Graph Attention Auto-Encoders

Amin Salehi
Computer Science and Engineering
Arizona State University
Tempe, Arizona
asalehi1@asu.edu

Hasan Davulcu
Computer Science and Engineering
Arizona State University
Tempe, Arizona
hdavulcu@asu.edu

***Abstract*—Auto-encoders have emerged as a successful framework for unsupervised learning. However, conventional auto-encoders are incapable of utilizing explicit relations in structured data. To take advantage of relations in graph-structured data, several graph auto-encoders have recently been proposed, but they neglect to reconstruct either the graph structure or node attributes. In this paper, we present the graph attention auto-encoder (GATE), a neural network architecture for unsupervised representation learning on graph-structured data. Our architecture is able to reconstruct graph-structured inputs, including both node attributes and the graph structure, through stacked encoder/decoder layers equipped with self-attention mechanisms. In the encoder, by considering node attributes as initial node representations, each layer generates new representations of nodes by attending over their neighbors' representations. In the decoder, we attempt to reverse the encoding process to reconstruct node attributes. Moreover, node representations are regularized to reconstruct the graph structure. Our proposed architecture does not need to know the graph structure upfront, and thus it can be applied to inductive learning. Our experiments demonstrate competitive performance on several node classification benchmark datasets for transductive and inductive tasks, even exceeding the performance of supervised learning baselines in most cases.**

I. Introduction

Low-dimensional vector representations of nodes in graphs have demonstrated their utility in a broad range of machine learning tasks such as node classification [1], recommender systems [2], community detection [3], graph visualization [4], link prediction [5] and relational modeling [6]. Accordingly, there has been a surge of research to learn better node representations. However, most of the proposed methods [1], [4], [7]–[17] only utilize the graph structure while nodes in real-world graphs usually come with a rich set of attributes (i.e. features). Typical examples are users in social networks, scientific articles in citation networks, protein molecules in biological networks and web pages on the Internet.

Significant efforts have been made [18]–[24] to utilize node attributes for graph representation learning. Nevertheless, the most successful methods, notably graph convolutional networks [22] and graph attention networks [23], depend on label information, which is not available in many real-world applications. Moreover, the process of annotating data suffers from many limitations, such as annotators' subjectivity, reproducibility, and consistency.

To avoid the challenges of annotating data, several unsupervised graph embedding methods [25]–[31] have been proposed, but these methods suffer from at least one of the three following problems. First, despite utilizing node features, some of these models [25], [27], [29] heavily depend on the graph structure. This hinders their capability to fully exploit node features. Second, many [29]–[31] are not capable of inductive learning, which is crucial to encounter unseen nodes (e.g., new users in social networks, recently published scientific articles and new web pages on the Internet). Third, even though some efforts have been made [24], [28] to address inductive learning tasks, they are not unified architectures for both transductive and inductive tasks.

Auto-encoders have recently become popular for unsupervised learning due to their ability to capture complex relationships between input's attributes through stacked non-linear layers [32], [33]. However, conventional auto-encoders are not able to take advantage of explicit relations in structured data. To utilize relations in graph-structured data, several graph auto-encoders [25], [27], [34] have been proposed. Although the encoders in these models fully utilize graph-structured inputs, the decoders neglect to reconstruct either the graph structure or node attributes.

Another successful neural network paradigm is the attention mechanism [35] proven useful in tackling many machine learning tasks [36]–[38], particularly sequence-based tasks [39]–[42]. The state-of-the-art attention mechanism is self-attention, which computes the representation of the input by focusing on its most relevant parts. Self-attention has been successfully applied to a variety of tasks including machine translation [39], video classification [38] and question answering [41]. Nonetheless, the majority of these efforts target supervised learning tasks, and few efforts [43], [44] are made to tackle unsupervised learning tasks. In graph representation learning, to our knowledge, the only proposed attention-based method uses supervised learning [23].

In this work, we present a novel graph auto-encoder to learn node representations within graph-structured data (i.e., attributed graphs) in an *unsupervised manner*. Our auto-encoder takes in and reconstructs node features by utilizing the graph structure through stacked encoder/decoder layers. In the encoder, node attributes are fed into stacked layers to generate node representations. By considering node features as initial node representations, each encoder layer generates new representations of nodes by utilizing neighbors' representations according to their relevance, which is determined by a graph

2375-0197/20/$31.00 ©2020 IEEE
DOI 10.1109/ICTAI50040.2020.00154

attention mechanism. In the decoder, we aim to reverse the entire encoding process to reconstruct node attributes. To this end, each decoder layer attempts to reverse the process of its corresponding encoder layer. Moreover, node representations are regularized to reconstruct the graph structure. To our knowledge, no auto-encoder is capable of reconstructing both node attributes and the graph structure. Our architecture can also be applied to inductive learning tasks since it doesn't need to know the graph structure upfront.

Our key contributions are summarized as follows:

- We propose a novel graph auto-encoder for unsupervised representation learning on graph-structured data by reconstructing both node features and the graph structure.
- We utilize self-attention for unsupervised attributed graph representation learning.
- We present a unified neural architecture capable of both transductive and inductive learning.

