entropy

MDPI

Article

Scalably Using Node Attributes and Graph Structure for Node Classification †

Arpit Merchant *, Ananth Mahadevan and Michael Mathioudakis

Department of Computer Science, University of Helsinki, 00014 Helsinki, Finland; ananth.mahadevan@helsinki.fi (A.M.); michael.mathioudakis@helsinki.fi (M.M.)

* Correspondence: arpit.merchant@helsinki.fi

† This paper is an extended version of our paper published in the Proceedings of the International Conference on Complex Networks and Their Applications, Madrid, Spain, 30 November–2 December 2021; pp. 511–522.

Abstract: The task of node classification concerns a network where nodes are associated with labels, but labels are known only for some of the nodes. The task consists of inferring the unknown labels given the known node labels, the structure of the network, and other known node attributes. Common node classification approaches are based on the assumption that adjacent nodes have similar attributes and, therefore, that a node's label can be predicted from the labels of its neighbors. While such an assumption is often valid (e.g., for political affiliation in social networks), it may not hold in some cases. In fact, nodes that share the same label may be adjacent but differ in their attributes, or may not be adjacent but have similar attributes. In this work, we present JANE (Jointly using Attributes and Node Embeddings), a novel and principled approach to node classification that flexibly adapts to a range of settings wherein unknown labels may be predicted from known labels of adjacent nodes in the network, other node attributes, or both. Our experiments on synthetic data highlight the limitations of benchmark algorithms and the versatility of JANE. Further, our experiments on seven real datasets of sizes ranging from 2.5K to 1.5M nodes and edge homophily ranging from 0.86 to 0.29 show that JANE scales well to large networks while also demonstrating an up to 20% improvement in accuracy compared to strong baseline algorithms.

Keywords: node classification; graph embedding; representation learning

Citation: Merchant, A.; Mahadevan, A.; Mathioudakis, M. Scalably Using Node Attributes and Graph Structure for Node Classification. *Entropy* **2022**, *24*, 906. https://doi.org/10.3390/e24070906

Academic Editor: Adam Lipowski

Received: 26 May 2022
Accepted: 23 June 2022
Published: 30 June 2022

1. Introduction

Node classification involves an attributed network with a known graph structure, where each node is associated with a *label* (or *class*, a categorical variable), as well as other attributes. Moreover, labels are known only for some of the nodes and are considered unknown for others. Given the graph structure, the labels that are known for some of the nodes, and other attributes that are known for all nodes, the task is to predict the labels that are unknown. This task finds application in a wide range of domains, such as information networks [1], complex systems [2], protein function identification [3], medical term semantic classification [4] and disease prediction [5].

1.1. Previous Work

A common assumption in the literature is that adjacent nodes tend to have similar labels, often represented in theories of homophily [6] and social influence [7]. For instance, in a social network, the assumption stipulates that friends, i.e., close social connections, are likely to vote for the same political party as seen in Figure 1(left). Approaches that rely on this assumption typically consider node proximity and assign the same label to nearby nodes. For example, in label propagation, labels diffuse in an iterative fashion from labeled nodes to their unlabeled neighbors until convergence [8]. Other approaches induce label uniformity within cuts or clusters of the graph [9–11], or consider node proximity

in a latent space that preserves graph distances, as in DeepWalk [12] and similar matrix factorization approaches [13].

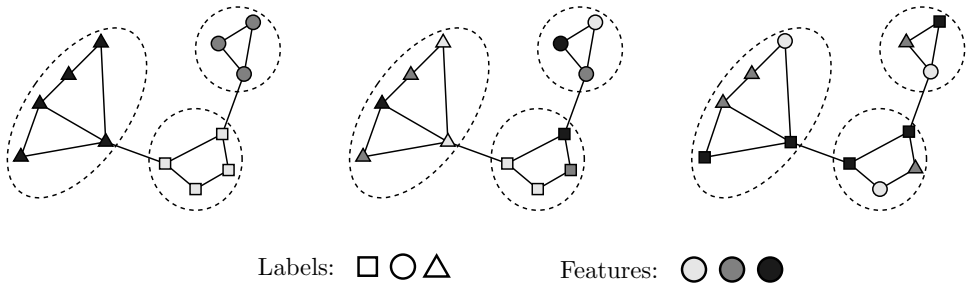


Figure 1. Nodes with the same labels may be adjacent and have similar features (**left**), or may be adjacent but have different features (**center**), or may not be adjacent but have similar features (**right**). Features are depicted by colors and labels are depicted by shapes. GCN and GraphSAGE can perform well in the first two cases but not the third because they predict node labels by aggregating the features of adjacent nodes.

However, the aforementioned methods ignore other node attributes, which can be detrimental. For example, Hamilton et al. [3] show that, for certain predictive tasks on citation and social graphs, a linear classifier that is built only on node attributes outperforms approaches, such as DeepWalk, which are based on node proximity but ignore node attributes. Moreover, although homophily is often observed in some classification tasks, it is not uncommon to find that adjacent nodes do not share a particular label and that, in such cases, other node attributes can serve as better label predictors than graph structure [14,15]. For example, two individuals may be friends (i.e., be connected on a social network) but vote for different political parties ('label')—something that could be better predicted by rich, node-level attribute data (e.g., geographic location, income, or profession). Therefore, for node classification, it is important to appropriately leverage both the proximity of nodes on the graph structure and other attributes.

Partially addressing this limitation, AANE [16] and DANE [17] combine low-dimensional encodings of node attributes with graph-distance-preserving node embeddings, and use them as input features for label prediction. However, they do not account for known labels during training, and thus potentially ignore information that would be useful in predicting the unknown labels. LANE [18] overcomes this limitation by learning the joint latent representations of node attributes, proximity, and labels. However, LANE does not directly address the node classification task, i.e., it does not optimize the conditional probability distribution of node labels given the node attributes and graph structure, but rather targets their joint distribution of all quantities.

In a different line of work, graph convolutional networks use graph topology for low-pass filtering on node features [19]. GAT [20] introduces an attention mechanism to learn weights and aggregate features. GraphSAGE [3] uses mean/max pooling to sample and aggregate features from nodes' local neighbourhoods. However, these convolutions are equivalent to a repeated smoothing over the node attributes, and performance quickly degrades [21]. Subsequent approaches such as DiffPool [22] have sought to address this limitation, but these too aggressively enforce homophily and require that nodes with the same labels have similar graph and attribute representations. Recent work by AM-GCN [23] attempts to weaken this assumption by analyzing the fusing abilities of convolutional models. They define two modules, one each for the topology space and feature space, and adaptively combine them using an attention mechanism. We direct the reader to the survey paper of Xiao et al. [24]'s for a comparison of recent works on graph convolutions for the task of node classification.

1.2. Our Contribution

In this work, we develop an approach to node classification that can flexibly adapt and perform consistently well in a range of settings, from cases where node labels exhibit strong homophily (i.e., a node's label can be determined by the labels of its neighbors) to cases where labels are independent of graph structure and solely determined by other node attributes, or vice versa, as well as cases that lie between these extremes. We propose a novel and principled approach to node classification based on a generative probabilistic model that jointly captures the role of graph structure and node similarity in predicting labels. Our analysis leads to JANE, a training algorithm that learns two unknown model parameters, namely, a latent node embedding $\mathbf{U}$ and weight matrices $\mathbf{W}$ of a neural network for the task of node classification. JANE combines node attributes with the embedding and iteratively updates both model parameters $\mathbf{U}$ and $\mathbf{W}$ with label information during training. We show that this flexibly adapts to the prediction task in a variety of cases, depending on the correlation between attributes and graph structure on node labels. Unlike the aforementioned approaches, which are heavily based on label homophily (e.g., label propagation), we account not only for the graph structure but also for node attributes, and flexibly and appropriately weigh each of them depending on the case. Moreover, unlike AANE [16] and DANE [17], our approach learns a low-dimensional node representation that is informed by labels, which is then used for prediction. Unlike LANE [18], we directly optimize the conditional probability of labels given the graph structure and node attributes, without necessarily enforcing homophily.

We summarize our main contributions below:

- We define a generative model that formally captures a diverse set of relationships between graph structure, node attributes, and node labels.
- We describe a general algorithm to compute a good initial estimate of $\mathbf{U}$ and a training algorithm called JANE for node classification. We also design batching and minibatching variants of JANE that scale well to large graphs.
- From our experimental results, we present three findings. First, we demonstrate the shortcomings of existing approaches and versatility of JANE on synthetic datasets. Second, we empirically validate the performance of four variants of JANE on seven real-world datasets and compare to five standard baselines. Thirdly, we conduct an extensive empirical analysis characterizing the usefulness of a good initial node embedding, the importance of updates to the embedding during training, and the trade-off between preserving adjacency and label information on classification accuracy.

2. Problem Setting

Let us consider an undirected and connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of node size $|\mathcal{V}| = n$. Let its structure be represented by the adjacency matrix $\mathbf{A} = \left[a_{ij}\right] \in \mathbb{R}^{n \times n}$. Denote $\mathbf{D} = \text{diag}(d_1, d_2, \ldots, d_n)$ to be the degree matrix where $d_i = \sum_j a_{ij}$; and $\mathbf{L} = \mathbf{D} - \mathbf{A}$ as its unnormalized Laplacian matrix. Let λ_i be the i-th smallest eigenvalue of $\mathbf{L}$ and $\mathbf{e}_i$ its corresponding eigenvector.

