Received September 23, 2021, accepted October 16, 2021, date of publication October 21, 2021, date of current version October 29, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3122100

JONNEE: Joint Network Nodes and Edges Embedding

ILYA MAKAROV[1,2,3], KSENIA KOROVINA[1], AND DMITRII KISELEV[1,3]
[1]HSE University, 101000 Moscow, Russia
[2]University of Ljubljana, 1000 Ljubljana, Slovenia
[3]Artificial Intelligence Research Institute (AIRI), 105064 Moscow, Russia

Corresponding authors: Ilya Makarov (iamakarov@hse.ru) and Dmitrii Kiselev (dkiseljov@hse.ru)

This work was supported by HSE University.

ABSTRACT Recently, graph embedding models significantly improved the quality of graph machine learning tasks, such as node classification and link prediction. In this work, we propose a model called JONNEE (JOint Network Nodes and Edges Embedding), which learns node and edge embeddings under self-supervision via joint constraints in a given graph and its edge-to-vertex dual representation as a Line graph. The model uses two graph autoencoders with additional structural feature engineering and several regularization techniques to train for an adjacency matrix reconstruction task in an unsupervised setting. Experimental results show that our model performs on par with state-of-the-art undirected attribute graph embedding models and requires less number of epochs to achieve the same quality due to Line graph self-supervision under a unified embedding framework.

INDEX TERMS Graph machine learning, graph neural networks, line graph, link prediction, network embedding, network representation learning, node classification.

I. INTRODUCTION

Networks appear in many real-world tasks that require describing important relations between objects and their attributes. Effective network representation provides valuable information on how graph structure can improve understanding and feature extraction of structural information accompanied by non-network features.

Nowadays, machine learning methods demand the representation of information in the vectorized form to follow standard frameworks for classic machine learning tasks. Graph machine learning is usually associated with particular node-level machine learning problems, such as node classification, link prediction, and node clustering, followed by network visualization. In order to feed graph data to machine learning frameworks, network embedding emerges as an effective and efficient approach to solve machine learning problems on networks. It maps graph motifs, such as nodes and edges, into a low-dimensional space while preserving certain graph information and constraints related to the graph.

Today, there exists a large variety of graph embeddings that automatically extract vector representation for networks [1]–[8]. All network embeddings have different training and inference complexity, various construction ideas, and diverse applications to network data domains. However, the concept of network representation learning allows to combine them in a general pipeline verifying in terms of the quality of graph machine learning tasks on benchmark networks.

Different models have their own advantages and shortcomings. For example, sequence-based models can be efficiently trained for large graphs but do not take into account features in nodes and edges. Adjacency matrix factorization frameworks have high quality while being very costly and slow to train. Models based on deep learning methods can efficiently incorporate features but are hard to train and interpret in terms of which factors and neighbors impact model performance the most.

Moreover, all of these models mostly rely on node embeddings for constructing edge embeddings: while node embeddings are trained, edge embeddings used for link prediction task are usually presented by certain symmetric operators of incident node embeddings [9] or their low-dimensional bi-linear representation [10]. Such an approach does not consider the noise in the edge data, nor does it describe the fact that edge formation and features may be obtained

The associate editor coordinating the review of this manuscript and approving it for publication was Vicente Alarcon-Aquino.

independently from graph structure and jointly learned under the self-supervised framework.

In our study, we aim to construct a model that combines the best of all three classical approaches for network embeddings under Line graph self-supervision: sequence-based models for fast feature generation, capturing structural information and high-order dependencies; deep learning model autoencoder for its expressiveness and effective use of generated features; and matrix factorizations for simplicity in model regularization.

A new model embeds nodes and edges into common vector space as latent codes of the original graph and Line graph autoencoders (the similar idea was used in [11] and [12]) with shared loss function similar to proposed in [13]–[15] edge embedding operators for "pooling" the first-order neighborhood of source and target nodes.

Our goal is to learn better node representations by using an edge-to-node direction and explicitly regularizing a node embedding to be close to the average of the embeddings of the edges incident to it as an objective instead of using deterministic mapping. This idea expresses that a node should be not only a center of its node neighborhood, but also a center of its dual edge neighborhood, and explicitly support edge reconstruction from incident nodes as their local averages based on Line graph representation.