II. Related Work

A. Graph Representation Learning

Most of the graph embedding methods fall into one of the following three categories: factorization based, random walk based, and auto-encoder based approaches. Factorization based approaches are inspired by matrix factorization methods, which assume that the data lies in a low dimensional manifold. Laplacian Eigenmaps [7] and LPP [8] rely on eigendecomposition to preserve the local manifold structure. Due to expensive eigendecomposition operations, these methods face difficulty to tackle large-scale graphs. To alleviate this problem, several techniques—notably the Graph Factorization (GF) [9], GraRep [10] and HOPE [11]—have been proposed. These methods differ mainly in their node similarity calculation. The graph factorization computes node similarity based on the first-order proximities directly extracted from the adjacency matrix. To capture more accurate node similarity, GraRep and HOPE utilize the high-order proximities obtained from different powers of the adjacency matrix and similarity measures (i.e., cosine similarity) respectively. Random walk based approaches assume a pair of nodes to be similar if they are close in simulated random walks over the graph. Therefore, node similarity is stochastically computed in contrast to the deterministic approach used by factorization based methods. DeepWalk [4] and node2vec [1] are the most successful methods in this category and differ primarily in their random walk generation. DeepWalk simulates uniform random walks while node2vec relies on a biased random walk generation. Preozzi et al. [12] extend DeepWalk to encode multiscale node relationships in the graph. In contrast to DeepWalk and node2vec, which embed nodes in the Euclidean space, Chamberlan et al. [13] utilize the hyperbolic space.

Factorization and random walk based approaches adopt shallow models, which are incapable of capturing complex graph structures. To solve this problem, auto-encoder based approaches are proposed to capture non-linear graph structures by using deep neural networks. Tian et al. [14] present a stacked sparse auto-encoder to embed nodes by reconstructing the adjacency matrix. Moreover, Wang et al. [15] propose a stacked auto-encoder, which reconstructs the second-order proximities by using the first-order proximities as a regularization. Cao et al. [16] use stacked denoising auto-encoder to reconstruct the pointwise mutual information matrix.

B. Attributed Graph Representation Learning

The aforementioned graph embedding methods only utilize the graph structure to learn node representations. However, nodes in real-world graphs usually come with a rich set of attributes. To take advantage of node features, many attributed graph embedding methods have been proposed, which fall into two main categories: supervised and unsupervised approaches.

Supervised attributed graph embedding approaches embed nodes by utilizing label information. For example, Huang et al. [18] propose a supervised method leveraging spectral techniques to project the adjacency matrix, node feature matrix, and node label matrix into a common vector space. Hamilton et al. [24] present four variants of GraphSAGE, a framework to compute node embeddings in an inductive manner. Many approaches address graphs with partial label information. For example, Graph Convolution Network (GCN) [22] incorporates spectral convolutions into neural networks. Graph Attention Network (GAT) [23] utilizes an attention mechanism to determine the influence of neighboring nodes in final node representations.

The unsupervised attributed graph embedding methods address the lack of label information, which exists in many real-world applications. Yang et al. [31] and Huang et al. [30] propose matrix factorization methods to combine the graph structure and node attributes. Moreover, Kipf et al. [25] propose two graph auto-encoders utilizing graph convolution networks. Pan et al. [27] also introduce a graph-encoder based on an adversarial approach. For graph clustering, Wang et al. [34] present a graph auto-encoder, which is able to reconstruct node features. However, these auto-encoders reconstruct either the graph structure or node attributes instead of both. To alleviate this limitation, Gao et al. [29] propose a framework consisting of two conventional auto-encoders, which reconstruct the graph structure and node attributes separately. These two auto-encoders are regularized in a way that their learned representations of neighboring nodes are similar. However, their framework does not fully leverage the graph structure due to the incapability of conventional auto-encoders in utilizing explicit relations in structured data. Most of the aforementioned unsupervised methods are not designed for inductive learning, which is crucial to encounter unseen nodes. Velivckovic et al. [28] and Hamilton et al. [24] propose unsupervised models for tackling inductive tasks, but their models are not unified frameworks for both transductive and inductive tasks.

III. Problem Statement

In this section, we present the notations used in the paper and formally define the problem of unsupervised node

representation learning on graph-structured data. We use bold upper-case letters for matrices (e.g., $\mathbf{X}$), bold lowercase letters for vectors (e.g., $\mathbf{x}$), and calligraphic fonts for sets (e.g., $\mathcal{N}$). Moreover, we represent the transpose of a matrix $\mathbf{X}$ as $\mathbf{X}^T$. The i^{th} element of vector $\mathbf{x}$ is denoted by $\mathbf{x}_i$. $\mathbf{X}_{ij}$ denotes the entry of matrix $\mathbf{X}$ at the i^{th} row and the j^{th} column. Table I summarizes the main notations used in the paper.

In the attributed graph representation learning setup, we are provided with the node feature matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$, where N is the number of nodes in the graph and $\mathbf{x}_i \in \mathbb{R}^F$ corresponds to the i^{th} column of matrix $\mathbf{X}$, denoting the features of node i. We are also given the adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, representing the relations between nodes. Even though the matrix $\mathbf{A}$ may consist of real numbers, in our experiments, we assume the graph is unweighted and includes self-loops, i.e., $\mathbf{A}_{ij} = 1$ if there is an edge between node i and node j in the graph or i equals j, and $\mathbf{A}_{ij} = 0$ otherwise. Given the node feature matrix $\mathbf{X}$ and the adjacency matrix $\mathbf{A}$, our objective is to learn the node representation matrix $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_N]$, where $\mathbf{h}_i \in \mathbb{R}^D$ corresponds to the i^{th} column of matrix $\mathbf{H}$, denoting the representation of node i.

