70).

In Table 3, we investigate a few different variants from the proposed PersonaSAGE framework. Notably, we observe that these different variants often outperform the previous variant in Table 2. For instance, the PersonaSAGE-KM-max variant that uses K-Means (KM) with the max aggregator performs the best on PubMed. Furthermore, while there is not a single PersonaSAGE variant that always performs best across all graphs and input features, the PersonaSAGE variant that uses k-means with the mean relational aggregator outperforms the other variants most consistently. In particular, this variant always performs best when using the actual input features compared to using random input features (which models the case where such features may not exist for the input graph). As

Table 5: Ablation study results comparing different PersonaSAGE variants for node classification.

Model	Citeseer	Cora	PubMed
PersonaSAGE-KM (K=1)	0.663 ± 0.07	0.837 ± 0.04	0.872 ± 0.01
PersonaSAGE-KM-max	0.720 ± 0.02	0.858 ± 0.01	**0.878 ± 0.0**
PersonaSAGE-KM-sum	0.699 ± 0.03	0.850 ± 0.02	0.728 ± 0.17
PersonaSAGE-KM	0.716 ± 0.03	0.838 ± 0.02	0.874 ± 0.01
PersonaSAGE-Birch	0.722 ± 0.02	**0.861 ± 0.01**	0.875 ± 0.01
PersonaSAGE-Spec	**0.729 ± 0.02**	0.854 ± 0.02	-
PersonaSAGE	0.722 ± 0.02	0.859 ± 0.01	0.877 ± 0.01

Figure 4: Comparing the node classification test accuracy as a function of the number of epochs on the PubMed dataset. Notably, PersonaSAGE outperforms the other methods even when a relatively small number of epochs is used for training.

an aside, we also compare a PersonaSAGE variant that leverages a single embedding per node called PersonaSAGE-KM (K=1), which as shown in Table 3 often performs the worst compared to the other PersonaSAGE variants, indicating the effectiveness of learning multiple embeddings per user. We perform additional ablation study experiments in Section 3.4.

3.3 Node Classification

Now we investigate using PersonaSAGE for the node classification task. In Table 4, we observe that PersonaSAGE always outperforms the other methods across all benchmark datasets. For node classification, we use the raw input features since they are known to be correlated with the class labels we are predicting for the nodes. We also investigate a number of PersonaSAGE variants from the proposed framework for node classification. Results are provided in Table 5. Notably, we observe that different PersonaSAGE variants perform best for the different graphs. In particular, PersonaSAGE-Spec performs best for Citeseer, PersonaSAGE-Birch performs best for Cora, and PersonaSAGE-KM-max performs best for PubMed. Most importantly, in all cases, we find that these results are even better than the PersonaSAGE results in Table 4. Hence, these variants achieve even better performance compared to the other methods.

To investigate the effectiveness of PersonaSAGE as the number of epochs increases, we compare the node classification accuracy as the number of epochs increases for PersonaSAGE along with a variety of state-of-the-art methods. From Figure 4, we observe that PersonaSAGE achieves the best performance compared to the other methods, even when a modest number of epochs is used. In particular, PersonaSAGE always outperforms the other methods when using 40 epochs or more as shown in Figure 4.

3.4 Ablation Study

We investigate the impact of the maximum number of persona embeddings per node K by varying this hyperparameter from $K \in \{1, 2, \ldots, 10\}$. For this experiment, we use the PersonaSAGE variant that leverages K-Means and mean aggregator (PersonaSAGE-KM). We ensure the final embedding size remains constant by reducing the size of the persona embeddings accordingly. For instance, suppose the final embedding size is 50 and the maximum number of embeddings per node is $K = 10$, then the size of each of the $K = 10$

(a)

(b)

Figure 5: Varying the maximum number of persona embeddings K per node in PersonaSAGE for link prediction and node classification.

embeddings per node is set to be of size $D = 5$. Results are reported in Figure 5 for both link prediction and node classification. Overall, we observe that for link prediction, the performance generally increases for Cora and Citeseer while remaining approximately the same for PubMed. In contrast, node classification performance of Citeseer is best when $K = 4$, then decreases as K increases further, whereas the best performance of PersonaSAGE on Cora is $K = 2$ and remains approximately the same for Citeseer. These results indicate the utility of PersonaSAGE and its ability to learn multiple embeddings per node in the graph.

4 CASE STUDY

In this section, we investigate using PersonaSAGE to recommend different types of entities from usage log data of a data management system. Such user interaction logs contain the data queries executed, datasets used, along with the attributes selected in those datasets. By leveraging such usage logs of user activities, we can learn a model for the personalized recommendation of such entities. The goal is to leverage user interactions (from the usage logs) with the platform for building a personalized recommendation engine for users, datasets, queries, etc.