Since our focus is on the idea that edges may include additional information that could be presented not only by incident node embeddings alone, we first focus on the link prediction problem to guide our model evaluation. The link prediction problem is one of the core machine learning tasks on graphs, which can be solved by our model better than by other state-of-the-art solutions. Secondly, we aim to test JONNEE model on multi-class node classification problem under unsupervised and semi-supervised settings. Finally, JONNEE embeddings were used for demonstrating superior quality on network visualization problem.

Overall, we show that JONNEE is a viable model that often outperforms state-of-the-art unsupervised solutions for classical machine learning problems on graphs.

In this work, the following contributions were made:

1) Novel JONNEE model on a joint network node and edge embedding via Line graph self-supervision is suggested.
2) Extensive experiments show the importance of each component in JONNEE model on the resulting quality of the finetuned model.
3) A comprehensive comparison of existing methods is performed on node classification, link prediction, and network visualization problems. Our model show equal quality compared to state-of-the-art models while converging faster in an unsupervised setting.

The rest of the paper is organized as follows. We start by defining the problem field and reviewing existing network embedding models in Section II. Section III contains a detailed description of the proposed model and the motivation behind each component, followed by the proofs of their importance in Section IV. Next, the finetuned model was compared to state-of-the-art network embeddings in Section V. Section VI discusses experiment results, main takeaways, and future work.

II. RELATED WORK

We start by introducing essential definitions and establishing general notation for our paper related to network science and network representation learning.

Let us consider a graph $G = (V, E)$ defined as a set of vertices V and a set of edges $E \subseteq V \times V$ Additionally, certain graph substructures may be equipped with attributive features conveying information that cannot be expressed by graph structure alone. For example, each node of a citation network is a paper that may have a vector description of its textual or semantic content. If these features are available, we denote them as $X_0 \in \mathbb{R}^{|V| \times d_0}$, where d_0 is a dimension of original space for node attributive features.

The procedure for constructing a vector representation of a graph is called graph embedding, which is a mapping from a collection of substructures (most commonly, these include either all nodes, all edges, or certain subgraphs) to $\mathbb{R}^d$. We will mostly consider node embeddings: $f : V \rightarrow \mathbb{R}^d$, $d \ll |V|$.

For many graph-based tasks, the most natural task formulation is unsupervised learning: this is described as the case when it is required to learn embeddings using only the adjacency matrix A containing information on structural similarity and, possibly, features X_0 without task-specific loss part. Another possible case is that there are labels available for some graph substructures, and one wishes to recover missing labels in a semi-supervised approach. One example of this is the node classification task, in which all nodes are available from the start, but only a fraction is labeled.

In order to describe network embedding definitions and training settings, it is important to present a clarification of what is meant by good network embedding. During the embedding procedure, one should aim to compress the data while retaining most of the essential information about direct node similarities, and at the same time extract important features from the structural information usually described by high-order node proximities, which is some similarity measure over node-related graph substructures, e.g. neighborhoods. Higher-order proximities can be defined similarly, usually at a higher computational cost. Apart from the cosine measure usually adopted to quantify similarity, other metrics like Rooted PageRank, Katz Index, Common Neighbors, Jaccard coefficient, Adamic Adar can also be used (see [16] for more details).

Finally, when the task of constructing efficient node embeddings is stated, let us describe the main approaches for defining models' architectures and optimization frameworks and then describe the improvements suggested for edge embeddings.

In order to describe the rapidly growing field of network embeddings, we use one of the possible taxonomies covering the idea of network embedding construction via optimization problem. There are usually three general categories related to matrix factorizations, sequential models, and graph neural networks. Below, we provide a brief overview of existing approaches for each category following the detailed survey in [8] and review existing edge embedding models constructed over node embeddings which leads to the main motivation behind our model.

A. MATRIX FACTORIZATION AND LAPLACIAN METHODS

Starting with matrix factorization methods, these models usually factorize large adjacency matrix with a product of two matrices containing network and context representations. Factorization models are common techniques for approaching a dimension reduction problem in many domains.

Some methods directly decompose the adjacency matrix A [17]–[19] or the graph proximity matrix [20]–[22].

The goal for Laplacian Eigenmaps [23]–[25] class of models lies in preserving first-order similarities via representing each node by graph Laplacian eigenvectors associated with its first k nontrivial eigenvalues.

Thus, using graph Laplacian, a model gives a more significant penalty if two nodes with larger similarity are embedded far apart in the embedding space.

Another approach is to directly factor the node proximity matrix into a product of embedding F and context F_c matrices, with symmetric models using only F (e.g., GraRep [26]), or asymmetric models (e.g., HOPE [16]) concatenating representations from F and F_c rows.