IV. Architecture

In this section, we illustrate the architecture of the graph attention auto-encoder. First, we present the encoder and decoder to show how our auto-encoder reconstructs node features using the graph structure. Then, we describe the proposed loss function, which learns node representations by minimizing the reconstruction loss of node features and the graph structure. In the end, we present the matrix formulation of GATE, as well as its time and space complexities.

A. Encoder

The encoder in our architecture takes node features and generates node representations by using the graph structure through stacked layers. We use multiple encoder layers for two reasons. First, more layers make our model deeper, and hence increasing the learning capability. Second, they propagate node representations through the graph structure, resulting in richer node embeddings.

Each encoder layer generates new representations of nodes by utilizing their neighbors' representations according to their relevance. To determine the relevance between nodes and their neighbors, we use a self-attention mechanism with shared parameters among nodes, following the work of Velickovic et al. [23]. In the k^{th} encoder layer, the relevance of a neighboring node j to node i is computed as follows:

$$e_{ij}^{(k)} = \text{Sigmoid}\left(\mathbf{v}_s^{(k)^T}\sigma\left(\mathbf{W}^{(k)}\mathbf{h}_i^{(k-1)}\right) + \mathbf{v}_r^{(k)^T}\sigma\left(\mathbf{W}^{(k)}\mathbf{h}_j^{(k-1)}\right)\right) \tag{1}$$

where $\mathbf{W}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$, $\mathbf{v}_s^{(k)} \in \mathbb{R}^{d^{(k)}}$, and $\mathbf{v}_r^{(k)} \in \mathbb{R}^{d^{(k)}}$ are the trainable parameters of the k^{th} encoder layer, σ denotes the activation function and Sigmoid represents the sigmoid function (i.e., Sigmoid $(x) = 1/(1 + \exp^{-x})$).

TABLE I: The main notations used in the paper.

Notations	Definitions
N	The number of nodes in the graph
E	The number of edges in the graph
L	The number of layers
$d^{(k)}$	The number of node representation dimensions in the k^{th} encoder/decoder layer
F	The number of node features ($d^{(0)} = F$)
P	The number of iterations (i.e., epochs)
$\mathbf{A} \in \mathbb{R}^{N \times N}$	The adjacency matrix
$\mathbf{H}^{(k)} \in \mathbb{R}^{d^{(k)} \times N}$	The node representation matrix generated by the k^{th} encoder layer
$\widehat{\mathbf{H}}^{(k)} \in \mathbb{R}^{d^{(k)} \times N}$	The node representation matrix reconstructed by the k^{th} decoder layer
$\mathbf{H} \in \mathbb{R}^{d^{(L)} \times N}$	The node representation matrix ($\mathbf{H} = \mathbf{H}^{(L)} = \widehat{\mathbf{H}}^{(L)}$)
$\mathbf{X} \in \mathbb{R}^{F \times N}$	The node feature matrix ($\mathbf{H}^{(0)} = \mathbf{X}$)
$\widehat{\mathbf{X}} \in \mathbb{R}^{F \times N}$	The reconstructed node feature matrix ($\widehat{\mathbf{X}} = \widehat{\mathbf{H}}^{(0)}$)
$\mathbf{C}^{(k)} \in \mathbb{R}^{N \times N}$	The attention matrix in the k^{th} encoder layer
$\widehat{\mathbf{C}}^{(k)} \in \mathbb{R}^{N \times N}$	The attention matrix in the k^{th} decoder layer
$\mathbf{h}_i{}^{(k)} \in \mathbb{R}^{d^{(k)}}$	The representation of node i generated by the k^{th} encoder layer
$\widehat{\mathbf{h}}_i^{(k)} \in \mathbb{R}^{d^{(k)}}$	The representation of node i reconstructed by the k^{th} decoder layer
$\mathbf{h}_i \in \mathbb{R}^{d^{(L)}}$	The representation of node i ($\mathbf{h}_i = \mathbf{h}_i{}^{(L)} = \widehat{\mathbf{h}}_i^{(L)}$)
$\mathbf{x}_i \in \mathbb{R}^F$	The features of node i ($\mathbf{h}_i^{(0)} = \mathbf{x}_i$)
$\widehat{\mathbf{x}}_i \in \mathbb{R}^F$	The reconstructed features of node i ($\widehat{\mathbf{x}}_i = \widehat{\mathbf{h}}_i^{(0)}$)
$\alpha_{ij}^{(k)}$	The attention coefficient indicating the relative relevance of neighboring node j to node i in the k^{th} encoder layer
$\widehat{\alpha}_{ij}^{(k)}$	The attention coefficient indicating the relative relevance of neighboring node j to node i in the k^{th} decoder layer
$\mathcal{N}_i$	The neighborhood of node i, including itself

To make the relevance coefficients of node i's neighbors comparable, we normalize them by using the softmax function as follows:

$$\alpha_{ij}^{(k)} = \frac{\exp\left(e_{ij}^{(k)}\right)}{\sum_{l \in \mathcal{N}_i} \exp\left(e_{il}^{(k)}\right)} \tag{2}$$

where $\mathcal{N}_i$ represents the neighborhood of node i (i.e., a set of nodes connected to node i according to the adjacency matrix $\mathbf{A}$, including node i itself).