4.1 Data and Graph Construction

The usage log data (from a data management system) consists of a list of queries issued by users interacting with a data management

Figure 6: Personalized recommendation performance (ROC AUC) as a function of the max number of persona embeddings per node (number of clusters) K for the usage log data (from a data management system).

platform. Each query consists of a list of attributes (columns) referenced from their respective datasets. We parse the query to yield its referenced attributes and datasets using SQL Parser (Python Library) and map it with the corresponding user along with the issued query. We consider a heterogeneous graph with 4 types of nodes including users, queries, datasets, and attributes. We form an (undirected) edge between, for instance, a user and query node if the user has initiated that query. Similarly, we form edges between users and datasets, users and attributes based on interactions.

4.2 Experimental Setup

We investigate the persona embeddings for this large and sparse heterogeneous graph in the context of link prediction. The task designed is exactly same as stated in Section 3.2 just that the whole graph is treated as homogeneous and the negative samples are drawn uniformly across the graph. This set would contain unconnected node pairs from all combinations such as user-user, user-query, query-dataset, and so on. Further, for conducting the experiment, we use a two-layer GNN architecture with the same hyperparameters as described in Section 3.1.3 unless explicitly stated. Since we do not have input features, we randomly initialize all of them with embedding size 100. The hidden size is kept 128 and output dimension D is varied over $\{20, 30, 50, 70, 100\}$.

Table 6: Results for Personalized Query, Attribute, and Dataset Recommendation (AUC).

	Query	*Attribute*	*Dataset*
Cheb	0.500	0.500	0.500
Edge	0.668	0.684	0.548
GAT	0.593	0.495	0.689
GCN	0.700	0.730	0.635
GraphSAGE	0.723	0.716	0.660
SG	0.683	0.655	0.604
TAG	0.722	0.707	0.579
PersonaSAGE	**0.862**	**0.845**	**0.839**

4.3 Results

In this section, we investigate using PersonaSAGE to recommend any arbitrary type of entity to a user in a personalized fashion. In particular, results for three different personalized recommendation tasks including QUERY, ATTRIBUTE, and DATASET RECOMMENDATION are provided in Table 6. Overall, we observe that PersonaSAGE outperforms the other methods across all three personalized recommendation tasks as shown in Table 6. The improvement compared to the next best method across the three different recommendation tasks is significant. Notably, compared to the best performing baseline method, PersonaSAGE achieves a gain of 19.2%, 15.7%, and 21.7% in AUC for query, attribute, and dataset recommendation tasks, respectively. Furthermore, while PersonaSAGE always outperforms the other methods across all three recommendation tasks, there is not a single baseline method that always performs best across all three recommendation tasks. For instance, GraphSAGE is the best performing baseline method for query recommendation, whereas GAT outperforms the other baseline methods for the dataset recommendation task. In contrast, PersonaSAGE performs best independent of the personalized recommendation task.

To further understand the approach, we now investigate a slightly different experimental setup where the held out set for a specific user can contain a mix of different entity types, including datasets, attributes, and queries. Results are provided in Table 7. Notably, we observe that PersonaSAGE outperforms the other methods across all embedding sizes $D \in \{20, 30, 50, 70, 100\}$ as shown in Table 7. Furthermore, PersonaSAGE achieved a mean gain of 12.37% in AUC over the best performing baseline when $D = 100$ is used. In Figure 6, we report performance as we vary the maximum number of persona embeddings per node for $D \in \{20, 50, 100\}$. Notably, these results indicate that PersonaSAGE is well-suited to predict any arbitrary link type, as we do not restrict the held-out links to a specific type in this experiment. These results demonstrate the overall effectiveness of our approach for prediction in such complex heterogeneous graphs.

Table 7: Results for recommendation of any arbitrary link-type (AUC). These results include recommendation of a variety of different link-types such as user-query links, user-dataset links, and user-attribute links.

	$D = 20$	$D = 30$	$D = 50$	$D = 70$	$D = 100$
GraphSAGE	0.770	0.770	0.778	0.755	0.778
GCN	0.828	0.803	0.72	0.837	0.822
TAG	0.787	0.848	0.854	0.809	0.824
GAT	0.725	0.75	0.687	0.731	0.729
PersonaSAGE	**0.857**	**0.861**	**0.907**	**0.909**	**0.926**

5 RELATED WORK

Recent advances in approaches for network embeddings have received a lot of attention due to their effectiveness in capturing both local and global contexts of the networks. These approaches for learning network representations can be grouped based on various criteria. Some of the graph convolution methods include GCN [13] and Chebnet [6], which present a model based on spectral convolution graph operations, extending convolutions from Euclidean

grids to graphs. The forward propagation is computed by stacking layers of the product of a normalized graph Laplacian, the data, and the model parameters, which is the result of approximating Chebyshev polynomials. These papers utilize techniques to optimize computation, and report results across distinct datasets. In [6] results are presented for the image classification dataset, and text categorization dataset. [13] showcases results for a 2-layer GCN node classifier on citation network datasets.