In most cases, factorization approaches are not able to learn on which order proximities to focus attention, require a transductive learning paradigm and have high computational complexity.

B. SEQUENCE-BASED APPROACHES

Overcoming limitations on time complexity for the factorization models, sequence-based embeddings use different random walk strategies to maximize the probability of observing sampled neighborhood (context) of a node given its embedding.

The idea is to maximize the probability of observing the neighborhood of a node given its embedding similar to Skip-gram model [27]. The most well-known random walk based graph embeddings are DeepWalk [28] and node2vec [9].

There are also other models inspired by factorization models incorporating high-order proximities [29]–[31], preserving community structure [32]–[34] or taking into account attention on sampled walks [35], [36].

One of the most efficient sequential models based on graph Laplacian approximation and sampling strategy around the node is based on sampling neighborhood using graph diffusion presented in diff2vec [37].

In general, sequence-based models are efficient for representing structural information alone. They can be used for feature engineering in conjunction with other models, taking node and edge attributes into account.

C. GRAPH NEURAL NETWORKS

Graph neural networks incorporate graph structure into classic deep learning models, thus connecting neural networks and automated graph feature engineering.

Before the beginning of the deep learning era, graph signal processing techniques using graph spectral decomposition were suggested in [38] and [39]. Advances in deep learning allow applying of deep neural networks to graph data [40]–[44].

In [45], authors propose Graph Convolutional Layer that further simplifies approximation to spectral convolution and achieves improved computational efficiency for semi-supervised multi-class node classification. A model combining several such convolutions is referred to as Graph Convolutional Network (GCN). Improvements over speed and optimization of training GCNs were suggested in [46]–[48]. Different convolutions via spatial or graph spectral methods were suggested in [49]–[58].

Similar to the other fields, autoencoders were proposed to train unsupervised node-level representations [59]. Graph Autoencoder (GAE) is based on several graph convolutional layers to encode graph structure and inner-product decoder to reconstruct the adjacency matrix. Variational graph autoencoder (VGAE) [59] extends the previous model with different GCN encodings for mean and standard deviation.

In recent work, authors of GraphSAGE [60] offer an extension of GCN for inductive unsupervised representation learning via trainable aggregation functions instead of simple convolutions applied to neighborhoods in GCN. GraphSAGE learns aggregation functions for a different number of hops applied to sampled neighborhoods of different depths, which are then used for obtaining node representations from initial node features.

Another direction of research is to extend attention from random walks and factorization techniques. GAT model [61] uses masked self-attention layers for learning weights balancing the impact of neighbors on node embedding and supporting both inductive and transductive learning settings.

Recent deep learning based models readily exploit node features when those are available are flexible and generic but still least understood and not as efficient as conventional Euclidean deep learning models with independent samples in terms of optimization.

D. EDGE EMBEDDINGS

There exists a variety of models used to construct edge embeddings along with downstream tasks involving graphs and their embeddings. HARP [31] incorporates several hierarchical layers, and constructs node embedding from edge embedding. Considering attribute networks, CANE [62] and LANE [63] directly incorporate edge features and labels into network embedding. Multi-edge network embedding for event graphs (in which an event is described by several edges) was presented in HEBE [64]. Interestingly, Knowledge Graph (KG) completion solves link prediction between entities [65], however these methods are not applicable to homogeneous networks with one type of edges. Approaches for component-wise edge embedding operators based on node embeddings were suggested in many research papers [9], [10], [13], [15], [66]–[68] while other studies just concatenate node embeddings [45], [60], [69].

Recently, self-supervised graph machine learning arose as a popular direction on network embeddings via improving models under limited data constraints. It aims to use different low-level augmentation strategies for training model under self-supervised setting [70], [71]. However, there is no approach to include both node and edge embeddings directly under one optimization problem.

Definition 1 (Dual Graph): For a graph $G = (V, E)$ defined as a set of vertices V and a set of edges $E \subseteq V \times V$ we denote by $G^* = (V^*, E^*)$ a line (dual) graph, the nodes of which are the edges of G and edges are nodes, in the sense that two adjacent nodes are connected by an edge if corresponding edges have the incident node.