Each node in the graph is associated with the following: d observed attributes $\mathbf{x} \in \mathbb{R}^d$, k latent/unobserved node embeddings $\mathbf{u} \in \mathbb{R}^k$, and possibly an unobserved categorical variable $\mathbf{y} \in \{0,1\}^M$ (one-hot encoding) as the label from label-set $\mathbf{M} = \{1, 2, \ldots, M\}$. For example, in citation graphs, with nodes corresponding to articles and edges to citations between articles, $\mathbf{x}$ capture observed quantities such as the bag-of-words representation of the article text, and label $\mathbf{y}$ denotes the research area of the article (e.g., 'data mining' or 'machine learning'). The latent embedding $\mathbf{u}$ corresponds to the properties of the articles that are not directly captured by attributes $\mathbf{x}$ or label $\mathbf{y}$, but that could play a role in determining which articles are connected with a citation (as captured by adjacency matrix $\mathbf{A}$) and to which research area $\mathbf{y}$ an article is deemed to belong. In terms of notation, to refer to the attributes of all nodes, we write $\mathbf{X} = \left\{\mathbf{x}_i \in \mathbb{R}^d, i \in \{1, \ldots, n\}\right\}$ to denote the

observed node attributes, $\mathbf{U} = \left\{\mathbf{u}_i \in \mathbb{R}^k, i \in \{1,\ldots,n\}\right\}$ for the latent node embedding, and $\mathbf{Y} = \left\{\mathbf{y}_i \in \{0,1\}^M, i \in \{1,\ldots,n\}\right\}$ for the node labels. This naturally extends to the multi-label classification setting.

Having defined all elements in our setting, we now define the task that we address as Problem 1.

Problem 1 (Node-Classification). *Given adjacency matrix* $\mathbf{A}$*, node features* $\mathbf{X}$*, and labels* $\mathbf{Y}_L$ *for a subset* $L \subseteq \mathcal{V}$ *of nodes, predict labels* $\mathbf{Y}_{\mathcal{V}\setminus L}$ *for the remaining nodes* $\mathcal{V} \setminus L$ *in the graph.*

3. Our Approach

Our approach for Problem 1 is based on a probabilistic generative model (described in Section 3.1) and its analysis (Section 3.2).

3.1. Model

Figure 2 pictorially illustrates our generative model. First, the adjacency matrix $\mathbf{A}$ of the graph is generated from the latent embedding $\mathbf{U}$. Specifically, the probability that there is an edge between two nodes i and j is given by the inverse exponent of the squares $l2$-distance between their latent attributes $\mathbf{u}$ scaled by a factor s^2.

$$\Pr\left[\ (i,j) \in \mathcal{E} \mid \mathbf{u}_i, \mathbf{u}_j; s\ \right] = p_{ij} = e^{\frac{-\left\|\mathbf{u}_i - \mathbf{u}_j\right\|^2}{s^2}} \tag{1}$$

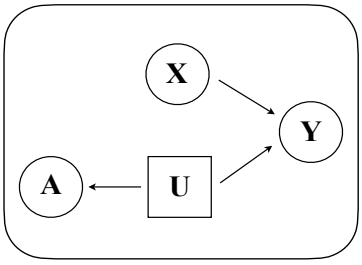


Figure 2. Visual illustration of the generative framework. Observed node attributes $\mathbf{X}$ (represented by circular box) and unobserved (latent) embeddings $\mathbf{U}$ (represented by square box) jointly generate node labels $\mathbf{Y}$. The (observed) adjacency matrix $\mathbf{A}$ is generated from $\mathbf{U}$ and indirectly correlates with $\mathbf{Y}$ (via $\mathbf{U}$). Reprinted with permission from Springer Nature Customer Service Centre GmbH: Springer, International Conference on Complex Networks and Their Applications, Joint Use of Node Attributes and Proximity for Node Classification, Merchant and Mathioudakis [25], 2021.

This equips our model with the desirable property, common in many types of graph embeddings, that the closer the two nodes are in the Euclidean space of $\mathbf{U}$, the higher the likelihood that they are connected in the graph and vice versa. Therefore, $\mathbf{U}$ represents a low-dimensional Euclidean embedding of the graph that preserves connectivity in the form of Equation (1). Moreover, since the existence of an edge is independent across pairs of nodes in this model, we have

$$\Pr[\mathbf{A}|\mathbf{U}; s] = \prod_{(i,j)\in\mathcal{E}} p_{ij} \times \prod_{(i,j)\notin\mathcal{E}} \left(1 - p_{ij}\right). \tag{2}$$

Notice that, with regard to Equations (1) and (2), the scaling factor s can be absorbed into the embedding parameter $\mathbf{U}$, i.e., any pair $(\mathbf{U}, s)$ of parameters is equivalent to the pair of parameters $(\mathbf{U}/s, 1)$. For this reason, we will omit s from the probability expressions that follow.

There are several other graph-generation models, such as the ϵ-neighbourhoods model, wherein nodes i, j are connected by an edge if $\left\|\mathbf{u}_i - \mathbf{u}_j\right\|^2 \leq \epsilon$ [26]. However,

the ϵ-neighbourhoods model often leads to several connected components. Moreover, Equation (2) is in line with typical assumptions in spectral graph theory [27].

Second, for fixed scale s, node labels $\mathbf{Y}$ are generated from $\mathbf{X}$ and $\mathbf{U}$. This assumption provides two benefits: (i) it allows for labels to be determined by node attributes $\mathbf{X}$ (directly) as well as graph structure $\mathbf{A}$ (indirectly, via $\mathbf{U}$); (ii) it allows for us to directly express and train the function of the conditional distribution $\Pr[\mathbf{Y}|\mathbf{X}, \mathbf{U}]$, which we then employ for *node classification*, i.e., to predict unobserved node labels.

In this work, we assume that this conditional probability is given by a simple two-layer neural network,

$$\Pr[\mathbf{Y}|\mathbf{X}, \mathbf{U}, \mathbf{W}] = \sigma\Big(\mathrm{ReLU}\Big(\texttt{CONCAT}[\mathbf{X},\ \mathbf{U}]W^{(0)}\Big)W^{(1)}\Big) \tag{3}$$

where σ denotes the softmax function and weight matrices $\mathbf{W} = \{W^{(0)}, W^{(1)}\}$ are parameters that control the effect of $\mathbf{X}$ and $\mathbf{U}$ on labels $\mathbf{Y}$.

The reason for this choice is that we found this model to be sufficiently expressive for our empirical evaluation. We note that this conditional probability (RHS in Equation (3)) can be replaced with other complex models (e.g., neural networks with more hidden layers).

3.2. Algorithms

Given data $\mathcal{D} = (\mathbf{X}, \mathbf{A}, \mathbf{Y}_L)$ as input, Problem 1 asks for predictions for $\mathbf{Y}_{\mathcal{V}\setminus L}$. Here, the latent embedding $\mathbf{U}$ and the weights of the neural network $\mathbf{W}$, are considered as unknown parameters $\subseteq = (\mathbf{U}, \mathbf{W})$ of the model. Our approach, JANE, proceeds in two stages: first, a training stage, from which we learn the maximum likelihood estimates $\hat{\theta} = \left(\widehat{\mathbf{U}}, \widehat{\mathbf{W}}\right)$ of the model (cf. Equation (3)); second, a prediction stage, in which we use $\hat{\theta}$ to predict the missing labels.

3.2.1. Training

Training Objective. From the product rule of probability, the likelihood of the data $\mathcal{D}$ given parameters $\subseteq$ is as follows:

$$\begin{aligned} \Pr[\mathcal{D}|\subseteq] &= \Pr[\mathbf{X}, \mathbf{A}, \mathbf{Y}_L|\mathbf{U}, \mathbf{W}] \\ &\propto \Pr[\mathbf{Y}_L|\mathbf{X}; \mathbf{U}, \mathbf{W}] \times \Pr[\mathbf{A}|\mathbf{U}] \end{aligned} \tag{4}$$

Taking the negative logarithm on both sides, we define the total loss as the sum of the loss with respect to node labels ($\mathcal{L}_{\text{label}}$) and the loss with respect to the graph structure ($\mathcal{L}_{\text{graph}}$).

$$\begin{aligned} \mathcal{L}_{\text{total}} &= -\log \Pr[\mathcal{D}|\subseteq] \\ &= \underbrace{-\log \Pr[\mathbf{Y}_L|\mathbf{X}; \mathbf{U}, \mathbf{W}]}_{\mathcal{L}_{\text{label}}} \underbrace{-\log \Pr[\mathbf{A}|\mathbf{U}]}_{\mathcal{L}_{\text{graph}}} \end{aligned} \tag{5}$$

The goal is to identify parameter estimates $\hat{\theta}$ that minimize the objective function $\mathcal{L}_{\text{total}}$. We begin by describing a general routine to construct a good initial estimate of $\mathbf{U}$ (Algorithm 1). Then, we describe a training procedure to update $\widehat{\mathbf{U}}$ and $\widehat{\mathbf{W}}$ to their maximum-likelihood estimates of the model using stochastic gradient descent, starting from $\widehat{\mathbf{U}}_{\text{initial}}$ (Algorithm 2).

Algorithm 1: INITIALIZE.

Input: Adjacency Matrix $\mathbf{A}$; Embedding method EMBEDDING $\in$ {LAPLACIAN, GOSH}, embedding dimensionality k;
Output: Latent attributes $\widehat{\mathbf{U}}_{\text{initial}}$, Scale s;
$s \leftarrow 1$
$\overline{\mathbf{U}} \leftarrow \texttt{EMBEDDING}\left(\left(\frac{1}{s} \cdot \mathbf{A}\right),\ k\right)$
$s \leftarrow \texttt{MINIMIZE}\left(\text{objective} = \mathcal{L}_{\text{edge}} + \mathcal{L}_{\text{nonedge}}, \text{constants} = \mathbf{A}, \overline{\mathbf{U}}\right)$ // (cf. Equation (7))
$\widehat{\mathbf{U}}_{\text{initial}} \leftarrow \frac{1}{s} \cdot \overline{\mathbf{U}}$, $s_{\text{initial}} \leftarrow 1$ // Rescale
return $\widehat{\mathbf{U}}_{\text{initial}}$, s_{initial}

Algorithm 2: JANE-U (Batching).