By considering node features as initial node representations (i.e., $\mathbf{h}_i{}^{(0)} = \mathbf{x}_i$, $\forall i \in \{1, 2, .., N\}$), the k^{th} encoder layer generates the representation of node i in layer k as follows:

$$\mathbf{h}_i^{(k)} = \sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(k)} \sigma\left(\mathbf{W}^{(k)}\mathbf{h}_j^{(k-1)}\right) \tag{3}$$

After applying L encoder layers, we consider the output of the last layer as the final node representations (i.e., $\mathbf{h}_i = \mathbf{h}_i^{(L)}$, $\forall i \in \{1, 2, .., N\}$).

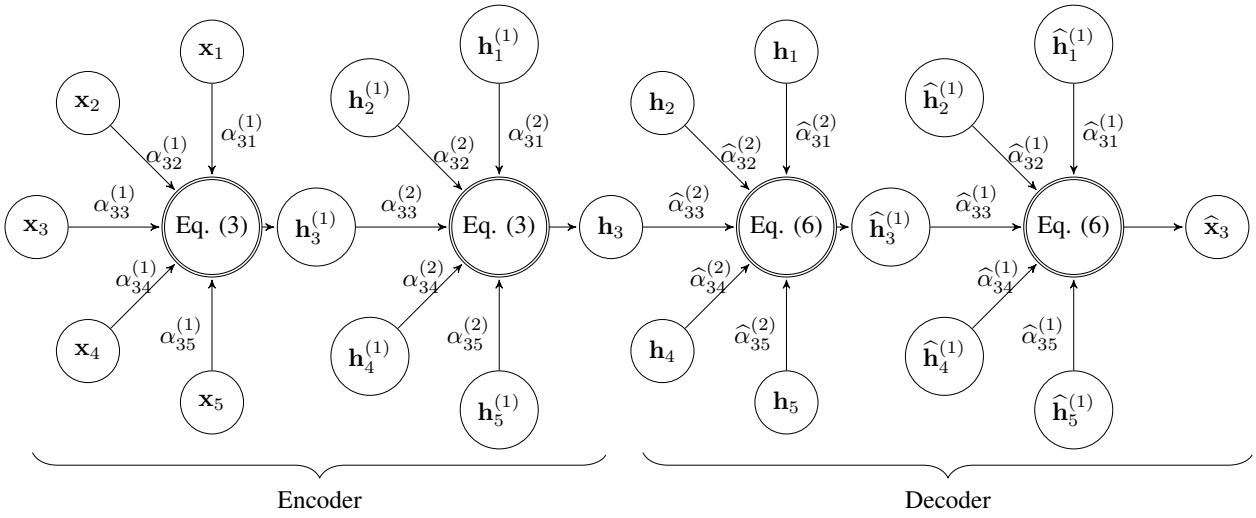


Fig. 1: The illustration of reconstructing the features of node 3, with neighborhood $\mathcal{N}_3 = \{1, 2, 3, 4, 5\}$, using the graph attention auto-encoder with 2 layers; we note that $\mathbf{h}_i^{(0)} = \mathbf{x}_i$, $\mathbf{h}_i = \mathbf{h}_i^{(2)} = \widehat{\mathbf{h}}_i^{(2)}$, and $\widehat{\mathbf{x}}_i = \widehat{\mathbf{h}}_i^{(0)}$, $\forall i \in \{1, 2, .., N\}$.

B. Decoder

Our encoder is reminiscent of graph attention networks [23], which use supervised learning to embed nodes. Our main contribution is reversing the encoding process in order to learn node representations without any supervision. To this end, we use a decoder with the same number of layers as the encoder. Each decoder layer attempts to reverse the process of its corresponding encoder layer. In other words, each decoder layer reconstructs the representations of nodes by utilizing the representations of their neighbors according to their relevance. The normalized relevance (i.e., attention coefficient) of a neighboring node j to node i in the k^{th} decoder layer is computed as follows:

$$\widehat{\alpha}_{ij}^{(k)} = \frac{\exp\left(\widehat{e}_{ij}^{(k)}\right)}{\sum_{l \in \mathcal{N}_i} \exp\left(\widehat{e}_{il}^{(k)}\right)} \tag{4}$$

$$\widehat{e}_{ij}^{(k)} = \text{Sigmoid}\left(\widehat{\mathbf{v}}_s^{(k)^T} \sigma\left(\widehat{\mathbf{W}}^{(k)} \widehat{\mathbf{h}}_i^{(k)}\right) + \widehat{\mathbf{v}}_r^{(k)^T} \sigma\left(\widehat{\mathbf{W}}^{(k)} \widehat{\mathbf{h}}_j^{(k)}\right)\right) \tag{5}$$

where $\widehat{\mathbf{W}}^{(k)} \in \mathbb{R}^{d^{(k-1)} \times d^{(k)}}$, $\widehat{\mathbf{v}}_s^{(k)} \in \mathbb{R}^{d^{(k-1)}}$, and $\widehat{\mathbf{v}}_r^{(k)} \in \mathbb{R}^{d^{(k-1)}}$ are the trainable parameters of the k^{th} decoder layer.

By feeding the encoder's output to the decoder (i.e., $\widehat{\mathbf{h}}_i^{(L)} = \mathbf{h}_i^{(L)}$, $\forall i \in \{1, 2, .., N\}$), the k^{th} decoder layer reconstructs the representation of node i in layer $k-1$ as follows:

$$\widehat{\mathbf{h}}_i^{(k-1)} = \sum_{j \in \mathcal{N}_i} \widehat{\alpha}_{ij}^{(k)} \sigma\left(\widehat{\mathbf{W}}^{(k)} \widehat{\mathbf{h}}_j^{(k)}\right) \tag{6}$$

After applying L decoder layers, we consider the output of the last layer as the reconstructed node features (i.e., $\widehat{\mathbf{x}}_i = \widehat{\mathbf{h}}_i^{(0)}$, $\forall i \in \{1, 2, .., N\}$). Figure 1 illustrates the process of reconstructing node features in GATE through an example.