Remark 1: Of course, by "dual" we mean "edge-to-vertex dual" or congruent graph, also known as Line graph. The "edge-to-vertex" duality concept is preserved throughout the paper as it gives additional motivation for our graph representation. There is also correspondence, albeit not a bijective one, between nodes of G and edges of G^*. Finally, it is important to mention that only simple undirected graphs without loops and multiple edges will be considered for our study.

The Line graph usage was suggested for joint node and edge structure learning in Dual-Primal GCN [12] and ELAINE [72] in semi-supervised learning setting. However, these models were not applicable to any undirected weighted networks and graph machine learning tasks.

With the present work, we propose a model that combines the best of three core approaches for node embeddings: sequence-based models for fast feature generation, capturing structural information and high-order dependencies; deep learning model autoencoder for its expressiveness and effective use of generated features; and matrix factorizations for simplicity of model regularizing. In addition, self-supervision in matrix reconstruction task via Line graph embedding is a novel idea, which is also one of the main contributions of our work.

III. MODEL: DUAL GRAPH AUTOENCODER

A novel approach is required to address the problems discussed above and further improve upon the existing models. We propose a model of Dual Graph Autoencoder, which learns JOint Nodes and Edges Embedding, called JONNEE. The model name was chosen specifically to distinguish it from another approach presented in the paper on Dual-GCN [73], in which duality concept stands for local and global network features. Our model is designed to be especially effective when only structural information is available, but it also accommodates node and edge weights and features. The model is shown in Figure 1.

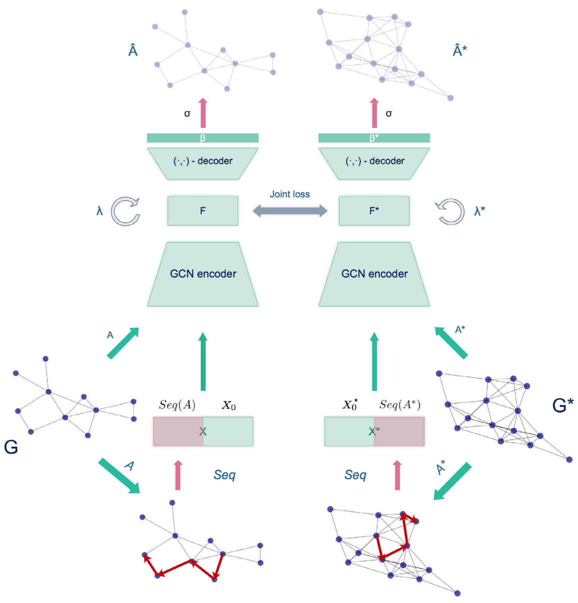


FIGURE 1. Proposed JONNEE model for learning unsupervised embeddings.

We start by describing the model and its essential components. Learning in JONNEE proceeds in two steps.

1) Feature learning: during this stage, features for the graph and its dual graph are generated as embeddings through a sequence-based method.
2) Joint learning of two autoencoders on a graph and on its dual graph, learning both representations for nodes and edges simultaneously and in a consistent manner.

Remark 2: With minor modifications, both steps can be combined in an end-to-end pipeline. Feature generation may be skipped or substituted with end-to-end modeling random walks with graph neural network [36]. As with traditional autoencoders, our model has a deterministic version as well as its probabilistic analog. To keep the exposition simple, we focus on describing the deterministic GAE-based option here, implying that the extension to VGAE-based setting is carried out in the same manner as described in Section III-B and is thus straightforward.

As input to JONNEE, a graph $G = (V, E)$, represented with an adjacency matrix $A \in \mathbb{R}_+^{|V|\times|V|}$ (binary or non-negative valued for weighted graphs) and, optionally, node attributes $X_0 \in \mathbb{R}^{|V|\times d_0}$ and edge attributes $X_0^* \in \mathbb{R}^{|E|\times d_0^*}$ are available. Prior to learning, we obtain the graph dual $G^* = (E, E')$, $E' \subseteq E \times E$ in the form of its dual adjacency matrix. In the basic case, this graph just incorporates structural information and is thus binary. However, it can be weighted as well: for example, one can introduce node weights as averages of edge weights to use as weights for dual edges that correspond to these nodes. We leave experimental testing of this idea for future work in this area as suggested in [13] studying first-order proximity link embedding operators. If the graph is weighted, its weights are normalized to be bounded by 1 just using a normalized adjacency matrix $A_{\text{degree}}^{-\frac{1}{2}} \cdot A \cdot A_{\text{degree}}^{-\frac{1}{2}}$.