Input: Adjacency matrix $\mathbf{A}$; Node attributes $\mathbf{X}$; Training labels $\mathbf{Y}_L$; Embedding method EMBEDDING $\in$ {LAPLACIAN, GOSH}; embedding dimensionality k; number of training epochs T; learning rates η_{label}, η_{graph}, η_{weight}; number of batches K;
Output: Vector representations $\mathbf{h}_v$ for all $v \in \mathcal{V}$;
// Initialization
$\widehat{\mathbf{U}}, s \leftarrow \texttt{INITIALIZE}(\mathbf{A}, \texttt{Emd}, k)$ // (cf. Algorithm 1)
Create K batches of nodes $\mathbb{B}_1 \cup \ldots \cup \mathbb{B}_K \leftarrow \mathcal{V}$
for $t \leftarrow 1$ ***to*** T **do**
  for $b \leftarrow 1$ ***to*** K **do**
    // Forward propagation (cf. Equation (3))
    $\forall i \in \mathbb{B}_b, \mathbf{h}_i^{(t)} \leftarrow \sigma\left(\text{ReLU}\left(\texttt{CONCAT}\left[\mathbf{X}_{\mathbb{B}_b},\ \widehat{\mathbf{U}}_{\mathbb{B}_b}^{(t-1)}\right] W^{(0)}\right) W^{(1)}\right)$
    // Update $\widehat{\mathbf{U}}$ (cf. Equation (9), Equation (10))
    $\widehat{\mathbf{U}}_{\mathbb{B}_b}^{(t)} \leftarrow \widehat{\mathbf{U}}_{\mathbb{B}_b}^{(t-1)} - \eta_{\text{label}} \cdot \frac{\partial \mathcal{L}_{\text{label}}}{\partial \widehat{\mathbf{U}}_{\mathbb{B}_b}^{(t-1)}} - \eta_{\text{graph}} \cdot \frac{\partial \mathcal{L}_{\text{graph}}}{\partial \widehat{\mathbf{U}}_{\mathbb{B}_b}^{(t-1)}}$
  end
  // Update $\widehat{\mathbf{W}}$
  $\widehat{\mathbf{W}}^{(t)} \leftarrow \widehat{\mathbf{W}}^{(t-1)} - \eta_{\text{weight}} \cdot \frac{\partial \mathcal{L}_{\text{label}}}{\partial \widehat{\mathbf{W}}^{(t-1)}}$
end
return $\forall v \in \mathcal{V}, \mathbf{h}_u^T$

Choosing $\mathbf{U}_{\text{initial}}$. We construct an initial estimate of $\widehat{\mathbf{U}}$ and s in an unsupervised manner using only the adjacency information and (temporarily) ignore node labels. That is,

$$\widehat{\mathbf{U}}_{\text{initial}}, s_{\text{initial}} = \arg\min_{\mathbf{U},s}\ \mathcal{L}_{\text{graph}} \tag{6}$$

Plugging in Equation (2), we get

$$\begin{aligned}\widehat{\mathbf{U}}_{\text{initial}}, s_{\text{initial}} &= \arg\min_{\mathbf{U},s} \sum_{(i,j)\in\mathcal{E}} \frac{\left\|\mathbf{u}_i - \mathbf{u}_j\right\|^2}{s^2} - \log\left(\prod_{(i,j)\notin\mathcal{E}}\left(1 - e^{\frac{-\left\|\mathbf{u}_i-\mathbf{u}_j\right\|^2}{s^2}}\right)\right) \\ &\leq \arg\min_{\mathbf{U},s} \underbrace{\sum_{(i,j)\in\mathcal{E}} \frac{\left\|\mathbf{u}_i - \mathbf{u}_j\right\|^2}{s^2}}_{\mathcal{L}_{\text{edge}}} + \underbrace{\sum_{(i,j)\notin\mathcal{E}} e^{\frac{-\left\|\mathbf{u}_i-\mathbf{u}_j\right\|^2}{s^2}}}_{\mathcal{L}_{\text{nonedge}}}\end{aligned} \tag{7}$$

where the last inequality holds because $-(\log(1-x)) \leq x, x \in [0,1)$. The first term, $\mathcal{L}_{\text{edge}}$, captures the likelihood of observing all edges in the graph, while the second term, $\mathcal{L}_{\text{nonedge}}$, captures the likelihood of all non-edges (i.e., node pairs without an edge between them).

We obtain the initial estimates for $\mathbf{U}$ and s by iteratively minimizing the upper bound of Equation (7). In each iteration, we first fix the scale s and minimize over $\mathbf{U}$ to obtain $\overline{\mathbf{U}}$; then, we fix $\overline{\mathbf{U}}$ and minimize over s to obtain $\widehat{\mathbf{U}}_{\text{initial}}$. We have explored two approaches to obtain $\overline{\mathbf{U}}$, namely via *Laplacian eigenvectors* and *GOSH embeddings*, and we describe them below in this section. Moreover, we use Scipy's implementation of the standard *Brent* bounded scalar optimization algorithm (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize_scalar.html (25 May 2022)) to obtain s within each iteration. Note that, in practice, we observed that a single iteration is sufficient to obtain good initial estimates for $\mathbf{U}$ and s in both aforementioned approaches, and we thus use a single iteration for the initialization in all that follows. The initialization procedure is shown in Algorithm 1. Note that, for conventional simplicity, we always absorb the scale factor s into the initial embedding $\mathbf{U}$ at the end of the initialization procedure (Algorithm 1, Step 6).

We now proceed to describe the two approaches we have explored to obtain $\overline{\mathbf{U}}$.

1. *Laplacian Eigenvectors* [25]: Observe that for an appropriate choice of scale, i.e., for smaller values of s, $\mathcal{L}_{\text{nonedge}}$ tends to 0 and the value of $\mathcal{L}_{\text{edge}}$ dominates Equation (7). Moreover, given that $\mathbf{L}$ is symmetric and $d_i = \sum_j a_{ij}, \forall i \in [n]$, based on the results of Belkin et al. [26],

$$\min_{\substack{\mathbf{U}:\forall l\in[k],\\ \|\mathbf{u}_l\|=1,\\ \sum_p \mathbf{u}_{l_p}=0}} \sum_{(i,j)\in\mathcal{E}} \left\|\mathbf{u}_i - \mathbf{u}_j\right\|^2 = \text{tr}\left(\mathbf{U}^{\text{T}}\mathbf{L}\mathbf{U}\right) = \sum_{l=1}^{k} \lambda_l.$$

The condition $\mathbf{U} : \forall l \in [k],\ \|\mathbf{u}_l\| = 1, \sum_p \mathbf{u}_{l_p} = 0$ normalizes the columns of $\mathbf{U}$, removes translational invariance, and centers the solution around $\not\vdash$. This result implies that the minimum value of $\mathcal{L}_{\text{edge}}$ in Equation (7) is the sum of the k smallest eigenvalues of the graph Laplacian. Moreover, this minimum value is achieved when columns of $\mathbf{U}$ are the corresponding eigenvectors. That is:

$$\overline{\mathbf{U}} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_k]. \tag{8}$$

where $\mathbf{e}_i$ is the i-th smallest spectral eigenvector. We make use of the above to obtain a heuristic value for $\overline{\mathbf{U}}$. Specifically, in Step 4 of Algorithm 1, we set $\overline{\mathbf{U}}$ to the Laplacian eigenvectors, effectively choosing to minimize only $\mathcal{L}_{\text{edge}}$ and ignore $\mathcal{L}_{\text{nonedge}}$. Note that when selecting the scale factor s at Step 5 we make use of the sum $\mathcal{L}_{\text{edge}} + \mathcal{L}_{\text{nonedge}}$, i.e., the full Equation (7).

2. *GOSH embeddings* [28]: GOSH is an efficient and state-of-the-art method to obtain embeddings that preserve vertex-to-vertex similarity measures. For our purposes, we use the normalized adjacency matrix of the graph as the similarity measure. While the objective function that is optimized by GOSH out-of-the-box is not identical to that of Equation (7), the resulting optimization performed by GOSH leads to embeddings that preserve both edge and non-edge information, in line with our objective in Equation (7). To further ensure that GOSH provides an embedding with a good objective value, we perform a grid-search over its hyperparameters and maintain the GOSH embedding with maximum objective value as per Equation (7). We have found this approach to work extremely well in practice.

GOSH vs. Laplacian Embeddings: We found experimentally that performing the initialization of $\mathbf{U}$ with GOSH rather than Laplacian embeddings consistently leads to a better performance. Therefore, for ease of presentation in what follows, we will use GOSH for the initialization procedure (Algorithm 1), unless explicitly mentioned otherwise. Beyond Laplacian and GOSH embeddings, this framework is flexible enough to support other unsupervised node embeddings, such as VERSE [29], Force2vec [30], which aim to preserve edge and non-edge information.