Apart from spectral-based approaches [11], there are various spatial-based convolution approaches [10, 16] for effectively leveraging the spatial contexts. For instance, Graph Attention Networks (GAT) [27] implements multi-head attention on the local neighborhood of a node and demonstrates results on citation networks, and a protein-protein interaction dataset. Edgeconv [29] builds models for the tasks of classification, part segmentation, and semantic segmentation through edge convolutions on point cloud data. SGN [30] simplifies the architecture of GCN by removing non-linearities, and presents results on citation and social networks. A survey of several graph neural network architectures is presented in [31].

Another line of works learn node representations using simulated random walks on the graph. Works by [2, 9, 20, 26], which compute (single) node embeddings based on random walks are transductive approaches in nature. They require a retraining procedure every time a new node is encountered. There have been some inspiring recent advances in the direction of capturing the polysemous behavior of nodes in a graph. PolyDeepwalk [14] propose a multi-embedding approach to model multiple *facets* of nodes and highlight the effectiveness of a multi-embedding representation than a single vector. Asp2vec [19] propose an unsupervised end-to-end pipeline to compute multiple *aspects* of nodes based on their local context. However, these approaches follow the same drawback of using random walks which render them less effective in scenarios for growing networks. PinnerSage [18] is a bipartite clustering approach that assigns users multiple embeddings. However, the embeddings assigned to the users are not unique, that is, two or more users can share the exact same embedding vector, which makes the multiple embeddings not as useful. Other work introduced polysemy embeddings [8] but simplify the problem by learning embeddings for each facet (or graph). This is essentially equivalent to heterogeneous embedding methods that learn embedding vectors for each context/mode.

All these works inspire our thinking for developing a flexible, general, and yet performant framework for capturing polysemous nature (persona) in a large-scale network. We note a differentiation of our work from these multi-embedding approaches is that our approach learns a *set* of embeddings where the *number* of (persona) embeddings may be different for different nodes which are learned automatically from local and global structures.

6 CONCLUSION

We generally come across scenarios where a single entity performs in a polysemous way, such as an individual's behavior in the context of a sports player, father, etc. We represent this nature as multiple personas of the same entity but in different contexts. In this work, we proposed a novel approach called PersonaSAGE that learns multiple persona embeddings per node along with their persona

weights. Notably, the set of embeddings learned for every node may differ depending on the structural context around a given node. We demonstrated the effectiveness of PersonaSAGE for a wide variety of application tasks including node classification and link prediction. Finally, we also conducted a case study where we investigated using PersonaSAGE for recommending queries, attributes, and datasets to users. In all cases, PersonaSAGE significantly outperformed the other methods across all graphs and application tasks.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, and Rami Al-Rfou. 2017. Learning edge representations via low-rank asymmetric projections. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1787–1796.

[2] Nesreen K Ahmed, Ryan A Rossi, John Boaz Lee, Theodore L Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. 2019. role2vec: Role-based network embeddings. In *Proc. DLG KDD*. 1–7.

[3] Nesreen K Ahmed, Ryan A Rossi, Theodore L Willke, and Rong Zhou. 2017. Edge role discovery via higher-order structures. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 291–303.

[4] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. 2019. *On the Equivalence between Graph Isomorphism Testing and Function Approximation with GNNs*. Curran Associates Inc., Red Hook, NY, USA.

[5] Clemens Damke, Vitalik Melnikov, and Eyke Hüllermeier. 2020. A Novel Higher-order Weisfeiler-Lehman Graph Convolution. *CoRR* abs/2007.00346 (2020). arXiv:2007.00346

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375* (2016).

[7] Jian Du, Shanghang Zhang, Guanhang Wu, José MF Moura, and Soummya Kar. 2017. Topology adaptive graph convolutional networks. *arXiv preprint arXiv:1710.10370* (2017).

[8] Alessandro Epasto and Bryan Perozzi. 2019. Is a single embedding enough? learning node representations that capture multiple social contexts. In *The World Wide Web Conference*. 394–404.

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

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 1025–1035.

[11] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv:1506.05163* (2015).

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

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

[14] Ninghao Liu, Qiaoyu Tan, Yuening Li, Hongxia Yang, Jingren Zhou, and Xia Hu. 2019. Is a single vector enough? exploring node polysemy for network embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 932–940.

[15] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2019. Provably Powerful Graph Networks. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc.

[16] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5115–5124.