C. Loss Function

Graph-structured data include node features and the graph structure, and both should be encoded by high-quality node representations. Therefore, we first minimize the reconstruction loss of node features as follows:

$$\sum_{i=1}^{N} ||\mathbf{x}_i - \widehat{\mathbf{x}}_i||_2 \tag{7}$$

The absence of an edge between two nodes in the graph does not necessarily imply dissimilarity due to the possibility of feature similarity. Thus, we minimize the reconstruction loss of the graph structure by making the representations of neighboring nodes similar. We accomplish this by minimizing the following equation:

$$-\sum_{i=1}^{N} \sum_{j \in \mathcal{N}_i} \log\left(\frac{1}{1 + \exp(-\mathbf{h}_i^T \mathbf{h}_j)}\right) \tag{8}$$

By merging Eq. (7) and Eq. (8), we minimize the reconstruction loss of node features and the graph structure as follows:

$$\text{Loss} = \sum_{i=1}^{N} ||\mathbf{x}_i - \widehat{\mathbf{x}}_i||_2 - \lambda \sum_{j \in \mathcal{N}_i} \log\left(\frac{1}{1 + \exp(-\mathbf{h}_i^T \mathbf{h}_j)}\right) \tag{9}$$

where λ controls the contribution of the graph structure reconstruction loss.

D. Complexity

Our proposed auto-encoder is highly efficient because the operations involved in the graph attention mechanisms can be parallelized across edges, and the rest of the operations in the encoder and decoder can be parallelized across nodes.

TABLE II: The statistics of the benchmark datasets.

Dataset	Nodes	Edges	Features	Classes	Train/Val/Test Nodes
Cora	2,708	5,429	1,433	7	140/500/1,000
Citeseer	3,327	4,732	3,703	6	120/500/1,000
Pubmed	19,717	44,338	500	3	60/500/1,000

Theoretically, the time complexity of our architecture for one iteration can be expressed as follows:

$$O(NFD + ED) \tag{10}$$

where N and E are respectively the number of nodes and edges in the graph, F is the number of node features and D is the maximum $d^{(k)}$ in all layers (i.e., $D = \max_{k\in\{1,2,..,L\}} d^{(k)}$). By taking advantage of sparse matrix operations, the space complexity of our auto-encoder is linear in terms of the number of nodes and edges.

V. Evaluation

In this section, we quantitatively and qualitatively evaluate the proposed GATE architecture using several benchmark datasets. Section V-A and V-B respectively describe the datasets, baselines, and experimental setup used in our experiments. In Section V-C, we quantitatively evaluate the efficacy of our architecture. Section V-D investigates the impact of the three main components used in our proposed architecture, namely the self-attention mechanism, graph structure reconstruction, and node feature reconstruction. Finally, we investigate the quality of the node representations learned by GATE in Section V-E.

A. Datasets

For transductive tasks, we use three benchmark datasets—Cora, Citeseer and Pubmed [45]—widely used to evaluate attributed graph embedding methods. In all datasets, each node belongs to one class. We follow the experimental setup of Yang et al. [19], where 20 nodes per class are used for training. In the transductive setup, we have access to the graph structure and nodes' feature vectors during training. We evaluate the predictive performance of methods on 1000 test nodes; 500 additional nodes are used for validation of supervised methods. The statistics of the datasets are presented in Table II.

For inductive tasks, we also use the same datasets and experimental setup in order to evaluate the generalization power of different methods to unseen nodes by comparing the difference between their performance in transductive and inductive tasks for the same dataset. As required by inductive learning, any information related to (unseen) test nodes, including features and edges, are completely unobserved during training.

B. Experimental Setup

In our experiments, Adam optimizer [48] is used to learn model parameters with an initial learning rate of 10^{-4}. For all datasets, we use two layers with 512 node representation dimensions (i.e., $d^{(1)} = d^{(2)} = 512$). We set the number of epochs to 100 for Cora and Citeseer, and 500 for Pubmed. We also set λ to 0.5 for Cora and Pubmed, and 20 for Citeseer. We use only half of the trainable parameters by setting $\widehat{\mathbf{W}}^{(k)} = \mathbf{W}^{(k)^T}$ and $\widehat{\mathbf{C}}^{(k)} = \mathbf{C}^{(k)}$. Moreover, σ is set to the identity function, empirically resulting in better performance compared to other activation functions. We have used Tensorflow to implement GATE [1].

For the baselines, we use their default hyperparameter settings as well as the following settings. We perform a hyperparameter sweep on initial learning rates $\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$ and $\{10^{-2}, 10^{-3}, 10^{-4}\}$ for unsupervised and supervised methods respectively. We also swept over the number epochs in the set $\{50, 100, 200, 300\}$ for VGAE and GAE due to their sensitivity to this hyperparameter. For supervised methods, we perform a sweep on dropouts $\{0, 0.2, 0.5\}$. We also set the number of node representation dimensions to 512 for all of these baselines.