Updating Model Parameters. Next, we describe the training procedure for JANE to minimize the overall loss $\mathcal{L}_{\text{total}}$. Training begins with the values $\mathbf{U} = \mathbf{U}_{initial}$ and $s = 1$ provided by Algorithm 1, with the scale s remaining fixed during training. During each training epoch, JANE starts by fixing $\widehat{\mathbf{W}}$ and updating its estimate $\widehat{\mathbf{U}}$ using gradient

descent on $\mathcal{L}_{\text{total}}$ (cf. Equation (5)). This involves computing two gradients. First, the gradient of $\mathcal{L}_{\text{label}}$ w.r.t. the i-th row of the current latent embedding, $\hat{\mathbf{u}}_i$, is obtained using backpropagation as follows,

$$\frac{\partial \mathcal{L}_{\text{label}}}{\partial \hat{\mathbf{u}}_i} = \sum_{l \in |\mathbf{Y}_L|} \left(\sum_{r=1}^{M} \left(a_{lr}^{(1)} - \mathbf{Y}_{lr} \right) W_{ri}^{(1)} \right) \times \left(\frac{a_{li}^{(0)}}{1 + \exp\left(-z_{li}^{(0)} \right)} \right) \times \left(\sum_{p=1}^{d+k} \left[W_{il}^{(0)} \right] \right) \tag{9}$$

where $a^{(0)}$, $a^{(1)}$ are activations from the hidden and output layers, and $z^{(0)}$ is the weighted sum from the input layer. The index i ranges over all nodes with known labels, and r indexes over the M different classes available for prediction. Second, the gradient of (upper-bounded) $\mathcal{L}_{\text{graph}}$ in Equation (7) w.r.t. the i-th row of the current latent embedding, $\hat{\mathbf{u}}_i$, is as follows.

$$\frac{\partial \mathcal{L}_{\text{graph}}}{\partial \hat{\mathbf{u}}_i} = \left(\sum_{j:(i,j) \in \mathcal{E}} 2 \times \frac{\hat{\mathbf{u}}_i - \hat{\mathbf{u}}_j}{s^2} \right) + \left(\sum_{j:(i,j) \notin \mathcal{E}} -2 \times \frac{\hat{\mathbf{u}}_i - \hat{\mathbf{u}}_j}{s^2} \times \mathrm{e}^{-\frac{\left\| \hat{\mathbf{u}}_i - \hat{\mathbf{u}}_j \right\|^2}{s^2}} \right) \tag{10}$$

where indices i and j range over all nodes in $\mathcal{V}$. The learning rates corresponding to the gradients above are denoted with $\eta_{\text{label}}, \eta_{\text{graph}}$.

Once $\widehat{\mathbf{U}}$ is updated, JANE treats it as fixed and updates the weights $\widehat{\mathbf{W}}$ of the neural network using stochastic gradient descent for $\mathcal{L}_{\text{total}}$ (cf. Equation (5)). Since $\mathcal{L}_{\text{graph}}$ is independent of $\mathbf{W}$, the optimization problem reduces to the following:

$$\widehat{\mathbf{W}} = \arg\min_{\mathbf{W}} \mathcal{L}_{\text{label}} = \arg\min_{\mathbf{W}} -\log\left(\Pr\left[\mathbf{Y}_L | \mathbf{X}; \widehat{\mathbf{U}}, \mathbf{W} \right] \right) \tag{11}$$

The gradient of $\mathcal{L}_{\text{label}}$ w.r.t $\widehat{\mathbf{W}}$ is computed using standard backpropagation (as in Step 7) with learning rate η_{weight}.

Batching. To reduce memory overhead and improve scalability to larger graphs, we adapt our training algorithm to allow the forward and backward propagations described above to take place in K smaller batches of nodes, where $\mathcal{V} = \mathbb{B}_1 \cup \ldots \cup \mathbb{B}_K$. Algorithm 2 gives the pseudocode for JANE using this batched approach. This takes advantage of the observation that, during any current iteration of gradient descent update, $\widehat{\mathbf{U}}_{\mathbb{B}_b}$ for a particular batch of nodes $b \in [K]$ does not depend on other nodes in the graph. An analogous minibatching variant called JANE-MU differs in only one aspect from JANE, namely that it updates $\widehat{\mathbf{W}}$ for every batch during each iteration. For completeness, we provide the pseudocode for the minibatching variant in Algorithm 3.

3.2.2. Prediction

Given maximum-likelihood estimates $\widehat{\mathbf{U}}^{(T)}$ and $\widehat{\mathbf{W}}_{(T)}$ at the end of T epochs, we predict labels $\widehat{\mathbf{Y}}_{\mathcal{V} \setminus L}$ for all nodes in $\mathcal{V} \setminus L$ using the softmax function applied row-wise

$$\widehat{\mathbf{Y}}_{\mathcal{V} \setminus L} = \arg\max_{r \in M} \sigma\left(\text{ReLU}\left(\texttt{CONCAT}\left[\mathbf{X}_{\mathcal{V} \setminus L}, \widehat{\mathbf{U}}_{\mathcal{V} \setminus L}^{(T)} \right] \hat{W}_{(T)}^{(0)} \right) \hat{W}_{(T)}^{(1)} \right)_r, \tag{12}$$

where $\mathbf{X}_{\mathcal{V} \setminus L}$ and $\widehat{\mathbf{U}}_{\mathcal{V} \setminus L}^{(T)}$ are the corresponding node features and latent embeddings.

3.2.3. Complexity Analysis

Computing $\widehat{\mathbf{U}}_{\text{initial}}$ using the GOSH embedding has a time complexity of $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ [28]. We assume that the dimensionality of the hidden neural layers are of the same order as $\mathcal{O}(d + k)$. A forward pass of JANE requires $\mathcal{O}\left((d+k)^2 L |\mathcal{V}| + (d+k)|\mathcal{E}| \right)$ where L is the number of layers, $\mathcal{O}((d+k)|\mathcal{E}|)$ is the cost of the first layer's linear mapping and $\mathcal{O}\left((d+k)^2 |\mathcal{V}| \right)$ is the cost of each subsequent hidden layers.

Algorithm 3: JANE-U-M (Mini-Batching).

Input: Adjacency matrix $\mathbf{A}$; Node attributes $\mathbf{X}$; Training labels $\mathbf{Y}_L$; Embedding method $\texttt{EMBEDDING} \in \{\textsc{Laplacian}, \textsc{Gosh}\}$
Parameters: dimensionality of $\widehat{\mathbf{U}}$, $\widehat{\mathbf{U}}_{\text{initial}}$ k; number of training epochs T; learning rates η_{label}, η_{graph}, η_{weight}; number of batches K;
Output: Vector representations $\mathbf{h}_v$ for all $v \in \mathcal{V}$;

// Initialize
$\widehat{\mathbf{U}}, s \leftarrow \texttt{INITIALIZE}(\mathbf{A}, \texttt{Emd}, k)$ // (cf. Algorithm 1)
Create K batches of nodes $\mathbb{B}_1 \cup \ldots \cup \mathbb{B}_K \leftarrow \mathcal{V}$
for $t \leftarrow 1$ ***to*** T **do**
  for $b \leftarrow 1$ ***to*** K **do**
    // Forward propagation
    $\forall i \in \mathbb{B}_b, \mathbf{h}_i^{(t)} \leftarrow \sigma\Big(\text{ReLU}\Big(\texttt{CONCAT}\Big[\mathbf{X}_{\mathbb{B}_b}, \widehat{\mathbf{U}}_{\mathbb{B}_b}^{(t-1)}\Big] W^{(0)}\Big) W^{(1)}\Big)$ // (Equation (3))
    // Update $\widehat{\mathbf{U}}$
    $\widehat{\mathbf{U}}_{\mathbb{B}_b}^{(t)} \leftarrow \widehat{\mathbf{U}}_{\mathbb{B}_b}^{(t-1)} - \eta_{\text{label}} \cdot \frac{\partial \mathcal{L}_{\text{label}}}{\partial \widehat{\mathbf{U}}_{\mathbb{B}_b}^{(t-1)}} - \eta_{\text{graph}} \cdot \frac{\partial \mathcal{L}_{\text{graph}}}{\partial \widehat{\mathbf{U}}_{\mathbb{B}_b}^{(t-1)}}$ // (cf. Equations (9)-(10))
    // Update $\widehat{\mathbf{W}}$
    $\widehat{\mathbf{W}}_{\mathbb{B}_b}^{(t)} \leftarrow \widehat{\mathbf{W}}_{\mathbb{B}_b}^{(t-1)} - \eta_{\text{weight}} \cdot \frac{\partial \mathcal{L}_{\text{label}}}{\partial \widehat{\mathbf{W}}_{\mathbb{B}_b}^{(t-1)}}$
  end
end
return $\forall v \in \mathcal{V}, \mathbf{h}_u^T$

4. Experiments

In this section, we empirically evaluate the performance of JANE on synthetic and real-world datasets.

4.1. Algorithms

Variants of JANE. We conducted experiments for four variants of JANE, depending on the choice $\widehat{\mathbf{U}}_{\text{initial}}$ and the training procedure.

- JANE-*R:* This chooses a random matrix of appropriate dimensions ($n \times k$) as $\widehat{\mathbf{U}}_{\text{initial}}$ and trains according to Algorithm 2 in a batched manner.
- JANE-*U:* This computes a GOSH embedding [28] based on a hyperparameter search that minimizes Equation (7) as $\widehat{\mathbf{U}}_{\text{initial}}$ and trains according to Algorithm 2 in a batched manner.
- JANE-*NU:* This computes a GOSH embedding [28] based on a hyperparameter search that minimizes Equation (7) as $\widehat{\mathbf{U}}_{\text{initial}}$. However, $\widehat{\mathbf{U}}_{\text{initial}}$ does not update during training (i.e. ignores Step 7). In effect, this becomes a simple feed-forward multi-layer perceptron.
- JANE-*U-M:* This is similar to JANE-U, except that it trains in a minibatched fashion (cf. Algorithm 3).

Baselines. We compare the performance of JANE with five baselines divided into three broad groups.

- *Graph-structure agnostic algorithms:* Random Forest (RF) [18] trains a random forest on the attribute matrix $\mathbf{X}$ and does not incorporate adjacency information.
- *Node-attribute agnostic algorithms:* Label Propagation (LP) [8] chooses labels for nodes based on the community label of its r-hop neighbors. DeepWalk (DW) [12] encodes neighbourhood information via truncated random walks and uses a Random Forest Classifier for label prediction. These do not incorporate node attributes.
- *Graph-convolution based algorithms:* GCN [31] and GraphSAGE (mean aggregator) [3] convolve over node attributes using the adjacency matrix. We acknowledge that this

is extremely active area of research today and there exist several advancements that demonstrate improved performance in terms of training efficiency [32,33] and low homophily datasets [34,35]. However, GCN and GraphSAGE continue to prove to be strong benchmarks, and so we empirically compare with them to demonstrate our central point: JANE is a simple algorithm that efficiently scales and is competitive with the state-of-the-art on a variety of datasets.