[17] Christopher Morris, Gaurav Rattan, and Petra Mutzel. 2020. Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 21824–21840.

[18] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. 2020. PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2311–2320.

[19] Chanyoung Park, Carl Yang, Qi Zhu, Donghyun Kim, Hwanjo Yu, and Jiawei Han. 2020. Unsupervised differentiable multi-aspect network embedding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1435–1445.

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

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

[22] Ryan A Rossi and Nesreen K Ahmed. 2014. Role discovery in networks. *IEEE Transactions on Knowledge and Data Engineering* 27, 4 (2014), 1112–1131.

[23] Ryan A Rossi, Di Jin, Sungchul Kim, Nesreen K Ahmed, Danai Koutra, and John Boaz Lee. 2020. On proximity and structural role-based embeddings in networks: Misconceptions, techniques, and applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 14, 5 (2020), 1–37.

[24] Ryan A Rossi, Rong Zhou, and Nesreen K Ahmed. 2018. Deep inductive graph representation learning. *IEEE Transactions on Knowledge and Data Engineering* 32, 3 (2018), 438–452.

[25] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[26] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*. 1067–1077.

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

[28] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv:1909.01315* (2019).

[29] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)* 38, 5 (2019), 1–12.

[30] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

[31] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.

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

APPENDIX

A EXPERIMENTAL SETUP DETAILS

A.1 Data

The datasets used in the experiments are described below:

- Each node (a publication) in the Citeseer dataset is labeled as one of the six domains: Agents, Artificial Intelligence, Databases, Information Retrieval, Machine Learning, and Human Computer Interaction.
- The Cora dataset consists of publications (nodes) belonging to one of the following classes: Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, and Theory.
- The PubMed dataset comprises of publications (nodes) classified as one of the 3 categories of Diabetes (Experimental, Type 1 and Type 2).

A.2 Baselines

The baselines used in experiments are described below:

- ChebNet (Cheb) [6]: The Chebyshev spectral filter, derived from the Spectral CNN, uses an efficient pooling strategy to capture the features in a localized region.
- Graph Convolution Network (GCN) [13]: This kind of convolution operation introduced a first-order approximation of the ChebNet in the space of spectral filters. It can be considered as combining information from neighbors with a self-loop on every node.

- GraphSAGE [10]: The authors proposed an inductive approach to aggregate neighborhood information with the node's current information and apply linear transformations followed by non-linearity (*e.g.*, Sigmoid).
- Graph Attention Networks (GAT) [27]: Unlike GraphSAGE, this approach weighs the neighborhood information by learning the relative edge weights using an attention network.
- Topology Adaptive Graph convolutional networks (TAG) [7]: The authors propose a topology-aware spectral filter for graph convolution operation by extending the neighborhood definition used in GCN to capture higher-order neighborhood information.
- EdgeConv (Edge) [29]: Inspired from PointNet [21], the authors propose a differential and pluggable convolution operator module designed to capture the topological and geometric features of point clouds.
- Simple Graph convolution (SGN) [30]: The authors propose a rather simplifying computation for graph convolution operation by reducing the non-linearities in order to scale to large graphs and without drastically reducing the performance.

B PROPERTIES OF PERSONASAGE

In this subsection, we note some salient aspects of the PersonaSAGE algorithm by remarking on the variation in the number of persona embeddings per node.

Variable number of persona embeddings: For an arbitrary graph, the number of embeddings per node will depend on the clustering algorithm and topology of the graph. For instance, consider a graph with n nodes and $k = n$ clusters, where start and end nodes have degree 1 and interior nodes have degree 2 and are connected to the previous and next nodes. In this case, if the number of GNN layers is less than $\log_3 n$, then there will be a variable number of persona embeddings for each node, which can be obtained by analyzing the aggregation of the membership vectors across the layers.

Weisfeiler-Lehman test: The GraphSAGE algorithm is related to the Weisfeiler-Lehman test. This is a graph isomorphism test, which maintains node labels or hashes by aggregating labels of neighboring nodes, and across iterations. The 1-WL test, which is also called the vertex color refinement algorithm, aggregates information for a node across its node neighbors. The k-dimensional Weisfeiler-Lehman test defines the node neighborhood to be n many k-tuples of nodes, whereas the k-FWL version considers k number of k-tuples. It is known that for each $k \geq 2$ there is a pair of non-isomorphic graphs distinguishable by $(k + 1)$-WL but not by k-WL. Higher-order variants of the WL-test form the basis of recent work on graph neural networks [4, 5, 15, 17]. Though there are similarities with the WL-test, we do not consider the graph isomorphism problem in this paper further. In comparison to the prior art, k-dimensional cluster membership provides a distinct definition of a node's neighborhood, which determines the aggregation logic in the PersonaSAGE algorithm.