C. Comparison

In this section, we compare out proposed method with the aforementioned state-of-the-art baselines based on transductive and inductive node classifications. For transductive node classification, we report the mean classification accuracy (with standard deviation) of our method on the test nodes after 100 runs of training (followed by logistic regression). The accuracies for GCN, DeepWalk, and LP are retrieved from Kipf & Welling [22]. We also reuse the metrics reported in Velickovic et al. [28] for the performance of enhanced DeepWalk, DGI, and logistic regression with raw features. The accuracies for GAT, Chebyshev, and Monet are taken from Velickovic et al. [23]. For StoGCN and Planetoid, the metrics are retrieved from their papers [19], [47]. Moreover, we directly compare our method against GAE and VGAE.

Table IIIa shows the transductive node classification accuracies for the Cora, Citeseer, and Pubmed datasets. Accordingly, we make the following observations:

- GATE achieves strong performance across all three datasets. Particularly, GATE outperforms all baselines on the Cora and Pubmed datasets.
- Our unsupervised architecture is competitive with the performance of the best supervised baseline (i.e., GAT), even improving upon it by a margin of 1.9% and 0.2% on Pubmed and Cora respectively.
- GATE outperforms or matches all unsupervised baselines across all datasets. We observe an improvement of 2.7% and 0.9% over the best unsupervised baselines for Pubmed and Cora respectively.
- For the Citeseer dataset, the accuracy of GATE follows that of GAT. This can be attributed to the low average node degree of 1.4 for Citeseer—which is lower than Cora's (2) and Pubmed's (2.25). The scarcity of neighbors and the abundance of features can give the supervised baselines (i.e., GAT), which take advantage of a supervised loss, leverage over unsupervised methods

[1]The implementation of our architecture may be found at: https://github.com/amin-salehi/GATE

TABLE III: Node classification accuracies on the Cora, Citeseer and Pubmed datasets. The first column shows the type of data used during training for each method. Data types are the node feature matrix $\mathbf{X}$, adjacency matrix $\mathbf{A}$, and labels $\mathbf{Y}$.

(a) Transductive

Available Data	Method	Cora	Citeseer	Pubmed
$\mathbf{X}$	Raw features	$47.9 \pm 0.4\%$	$49.4 \pm 0.2\%$	$69.1 \pm 0.3\%$
$\mathbf{A}$	DeepWalk (Perozzi et al. [4])	67.2%	43.2%	65.3%
$\mathbf{A}$, $\mathbf{Y}$	LP (Zhu et al. [46])	68.0%	45.3%	63.0%
$\mathbf{X}$, $\mathbf{A}$	DeepWalk + features	$70.7 \pm 0.6\%$	$51.4 \pm 0.5\%$	$74.3 \pm 0.9\%$
$\mathbf{X}$, $\mathbf{A}$	VGAE (Kipf & Welling [25])	$72.4 \pm 0.2\%$	$55.7 \pm 0.2\%$	71.6 ± 0.4 %
$\mathbf{X}$, $\mathbf{A}$	GAE (Kipf & Welling [25])	$81.8 \pm 0.1\%$	$69.2 \pm 0.9\%$	$78.2 \pm 0.1\%$
$\mathbf{X}$, $\mathbf{A}$	DGI (Velickovic et al. [28])	$82.3 \pm 0.6\%$	$71.8 \pm 0.7\%$	$76.8 \pm 0.6\%$
$\mathbf{X}$, $\mathbf{A}$	**GATE** (ours)	$\mathbf{83.2} \pm 0.6\%$	$71.8 \pm 0.8\%$	$\mathbf{80.9} \pm 0.3\%$
$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	Planetoid (Yang et al. [19])	75.7%	62.9%	75.7%
$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	Chebyshev (Defferrard et al. [20])	81.2%	69.8%	74.4%
$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	Monet (Monti et al. [21])	$81.7 \pm 0.5\%$	—	$78.0 \pm 0.3\%$
$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	GCN (Kipf & Welling [22])	81.5%	70.3%	79.0%
$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	StoGCN (Chen et al. [47])	$82.0 \pm 0.8\%$	$70.9 \pm 0.2\%$	$79.0 \pm 0.4\%$
$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	GAT (Velickovic et al. [23])	$83.0 \pm 0.7\%$	$\mathbf{72.5} \pm 0.7\%$	$79.0 \pm 0.3\%$

(b) Inductive

Available Data	Method	Cora	Citeseer	Pubmed
$\mathbf{X}$, $\mathbf{A}$	GraphSAGE-LSTM (Hamilton et al. [24])	$50.1 \pm 0.2\%$	$40.3 \pm 0.2\%$	$77.1 \pm 0.1\%$
$\mathbf{X}$, $\mathbf{A}$	GraphSAGE-pool (Hamilton et al. [24])	$57.5 \pm 0.2\%$	$45.9 \pm 0.2\%$	$79.9 \pm 0.1\%$
$\mathbf{X}$, $\mathbf{A}$	VGAE (Kipf & Welling [25])	$58.4 \pm 0.4\%$	$55.4 \pm 0.2\%$	$71.1 \pm 0.2\%$
$\mathbf{X}$, $\mathbf{A}$	GraphSAGE-mean (Hamilton et al. [24])	$67.0 \pm 0.2\%$	$52.8 \pm 0.1\%$	$79.3 \pm 0.1\%$
$\mathbf{X}$, $\mathbf{A}$	GraphSAGE-GCN (Hamilton et al. [24])	$74.3 \pm 0.1\%$	$54.5 \pm 0.1\%$	$77.5 \pm 0.1\%$
$\mathbf{X}$, $\mathbf{A}$	GAE (Kipf & Welling [25])	$80.5 \pm 0.1\%$	$69.1 \pm 0.9\%$	$78.1 \pm 0.2\%$
$\mathbf{X}$, $\mathbf{A}$	**GATE** (ours)	$\mathbf{82.5} \pm 0.5\%$	$\mathbf{71.5} \pm 0.7\%$	$\mathbf{80.8} \pm 0.3\%$
$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	Planetoid (Yang et al. [19])	61.2%	64.7%	77.2%
$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	GAT (Velickovic et al. [23])	$76.4 \pm 0.2\%$	$66.4 \pm 0.2\%$	$77.7 \pm 0.03\%$