Implementation. We implemented Random Forest using Scikit-Learn [36]. All other baselines and all variants of JANE were implemented using Pytorch Geometric [37]. In all our experiments, as is standard, all approaches only received the adjacency matrix of the graph $\mathbf{A}$ and the node attributes $\mathbf{X}$ as input. We grid searched over the hyperparameter space to find the best setting for all our baselines.

Hardware. We performed all experiments on a Linux machine with 32 cores, 32GB RAM and a NVIDIA A100 GPU.

4.2. Node Classification on Synthetic Data

The goal of these experiments was two-fold—(1) demonstrate the fundamental differences between and limitations of existing classification approaches using synthetic datasets wherein labels derive from only $\mathbf{X}$, only $\mathbf{U}$, or partly from both; and (2) show the strengths and general-purpose nature of JANE vis-à-vis the source of node labels.

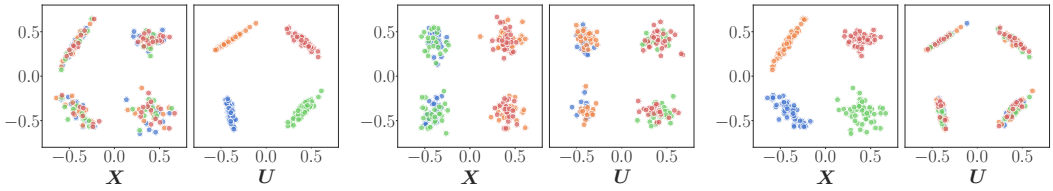
Synthetic Datasets. Figure 3 describes representative synthetic datasets generated according to the model described in Section 3.1. We set the number of individual node features $|\mathbf{X}| = d = 2$ and number of latent features $|\mathbf{U}| = k = 2$. We generated these $\mathbf{X}$ and $\mathbf{U}$ from gaussian distributions for $n = 200$ points, each of which belonged to one of $M = 4$ classes and set the scale $s^2 = 1$. The adjacency matrix was probabilistically created from $\mathbf{U}$ according to Equation (1). An influence parameter, $\alpha \in [0, 1]$, controlled the degree to which node labels derive from $\mathbf{X}$ or $\mathbf{U}$: $\alpha = 0.0$ signifies that they derive only from $\mathbf{U}$ and are independent of $\mathbf{X}$; $\alpha = 1.0$ that they derive only from $\mathbf{X}$ and are independent of $\mathbf{U}$; and $\alpha = 0.5$ that they derive equally from $\mathbf{X}$ and $\mathbf{U}$ (specifically, without loss of generality, only the first feature from $\mathbf{X}$ and $\mathbf{U}$ contributes to label assignment). Figure 3a depicts an instance of $\mathbf{X}$ and $\mathbf{U}$ each for $\alpha = \{0.0, 0.5, 1.0\}$, respectively. The colors of points represent classes. We constructed the adjacency matrix from $\mathbf{U}$ as per Equation (2). Figure 3b depicts an instance of the corresponding graphs.

Implementation Details. We used Scikit–Learn's MAKE_CLASSIFICATION to generate these datasets. The approaches did not have access to α or $\mathbf{U}$. JANE was trained as a two-layer neural network for a maximum of $T = 200$ epochs with dropout of 0.2 for each layer, weight decay of 0.05, and learning rate of 0.005 using Adam. We set the number of eigenvectors as $k = 2$ and chose a scaling factor of $s^2 = 0.01$.

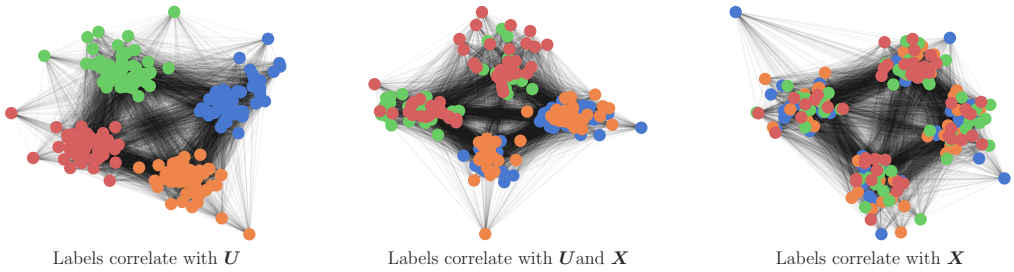
Performance. Figure 3c shows performance as a function of increasing training set sizes.

- $\alpha = 0.0$: LP and DW infer that labels derive from $\mathbf{A}$ (indirectly). GCN converges attribute values of nodes in the same cluster but is not perfectly accurate because $\mathbf{X}$ does not correlate with $\mathbf{Y}$. LANE forces the proximity representation to be similar to the attribute representation, and then smoothens it using the labels. It does not perform well ,since there is no correlation between them.
- $\alpha = 0.5$: LP, DW are able to correctly classify nodes belonging to 2 out of 4 classes, i.e. precisely those nodes whose labels are influenced by $\mathbf{U}$. Conversely, LANE is able to classify those nodes belonging to two classes of nodes that correlate with $\mathbf{X}$. GCN smoothens the attribute values of adjacent nodes, and thus can correctly infer labels correlated with $\mathbf{X}$.
- $\alpha = 1.0$ LP and DW reduce to random classifiers since adjacent nodes do not have similar labels. GCN reduces to a nearly random classifier because, by forcing adjacent nodes with different attribute values to become similar, it destroys the correlation between $\mathbf{X}$ and the labels.

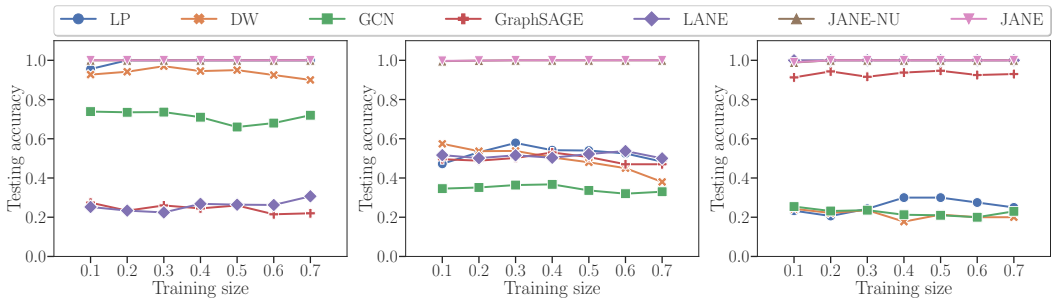
In each of the three cases, JANE-NU and JANE achieve perfect accuracy because they flexibly learn during training whether labels correlate or partially correlate with **X**, **A** (indirectly). While these datasets are simplistic in nature and represent hard cases of homophily and heterophily, it is interesting to find that a simple MLP such as JANE performs well where GCN and GraphSAGE do not. This demonstrates how the homophily assumption—requiring nodes with a similar proximity and attributes to have the same labels—limits the performance of other approaches.



(**a**) Generated **X** and **U**



(**b**) Random graph generated from **U**



(**c**) Classification accuracy

Figure 3. (**a**) depicts three synthetically generated datasets **X** and **U** with class labels **Y** influenced only by **U** ($\alpha = 0.0$, left), partly by **U** and partly by **X** ($\alpha = 0.5$, center), and only by **X** ($\alpha = 1.0$, right). (**b**) shows an instance of the corresponding graph generated from **U** according to Equation (2). (**c**) compares the node classification accuracy of JANE and JANE-NU, with the baselines averaged over five random train-test splits. Reprinted with permission from Springer Nature Customer Service Centre GmbH: Springer, International Conference on Complex Networks and Their Applications, Joint Use of Node Attributes and Proximity for Node Classification, Merchant and Mathioudakis [25], 2021.

4.3. Node Classification on Real-World Data

We seek to understand: (1) to what extent JANE can capture real graph structures and their correlations with node labels, and (2) how well JANE compares with our baselines on these datasets.

4.3.1. Datasets

We evaluated seven real datasets of sizes ranging from 2.5 thousand to 1.5 million nodes. If the original graph was disconnected, we extracted its largest connected component along with the corresponding node attributes and labels (This resulted in different datasets

and performances compared to the results reported in prior works [31,34,38]). Table 1 summarizes the dataset statistics.

- *Citation Networks:* Cora, Citeseer, PubMed [1] represent academic papers as nodes, edges denote a citation between two nodes, node features are 0/1-valued sparse bag-of-words vectors and class labels denote the subfield of research to which the papers belong.
- *Social Networks:* Flickr denotes users of the social media site that post images as nodes, edges represent follower relationships, and features are specified by a list of tags reflecting the interests of the users [39]. The labels used to predict are pre-defined categories of images.
- *Squirrel:* This is a Wikipedia dataset [40] where nodes are web pages, edges are mutual links between pages, features correspond to informative nouns in the pages, and labels are categories based on the average number of monthly views of the page.
- *Yelp [41] and Amazon [38]:* The multi-label classification task in these datasets is to predict the types of business or product categories given customer or buyer reviews and friendship or interaction relationships, respectively.

Table 1. Dataset statistics: number of nodes $|\mathcal{V}|$, number of edges $|\mathcal{E}|$, number of classes, number of features, and edge homophily of the graph, respectively. Edge homophily is defined as the fraction of edges in a graph where both nodes have the same class label. We do not report homophily scores for the multi-label datasets Yelp and Amazon.