The structure of the model description Section III is as follows. We start with a discussion feature learning from sequential models and the idea behind this process. Next, we discuss graph autoencoders for the graph and its Line graph. After the model is determined, we explain the loss function and the regularization tricks used in it. Finally, we discuss the training procedure and its convergence in the unsupervised setting and also present JONNEE semi-supervised counterpart with natural modifications from the original model.

A. FEATURE LEARNING

During the first stage, the algorithm linearizes the graph using an algorithm $Seq \in \{\text{node2vec}, \text{diff2vec}\}$, which can be either random walks or diffusion. Seq learns features $X_1 = Seq(G, u)$ and $X_1^* = Seq(G^*, u^*)$ of a size u for G and u^* for G^* separately. If node and dual node (edge) features X_0 and X_0^* are available from the outset, they are concatenated with learned sequence embeddings described below to form augmented feature sets:

$$X = [X_0, X_1] \in \mathbb{R}^{V\times D},\ \ X^* = [X_0^*, X_1^*] \in \mathbb{R}^{E\times D^*} \qquad (1)$$

In (1) we denote as $D = d_0 + u$ the total output number of features per node and $D^* = d_0^* + u$ the total output number of features per edge. We will assume that the same augmented feature dimensionality u is sufficient to learn sequence embeddings from both G and G^*, that is, $u^* = u$. In our experiments, we set both node2vec and diff2vec parameters to default values proposed in the reference implementations (see [9] and [37]), and leave parameter tuning for future work.

As has been discussed in Section II, informative node features can improve learning representations, especially for graph convolutional networks and graph autoencoders that employ a sequence of graph convolutions as an encoder. Indeed, these models could be interpreted to operate on node features, effectively passing messages between them to update a feature based on convolving its local neighborhood until they are sufficiently consistent. This means that a good initial approximation will most likely not only speed up convergence but also ensure a significantly better local optimum [74].

One way to generate such an approximation is to employ a more lightweight algorithm for learning embeddings. For this purpose, we use sequence-based algorithms (node2vec and diff2vec), as they have an essential property for light initialization, which is the lowest complexity among all the other models described. Moreover, there is an established intuition that ensembling, which we implicitly perform in the model, works better if it combines sufficiently different models. That supports our decision to take sequence-based algorithms that rely on node contexts and sampling to complement graph convolutions. Experiments show the importance of such a feature generation procedure.

Another goal is to explore the best way for feature generation in various settings: for weighted and unweighted graphs, for graphs with and without additional node features. We observe that although GAE works with any adjacency matrix, not necessarily a binary-valued one, it is helpful to make use of edge weights during feature learning as well. While node2vec supports node weights, we found out that the original diff2vec does not, with diffusion from a given point being performed uniformly over neighbors. We generalize this model by weighting the choice over neighbors, thus making diffusion more likely in the direction of heavier or wider edges.

B. GRAPH AUTOENCODERS

Following feature generation, the second stage receives as input the data A, X, and dual data A^*, X^*. Two graph autoencoders GAE and GAE^* are then trained on this data to learn hidden representations $F \in \mathbb{R}^{|V|\times d}$ and $F^* \in \mathbb{R}^{|E|\times d}$. These representations should have the same dimensionality d, so that node and edge embeddings reside in the same vector space. Autoencoders have the same simple architecture that consists of an encoder GCN with two graph convolutional layers, regularized with dropout, and a β-masked inner-product decoder obtaining reconstruction $\widehat{A}$ of the adjacency matrix A from graph embedding representations.

A two-layer GCN architecture is defined as

$$H_1 = \text{ReLU}(\widetilde{A}_{\text{degree}}^{-\frac{1}{2}}\widetilde{A}\widetilde{A}_{\text{degree}}^{-\frac{1}{2}}XW_1) \qquad (2)$$

$$H_2 = \text{ReLU}(\widetilde{A}_{\text{degree}}^{-\frac{1}{2}}\widetilde{A}\widetilde{A}_{\text{degree}}^{-\frac{1}{2}}H_1W_2) \qquad (3)$$

where $\widetilde{A} = A + I, A_{\text{degree}_{ii}} = \sum_j \widetilde{A}_{ij}$ and $\Theta = \{W_1 \in \mathbb{R}^{D\times h}, W_2 \in \mathbb{R}^{h\times d}\}$ are trainable parameters of $GCN(X, A)$.