- The reconstruction of node features by GATE results in a considerable improvement compared to the graph auto-encoder baselines reconstructing only the graph structure. Compared to the best graph auto-encoder baseline (i.e., GAE), we achieve an improvement gain of 2.7%, 2.6%, and 1.4% on Pubmed, Citeseer, and Cora respectively.

For inductive node classification, we utilize the same datasets used for the transductive tasks. This enables us to compare the performance of GATE between transductive and inductive tasks for the same dataset in order to evaluate the generalization power of our auto-encoder to unseen nodes. We report the mean classification accuracy (with standard deviation) of our method on the (unseen) test nodes after 100 runs of training (followed by logistic regression). For Planetoid, its accuracies are retrieved from Yang et al. [19]. We directly compare our method against VGAE, GAE, GAT, and four variants of GraphSAGE.

Table IIIb shows the inductive node classification accuracies for the Cora, Citeseer, and Pubmed datasets. Accordingly, we make the following observations:

- GATE exceeds the performance of all baselines across all three datasets. We are able to improve upon the best baselines by a margin of 2.4%, 2%, and 0.9% on Citeseer, Cora, and Pubmed respectively.
- We can observe that GATE achieves similar accuracies for inductive and transductive tasks with regard to the same dataset. For example, the accuracy difference between inductive and transductive tasks is 0.1%, 0.3%, and 0.7% on Pubmed, Citeseer, and Cora respectively.
- Unlike GATE, not every method performing well on transductive tasks can perform well on inductive tasks.

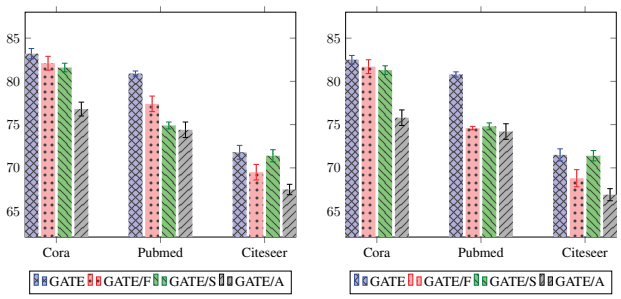
D. In-depth Analysis

In this section, we investigate the impact of the three main components used in our proposed architecture, namely the self-attention mechanism, graph structure reconstruction and node feature reconstruction. In our experiments, we use the following variants of our architecture:

- **GATE**: The full version of our proposed auto-encoder which includes all three components.
- **GATE/A**: A variant of our architecture which includes all components except the self-attention mechanism. In other words, we assign the same importance to each neighbor.
- **GATE/S**: A variant of our architecture which includes all components except the graph structure reconstruction.
- **GATE/F**: A variant of our architecture which includes all components except the node feature reconstruction.

We first compare the four variants of our architecture based on transductive node classification. Figure 2a shows the mean classification accuracy (with standard deviation) of all four variants on the test nodes after 100 runs of training (followed by logistic regression). Accordingly, we make the following observations:

- GATE outperforms other variants in all datasets. Therefore, each component contributes to the overall performance of our architecture.



(a) Trandsuctive (b) Inductive

Fig. 2: Node classification accuracies on the Cora, Citeseer and Pubmed datasets for four variants of our architecture.

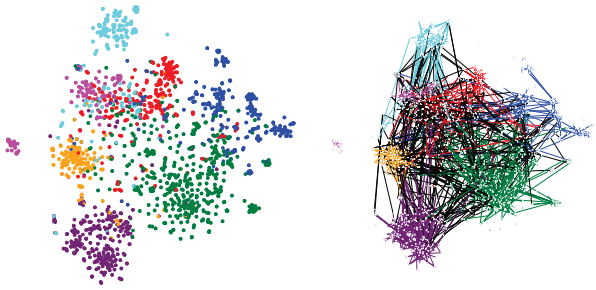
- GATE/A performs worse than other variants. This suggests that the self-attention mechanism contributes the most in our architecture compared to the graph structure and node feature reconstructions.
- In Cora and Pubmed which have higher average node degree (i.e., 2 and 2.5 respectively), GATE/F outweighs the performance of GATE/S. On the other hand, GATE/S exceeds the performance of GATE/F in Citeseer which has the lowest average node degree (i.e., 1.4) and the highest number of features.

Now we compare all variants of our architecture based on inductive node classification. Figure 2b shows the mean classification accuracy (with standard deviation) of all four variants on the (unseen) test nodes after 100 runs of training (followed by logistic regression). Accordingly, we make the following observations:

- Like the transductive node classification experiments, GATE and GATE/A are respectively the best and the worst variants of our architecture in all datasets.
- We observe that the performances of GATE/F and GATE/S in Cora and Citeseer are similar to those of transductive node classification experiments. However, we notice a huge drop in the performance of GATE/F in Pubmed even though the performance of GATE/S has not undergone such a decrease. This can be attributed to both the low number of features and high average node degree of Pubmed compared to those of Cora and Citeseer, which hugely benefit GATE/F in transductive learning over inductive learning.