Graph	Size		Properties		
	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	Number of Classes	Number of Features	Edge Homophily
Cora	2485	5069	7	1428	0.86
Citeseer	2110	3668	6	3669	0.80
PubMed	19,717	44324	3	500	0.86
Squirrel	5201	401,907	5	2089	0.29
Flickr	89,250	989,006	7	500	0.41
Yelp	703,655	13,927,667	100	300	NA
Amazon	1,066,627	263,793,649	107	200	NA

4.3.2. Experimental Setup

For the citation datasets, we used the same train–validation–test splits as in Yang, et al. [42], minus the nodes, which do not belong to the largest connected component. These comprise of 20 samples for each class and represent 5% of the entire dataset. We use 500 additional samples as a validation set for hyperparameter optimization as per Kipf, et al. [31] to enable fair comparison. For all other datasets, we use the training and validation splits reported in their original works. We evaluate the performance of all approaches on the remaining nodes of the graph. For each dataset, we set $k = 128$ dimensions for $\widehat{\mathbf{U}}$. We perform a grid search over the hyperparameter space defined by hidden dimension in $\{128, 256, 512\}$, dropout in $\{0.0, 0.1, 0.2, 0.3\}$, and learning rates $\{0.00001, 0.0001, 0.001, 0.01\}$ with weight decay set to 0.0005.

4.3.3. Performance Analysis

Table 2 reports the average test micro-F1 accuracy scores for each variant of JANE and the baselines. Values in bold denote the algorithm that performed best for each dataset. We make the following observations: (1) Choosing a good $\widehat{\mathbf{U}}_{\text{initial}}$ is important for classification accuracy. Starting from a random matrix and updating it during training results in poorer performance compared to using the GOSH embedding with or without updates. For instance, JANE-R achieves 58.50% on Cora compared to JANE-U and JANE-NU, which achieve 77.78% and 77.75%, respectively. This trend was observed across datasets. (2) The unsupervised GOSH embedding, by design, preserves adjacency information. Thus, even without further updates to $\widehat{\mathbf{U}}_{\text{initial}}$, it performed well in the classification task across datasets. In particular, we observe that it beats the other algorithms in the case of Citeseer with 66.58%. (3) However, updates to $\widehat{\mathbf{U}}_{\text{initial}}$ during training can improve test accuracy for some

datasets (that is, compared to not updating). This is particularly evident in PubMed where JANE-U obtains 77.56% compared to JANE-NU's 76.70%. This is not just influenced by the properties of the dataset, but also the choice of $\widehat{\mathbf{U}}_{\text{initial}}$ because when using eigenvectors, the performance gains for JANE-U are consistently and significantly higher than those for JANE-NU. (4) JANE-U-M significantly outperforms all other baselines on the larger datasets Yelp and Amazon, with 60.70% and 77.23% accuracy scores, respectively. This represents upto 18.05% and 7.66% improvement over the second best baseline. This is because JANE-U-M updates more frequently, i.e., for each batch in every epoch, and thus learns better. (5) We find that GCN beats other algorithms in the case of the low-homophily dataset Squirrel. JANE-U-M (42.65%) outperforms GraphSAGE (41.44%) on Squirrel, but is unable to correlate similarity of attributes of nodes 1-hop away with labels, which GCN, by way of convolutions, has an increased capture ability.

Table 2. Comparison of test micro-F1 scores with baseline algorithms. Accuracy numbers in blue represent the best model for each dataset. Results are averaged over 10 runs.

Algorithm	Cora	Citeseer	Squirrel	PubMed	Flickr	Yelp	Amazon
RandomForest	56.19 ± 0.2	49.20 ± 0.6	33.31 ± 0.5	73.70 ± 0.8	45.80 ± 0.0	42.55 ± 0.1	59.12 ± 0.1
LabelPropagation	70.53 ± 0.0	64.30 ± 0.0	32.37 ± 0.0	70.50 ± 0.0	46.96 ± 0.0	0.90 ± 0.0	0.02 ± 0.0
DeepWalk	36.74 ± 1.6	33.73 ± 2.2	33.78 ± 1.1	69.34 ± 1.3	51.13 ± 0.1	NA	NA
GCN	77.34 ± 0.3	63.90 ± 0.6	47.44 ± 2.4	76.48 ± 0.2	42.67 ± 0.4	39.58 ± 1.6	12.22 ± 0.1
GraphSAGE	74.97 ± 1.5	58.90 ± 1.2	41.44 ± 0.8	73.72 ± 0.7	51.00 ± 0.6	39.07 ± 3.2	68.23 ± 0.4
JANE-R	58.40 ± 0.8	49.58 ± 0.4	31.39 ± 1.4	51.42 ± 1.9	45.39 ± 0.4	39.03 ± 2.2	63.39 ± 0.6
JANE-NU	77.75 ± 0.3	66.58 ± 0.2	42.32 ± 0.5	76.70 ± 0.7	51.44 ± 0.2	40.98 ± 3.4	69.57 ± 0.2
JANE-U	77.78 ± 0.3	66.40 ± 0.1	41.81 ± 0.2	77.56 ± 0.3	51.49 ± 0.2	38.76 ± 1.2	69.53 ± 0.2
JANE-U-M	78.10 ± 0.1	66.30 ± 0.2	42.65 ± 1.2	77.14 ± 0.6	50.67 ± 0.2	60.70 ± 0.6	77.23 ± 0.1

4.3.4. Ablation Study on Choosing $\widehat{\mathbf{U}}_{\text{initial}}$

As described in Section 3.2, in all results reported to date, our method of choice to obtain $\widehat{\mathbf{U}}_{\text{initial}}$ was to invoke Algorithm 1 with GOSH, and after a grid-search of GOSH hyperparameters, chose the GOSH embedding that leads to the highest $\mathcal{L}_{\text{graph}}$ (Equation (7)). In this passage, we address the question of whether the entire $\mathcal{L}_{\text{graph}}$ expression is necessary to obtain a good initial embedding $\widehat{\mathbf{U}}_{\text{initial}}$, or if it would suffice to use only the edge ($\mathcal{L}_{\text{edge}}$) or non-edge ($\mathcal{L}_{\text{nonedge}}$) information for the same purpose. In Figure 4, we report the test accuracy of JANE-U for three different $\widehat{\mathbf{U}}_{\text{initial}}$, which aim to minimize $\mathcal{L}_{\text{edge}}$, $\mathcal{L}_{\text{nonedge}}$ and $\mathcal{L}_{\text{graph}}$, respectively (cf. Equation (7)). We computed this using a grid search over GOSH's hyperparameters for each loss function. Then, we trained JANE-U as per Algorithm 2 over a grid of training hyperparameters and report best results. We find that, in the case of graphs with high homophily, such as Cora, Citeseer, and PubMed, preserving the overall adjacency information results in better accuracy compared to preserving only edge or non-edge information. On the other hand, for low-homophily graphs such as Squirrel, JANE-U performs similarly when using a $\widehat{\mathbf{U}}_{\text{initial}}$ that minimizes any of the three loss functions with similar observations for Flickr.

4.3.5. Ablation Study on Updating $\widehat{\mathbf{U}}$

Figure 5 shows the change in $\mathcal{L}_{\text{label}}$ and $\mathcal{L}_{\text{graph}}$ before and after training for 200 epochs. We used this to analyze how different variants of JANE learn the tradeoff between fitting to adjacency and label information. For instance, since JANE-NU does not update $\widehat{\mathbf{U}}$ during training, its $\mathcal{L}_{\text{graph}}$ remains constant, but $\mathcal{L}_{\text{label}}$ reduces as expected. For other variants, both $\mathcal{L}_{\text{graph}}$ and $\mathcal{L}_{\text{label}}$ reduce over time. JANE-R starts with a higher $\mathcal{L}_{\text{graph}}$ (e.g., 31.16 vs. 11.78 for other variants of JANE in Cora), but all models begin with the same $\mathcal{L}_{\text{label}}$ because the initial $\mathbf{W}$ matrix of the neural network is random. We find that, for example, in the case of Yelp, the GOSH embedding achieves a stable tradeoff between capturing adjacency and label information, because $\mathcal{L}_{\text{graph}}$ only marginally improves. However, it achieves the best test accuracy (60.70%, Table 2). On the other hand, JANE-U updates $\widehat{\mathbf{U}}_{\text{initial}}$ significantly during training in order to reach a stable tradeoff and good test accuracy. Thus, we conclude that JANE flexibly adapts to various datasets during training.

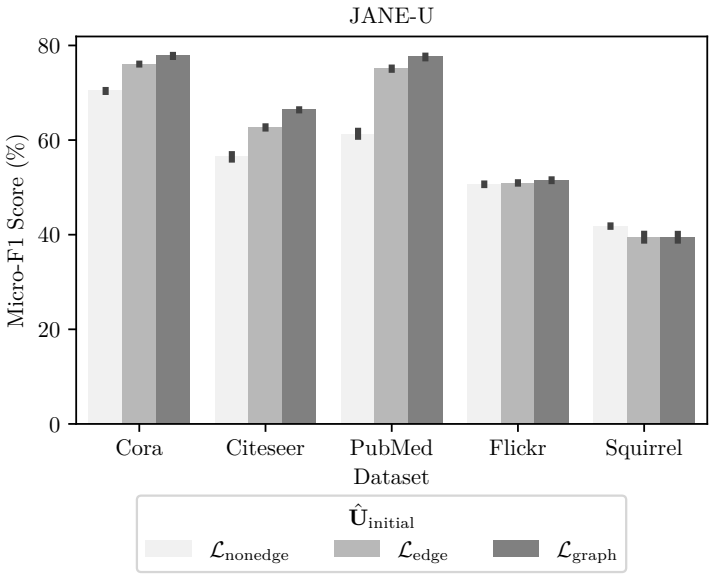


Figure 4. Test micro-F1 scores for JANE-U where $\hat{\mathbf{U}}_{\text{initial}}$ is that which minimizes $\mathcal{L}_{\text{edge}}$, $\mathcal{L}_{\text{nonedge}}$, or $\mathcal{L}_{\text{graph}}$. Jointly preserving edge as well as non-edge information results in best performance.

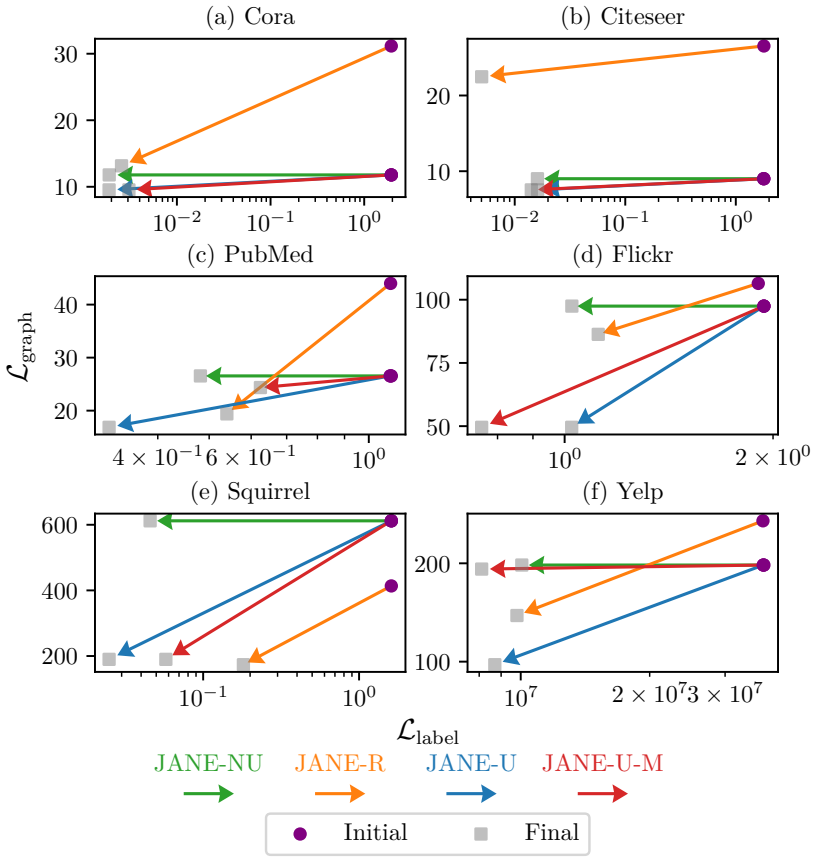


Figure 5. Change in $\mathcal{L}_{\text{label}}$ (X-axis) and $\mathcal{L}_{\text{graph}}$ (Y-axis) due to training for JANE-R, JANE-NU, JANE-U, and JANE-U-M. The purple circle denotes the loss value before the start of training and the grey square denotes the loss value after training for 200 epochs. JANE learns to fit to both the labels and adjacency during training.

4.3.6. Relationship between $\mathcal{L}_{\text{graph}}$ and Accuracy

Figure 6 depicts how $\mathcal{L}_{\text{graph}}$, represented by a blue line, and validation accuracy, represented by a red line, change during the training phase of JANE. Each row corresponds to a specific algorithm (e.g., first row depicts JANE-R on PubMed and Amazon) and each column corresponds to a specific dataset (e.g., first column depicts JANE-R, JANE-U, and JANE-U-M on PubMed). We observe that JANE-R's accuracy continues to improve while $\mathcal{L}_{\text{graph}}$ reduces over time, indicating that it is learning well. After 200 epochs, however, its micro-F1 score (51.42%) remains lower than that of JANE-U (77.56%) for PubMed. For Amazon, even though it starts with a larger $\mathcal{L}_{\text{graph}}$ (7304 vs. 3066), the updates to $\widehat{\mathbf{U}}_{\text{initial}}$ provide it with sufficient information, such that its performance (63.39%) is comparable to JANE-U (69.53%), but not JANE-U-M (77.23%). JANE-U-M learns very quickly and achieves a good accuracy score within a relatively low number of epochs. Lastly, we observe, e.g., in PubMed, that JANE-U and JANE-U-M show diminishing returns in terms of accuracy after reaching a peak at 30 and 87 epochs, respectively, even though $\mathcal{L}_{\text{graph}}$ continues to reduce. This implies that fitting to adjacency is only useful up to a certain extent and indicates that an early stopping criterion based on this may be beneficial to computational efficiency.

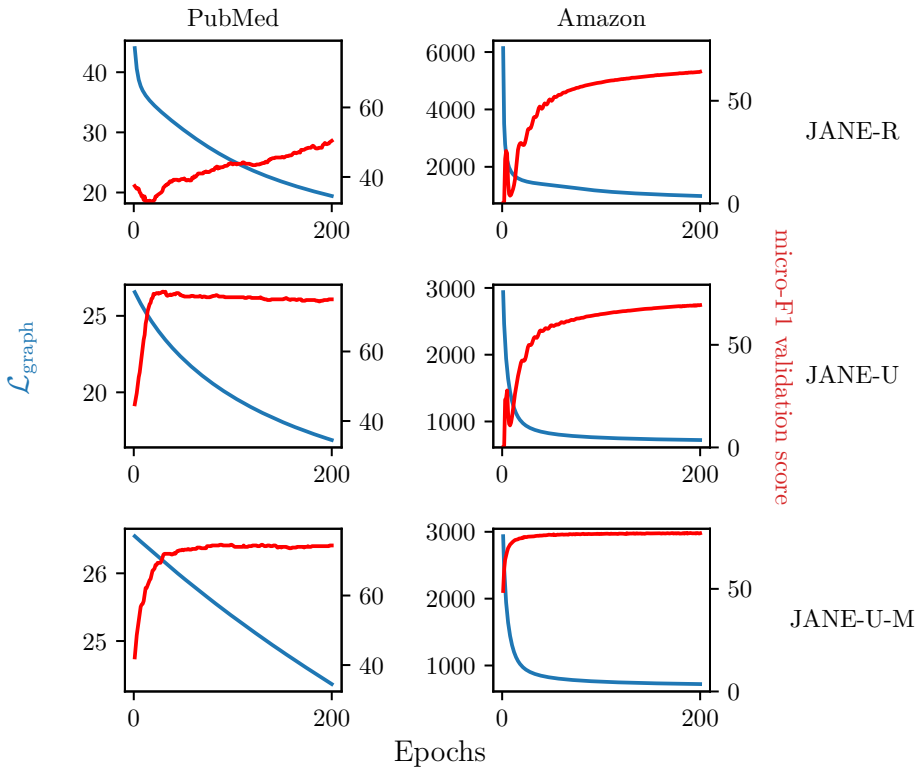


Figure 6. Decrease in $\mathcal{L}_{\text{graph}}$ (blue line, left Y-axis) and increase micro-F1 validation accuracy (red line, right Y-axis) as a function of training epochs (X-axis) for JANE-R, JANE-U, and JANE-U-M on Pubmed and Amazon datasets.

4.3.7. Runtime and Memory

Figure 7 plots the average time for a single training epoch for the various neural-network training algorithms on all datasets. This does not include the time to generate GOSH [28] embeddings, which range from 0.03 s for Cora to 17.6 s for Amazon over 200 epochs. JANE-NU is the fastest, since it is a vanilla feed-forward neural network and does not update $\widehat{\mathbf{U}}_{\text{initial}}$. In practice, this results in a 20× to 333× depending on the size of the dataset. For the small-to-medium-sized datasets such as Cora up to Flickr, JANE-U and JANE-U-M are up to 7× and 2× faster than GCN and GraphSAGE and comparable

or marginally slower on the two large datasets, Yelp and Amazon. However, it should be noted that both GCN and GraphSAGE use neighborhood sampling during the aggregation and inference steps, whereas JANE-U and JANE-U-M compute the full gradients for every batch in each epoch in a serial manner and do not employ a stopping criterion for updating $\widehat{\mathbf{U}}$, which would significantly speed up training. JANE is faster during inference in comparison to GCN and GraphSAGE. Thus, JANE is scalable to large graphs while providing strong performance. Moreover, the peak GPU memory utilization ranges from 54.2 MB to 68 GB for Cora and Amazon, respectively.

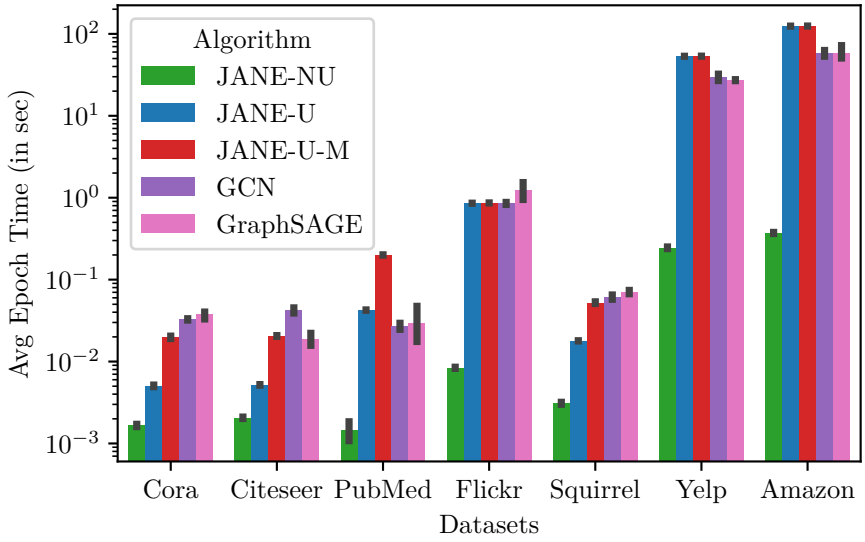


Figure 7. Average training time per epoch (in seconds) of training algorithms on different datasets.