E. Qualitative Analysis

In this section, we qualitatively investigate the effectiveness of the node representations and attention coefficients learned by GATE. To this end, we utilize t-SNE [49] to project the learned node representations into a two-dimensional space. Due to space limitation, we only show the visualization for the Cora dataset. Figure 3a shows the t-SNE visualization of the learned node representations for Cora, where node colors denote classes. We can observe that the learned node representations result in discernible clusters.



(a) The visualization of nodes. (b) The visualization of edges.

Fig. 3: The t-SNE visualizations of the node representations learned by GATE on the Cora dataset in node and edge perspectives.

Figure 3b shows the t-SNE visualization of the edges, in the Cora dataset, thickened by their attention coefficients averaged across all layers. In this figure, the edges with source and target nodes belonging to the same class are colored with the color of the class, and the others are colored black. Accordingly, we expect high-quality node representations to result in thicker colorful edges. In Figure 3b, we can observe that the colorful edges are usually thicker than the black edges. However, in few spots where GATE faces difficulty in separating nodes belonging to different classes, we can notice the presence of some thick black edges.

VI. Conclusion

Graph-structured data can be found in many real-world scenarios, such as social media [50], [51], protein-protein interaction networks [23], and citation networks [22]. In this paper, we introduced the graph attention auto-encoder (GATE), a novel neural architecture for unsupervised representation learning on graph-structured data. By stacking multiple encoder/decoder layers equipped with graph attention mechanisms, GATE is the first graph auto-encoder, which reconstructs both node features and the graph structure.

Experiments on both transductive and inductive tasks using three benchmark datasets demonstrate the efficacy of GATE, which learns high-quality node representations. In most experiments, our auto-encoder outweighs state-of-the-art supervised and unsupervised baselines. Moreover, our experiments show that GATE naturally generalizes to unseen nodes.

References

[1] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.

[2] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2018.

[3] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding." in *AAAI*, 2017, pp. 203–209.

[4] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[5] X. Wei, L. Xu, B. Cao, and P. S. Yu, "Cross view link prediction by learning noise-resilient representation consensus," in *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2017, pp. 1611–1619.

[6] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607.

[7] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in neural information processing systems*, 2002, pp. 585–591.

[8] X. He and P. Niyogi, "Locality preserving projections," in *Advances in neural information processing systems*, 2004, pp. 153–160.

[9] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd international conference on World Wide Web*. ACM, 2013, pp. 37–48.

[10] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 891–900.

[11] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 1105–1114.

[12] B. Perozzi, V. Kulkarni, and S. Skiena, "Walklets: Multiscale graph embeddings for interpretable network classification," *arXiv preprint arXiv:1605.02115*, 2016.

[13] B. P. Chamberlain, J. Clough, and M. P. Deisenroth, "Neural embeddings of graphs in hyperbolic space," *arXiv preprint arXiv:1705.10359*, 2017.

[14] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering." in *AAAI*, 2014, pp. 1293–1299.

[15] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 1225–1234.

[16] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations." in *AAAI*, 2016, pp. 1145–1152.

[17] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, "Harp: Hierarchical representation learning for networks," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[18] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 2017, pp. 731–739.

[19] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," *arXiv preprint arXiv:1603.08861*, 2016.

[20] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[21] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proc. CVPR*, vol. 1, no. 2, 2017, p. 3.

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[23] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[24] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[25] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *NeurIPS Workshop on Bayesian Deep Learning*, 2016.

[26] A. G. Duran and M. Niepert, "Learning graph representations with embedding propagation," in *Advances in Neural Information Processing Systems*, 2017, pp. 5119–5130.

[27] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding." in *IJCAI*, 2018, pp. 2609–2615.

[28] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *International Conference on Learning Representations*, 2019.

[29] H. Gao and H. Huang, "Deep attributed network embedding." in *IJCAI*, 2018.

[30] X. Huang, J. Li, and X. Hu, "Accelerated attributed network embedding," in *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 2017, pp. 633–641.

[31] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information." in *IJCAI*, 2015, pp. 2111–2117.

[32] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, 2012, pp. 37–49.

[33] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[34] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "Mgae: Marginalized graph autoencoder for graph clustering," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 2017, pp. 889–898.

[35] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[36] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in neural information processing systems*, 2015, pp. 577–585.

[37] L.-C. Chen, Y. Yang, J. Wang, W. Xu, and A. L. Yuille, "Attention to scale: Scale-aware semantic image segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3640–3649.

[38] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, no. 3, 2018, p. 4.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[40] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[41] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser, "Universal transformers," *arXiv preprint arXiv:1807.03819*, 2018.

[42] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," *arXiv preprint arXiv:1509.00685*, 2015.

[43] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[44] R. He, W. S. Lee, H. T. Ng, and D. Dahlmeier, "An unsupervised neural attention model for aspect extraction," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2017, pp. 388–397.

[45] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, p. 93, 2008.

[46] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," 2002.

[47] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction." in *ICML*, 2018, pp. 941–949.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[49] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[50] A. Salehi, M. Ozer, and H. Davulcu, "Sentiment-driven community profiling and detection on social media," in *Proceedings of the 29th on Hypertext and Social Media*, 2018, pp. 229–237.

[51] A. Salehi and H. Davulcu, "Detecting antagonistic and allied communities on social media," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2018, pp. 99–106.