5. Conclusions

In this paper, we developed an approach to node classification that flexibly adapts to settings where labels are strongly correlated to graph structure, on one hand, and to graphs where labels are strongly correlated to node attributes, on the other. We propose a generative framework to demonstrate how graph structural information and node attributes both, can jointly influence node labels Even simple instances of such situations, as shown in Figure 1 and empirically evaluated in Figure 3b, severely affect the performance of standard baselines. Our principled approach, JANE, starts with an initial unsupervised GOSH embedding that captures adjacency information. Then, jointly with attributes, it updates the initial embedding to also incorporate label information for the node classification task, leading to a strong performance in a variety of datasets. Given its simplicity, scalability, and performance, JANE can serve as a competitive algorithm for node classification and as a useful starting point when designing models that holistically account for different sources of node labels and go beyond requiring or enforcing homophily.

There are two main directions for future work. The first is to incorporate node subsampling [38] or graph subsampling [43] in the computation of Equation (7) to further reduce the computational bottleneck of JANE-U and JANE-u-M and scale to very large graphs. The second direction is to implement a distributed version of JANE-U to enable scaling to multi-GPU clusters.

Author Contributions: Conceptualization, A.M. (Arpit Merchant), A.M. (Ananth Mahadevan) and M.M.; Funding acquisition, M.M.; Investigation, A.M. (Arpit Merchant), A.M. (Ananth Mahadevan) and M.M.; Methodology, A.M. (Arpit Merchant), A.M. (Ananth Mahadevan) and M.M.; Software, A.M. (Arpit Merchant) and A.M. (Ananth Mahadevan); Supervision, M.M.; Writing—original draft, A.M. (Arpit Merchant), A.M. (Ananth Mahadevan) and M.M.; Writing—review and editing, A.M. (Arpit Merchant), A.M. (Ananth Mahadevan) and M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the University of Helsinki and the Academy of Finland Projects MLDB (322046) and HPC-HD (347747).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Real-world datasets that we use in our study are publicly available. Code for generating synthetic data and for our experiments is available at https://version.helsinki.fi/ads/jane (25 May 2022).

Acknowledgments: Open access funding provided by the University of Helsinki.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; Eliassi-Rad, T. Collective classification in network data. *AI Mag.* **2008**, *29*, 93. [CrossRef]
2. Zhu, X.; Goldberg, A.B. Introduction to semi-supervised learning. *Synth. Lect. Artif. Intell. Mach. Learn.* **2009**, *3*, 1–130.
3. Hamilton, W.L.; Ying, R.; Leskovec, J. Inductive representation learning on large graphs. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 1025–1035.
4. Yue, X.; Wang, Z.; Huang, J.; Parthasarathy, S.; Moosavinasab, S.; Huang, Y.; Lin, S.M.; Zhang, W.; Zhang, P.; Sun, H. Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics* **2019**, *36*, 1241–1251. [CrossRef] [PubMed]
5. Parisot, S.; Ktena, S.I.; Ferrante, E.; Lee, M.; Guerrero, R.; Glocker, B.; Rueckert, D. Disease prediction using graph convolutional networks: Application to Autism Spectrum Disorder and Alzheimer's disease. *Med. Image Anal.* **2018**, *48*, 117–130. [CrossRef] [PubMed]
6. McPherson, M.; Smith-Lovin, L.; Cook, J.M. Birds of a feather: Homophily in social networks. *Annu. Rev. Sociol.* **2001**, *27*, 415–444. [CrossRef]
7. Marsden, P.V.; Friedkin, N.E. Network studies of social influence. *Sociol. Methods Res.* **1993**, *22*, 127–151. [CrossRef]
8. Zhou, D.; Huang, J.; Schölkopf, B. Learning from labeled and unlabeled data on a directed graph. In Proceedings of the 22nd International Conference on Machine Learning—ICML '05, Bonn, Germany, 7–11 August 2005; ACM Press: New York, NY, USA, 2005; pp. 1036–1043. [CrossRef]
9. Blum, A.; Lafferty, J.; Rwebangira, M.R.; Reddy, R. Semi-supervised learning using randomized mincuts. In Proceedings of the Twenty-First International Conference on Machine Learning, Banff, AB, Canada, 4–8 July 2004; ACM: New York, NY, USA, 2004; p. 13.
10. Joachims, T. Transductive learning via spectral graph partitioning. In Proceedings of the 20th International Conference on Machine Learning (ICML-03), Washington, DC, USA, 21–24 August 2003; pp. 290–297.
11. Belkin, M.; Niyogi, P.; Sindhwani, V. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *J. Mach. Learn. Res.* **2006**, *7*, 2399–2434.
12. Perozzi, B.; Al-Rfou, R.; Skiena, S. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; pp. 701–710.
13. Qiu, J.; Dong, Y.; Ma, H.; Li, J.; Wang, K.; Tang, J. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, Los Angeles, CA, USA, 5–9 February 2018; pp. 459–467.
14. Lorrain, F.; White, H.C. Structural equivalence of individuals in social networks. *J. Math. Sociol.* **1971**, *1*, 49–80. [CrossRef]
15. Sailer, L.D. Structural equivalence: Meaning and definition, computation and application. *Soc. Netw.* **1978**, *1*, 73–90. [CrossRef]
16. Huang, X.; Li, J.; Hu, X. Accelerated attributed network embedding. In Proceedings of the 2017 SIAM International Conference on data mining, Houston, TX, USA, 27–29 April 2017; pp. 633–641.
17. Gao, H.; Huang, H. Deep Attributed Network Embedding. In Proceedings of the IJCAI, Stockholm, Sweden, 13–19 July 2018.
18. Huang, X.; Li, J.; Hu, X. Label informed attributed network embedding. In Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, Cambridge, UK, 6–10 February 2017; pp. 731–739.
19. Wu, J.; He, J.; Xu, J. Net: Degree-specific graph neural networks for node and graph classification. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 406–415.
20. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *arXiv* **2017**, arXiv:1710.10903.
21. Li, Q.; Han, Z.; Wu, X.M. Deeper insights into graph convolutional networks for semi-supervised learning. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.

22. Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In Proceedings of the 2018 Conference on Neural Information Processing Systems NeurIPS 2018, Montreal, QC, Canada, 3–8 December 2018; pp. 4800–4810.
23. Wang, X.; Zhu, M.; Bo, D.; Cui, P.; Shi, C.; Pei, J. Am-gcn: Adaptive multi-channel graph convolutional networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Virtual Event, 6–10 July 2020; pp. 1243–1253.
24. Xiao, S.; Wang, S.; Dai, Y.; Guo, W. Graph neural networks in node classification: survey and evaluation. *Mach. Vis. Appl.* **2022**, *33*, 1–19. [CrossRef]
25. Merchant, A.; Mathioudakis, M. Joint Use of Node Attributes and Proximity for Node Classification. In *Proceedings of the International Conference on Complex Networks and Their Applications*; Springer: Cham, Switzerland, 2021; pp. 511–522.
26. Belkin, M.; Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.* **2003**, *15*, 1373–1396. [CrossRef]
27. Chung, F.R.; Graham, F.C. *Spectral Graph Theory*; American Mathematical Society: Providence, RI, USA, 1998; Number 92.
28. Akyildiz, T.A.; Alabsi Aljundi, A.; Kaya, K. Gosh: Embedding big graphs on small hardware. In Proceedings of the 49th International Conference on Parallel Processing, Edmonton, AB, Canada, 17–20 August 2020; pp. 1–11.
29. Tsitsulin, A.; Mottin, D.; Karras, P.; Müller, E. Verse: Versatile graph embeddings from similarity measures. In Proceedings of the 2018 World Wide Web Conference, Lyon, France, 23–27 April 2018; pp. 539–548.
30. Rahman, M.K.; Sujon, M.H.; Azad, A. Force2Vec: Parallel force-directed graph embedding. In Proceedings of the 2020 IEEE International Conference on Data Mining (ICDM), Sorrento, Italy, 17–20 November 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 442–451.
31. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In Proceedings of the International Conference on Learning Representations (ICLR), Toulon, France, 24–26 April 2017.
32. Cong, W.; Forsati, R.; Kandemir, M.; Mahdavi, M. Minimal Variance Sampling with Provable Guarantees for Fast Training of Graph Neural Networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Virtual Event, 6–10 July 2020; pp. 1393–1403.
33. Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *arXiv* **2020**, arXiv:2005.00687.
34. Pei, H.; Wei, B.; Chang, K.C.C.; Lei, Y.; Yang, B. Geom-gcn: Geometric graph convolutional networks. *arXiv* **2020**, arXiv:2002.05287.
35. Zhu, J.; Yan, Y.; Zhao, L.; Heimann, M.; Akoglu, L.; Koutra, D. Generalizing graph neural networks beyond homophily. *arXiv* **2020**, arXiv:2006.11468.
36. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
37. Fey, M.; Lenssen, J.E. Fast graph representation learning with PyTorch Geometric. *arXiv* **2019**, arXiv:1903.02428.
38. Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; Prasanna, V. GraphSAINT: Graph Sampling Based Inductive Learning Method. In Proceedings of the International Conference on Learning Representations, Virtual Conference, 26–30 April 2020.
39. Li, J.; Hu, X.; Tang, J.; Liu, H. Unsupervised streaming feature selection in social media. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, Melbourne, Australia, 19–23 October 2015.
40. Rozemberczki, B.; Allen, C.; Sarkar, R. Multi-scale attributed node embedding. *J. Complex Netw.* **2021**, *9*, cnab014. [CrossRef]
41. Shchur, O.; Mumme, M.; Bojchevski, A.; Günnemann, S. Pitfalls of graph neural network evaluation. *arXiv* **2018**, arXiv:1811.05868.
42. Yang, Z.; Cohen, W.; Salakhudinov, R. Revisiting semi-supervised learning with graph embeddings. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 40–48.
43. Zeng, H.; Zhang, M.; Xia, Y.; Srivastava, A.; Malevich, A.; Kannan, R.; Prasanna, V.; Jin, L.; Chen, R. Decoupling the Depth and Scope of Graph Neural Networks. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 19665–19679.