Following equations (2)-(3) graph and dual graph embeddings and reconstructions are specified as:

$$F = GCN(X, A), \quad \widehat{A} = \sigma\left(FF^T \odot B\right) \qquad (4)$$

$$F^* = GCN(X^*, A^*), \quad \widehat{A^*} = \sigma\left(F^*F^{*T} \odot B^*\right) \qquad (5)$$

In (4)-(5) $\odot$ denotes the Kronecker (point-wise) product of matrices. Matrices $B = \left(1 + \beta a_{ij}\right)_{i,j}$ and $B^* = \left(1 + \beta^* a_{ij}^*\right)_{i,j}$ with $\beta, \beta^* > 0$ allow us to soften learning and regularize the model, especially on unweighted graphs.

This defines the forward pass of JONNEE model. Below, we discuss the choice of GCN architecture and then go into details of training and regularization procedures.

1) TRAINING WITH β-MASKING

Initially, an opposite idea was proposed in [43], where authors weigh higher the actual positive-class terms in MSE reconstruction loss in order to penalize their model more for missing a possible connection than for incorrectly predicting an absent edge as likely. In VGAE, this is implemented by sampling the positive class multiple times until the balance is corrected. However, in our case, β-masking worked better with respect to threshold-independent metrics, such as F1-score, by fitting the model more loosely on the provided adjacency matrix.

2) CHOICE OF THE ENCODER ARCHITECTURE

In our experiments, we consider an encoder composed of two GCN layers. Although the depth of this model could vary, we consider a shallow model is meaningful in many real-world applications for graphs. Compared to images, we have no regular filter structure and are struggling with the computational hardness of considering distant relations between nodes in a graph, which are not meaningful in many practical scenarios. For example, a well-known social graph rule of six handshakes states that any person knows any other person through an average of five other people; in most actual cases of local social networks, this number (effective graph diameter) is even smaller. Thus, any number of encoding layers over the graph diameter would be redundant. We also assume that the number of edges is linear to the number of nodes in most large real-world networks. So it is reasonable to reduce the number of hyperparameters and take the same number of first-layer hidden units for primal and dual encoders.

C. TRAINING

1) GRAPH RECONSTRUCTION LOSS

As reconstruction loss function, we employ mean squared error (MSE), which is proportional to Frobenius matrix norm of the error:

$$L_G = \|A - \widehat{A}\|_F^2 \sim \frac{1}{|V|^2} \sum_{i,j} (\hat{a}_{ij} - a_{ij})^2 \tag{6}$$

$$L_{G^*} = \|A^* - \widehat{A^*}\|_F^2 \sim \frac{1}{|E|^2} \sum_{i,j} (\hat{a}^*_{ij} - a^*_{ij})^2 \tag{7}$$

In the case of an unweighted graph, one can also use the standard cross-entropy loss function instead of MSE used in (6)-(7).

2) JOINT LEARNING

Let us denote again as f and f^* the node and edge representation mappings learned by GAE and GAE^* respectively, and matrices $F \in \mathbb{R}^{|V| \times d}$ and $F^* \in \mathbb{R}^{|E| \times d}$ as their training set images. Now, we describe how the learning of two models is coupled, so that node representations benefit from edge representations and vice versa. We suggest that a good representation for a node should be similar to averages across learned representations for edges adjacent to this node, expressing the idea that these edge representations have in common. Furthermore, on the other hand, an edge representation is similar to what its adjacent nodes share as representations learned by the primal autoencoder. Informally, we want to achieve some form of meaningful arithmetic for edge and node representations, akin to that learned by word2vec: by averaging over edges adjacent to a node, we ideally would extract some common features. Another way to describe this is to note that passing to a dual graph twice (from nodes to edges and then back to nodes combining edges so that $G^{**} = G$) should not affect their representation too much since we wish the edge representations to retain as much information about nodes as possible, and vice versa.

Based on this idea, we add dual terms to our loss function:

$$L_{G^* \to G} = \sum_{v \in V} \|f(v) - \frac{1}{|N^G(v)|} \sum_{u \in N(v)} f^*((u, v))\|_2^2 \tag{8}$$

$$L_{G \to G^*} = \sum_{e=(u,v) \in E} \|f^*(e) - \frac{\sum_{t \in N^G(u) \cup N^G(v)} f(t)}{|N^{G^*}(e)|}\|_2^2 \tag{9}$$

In (8)-(9) $f(v)$ stands for embedding of node v in graph G, $f^*((u, v))$ stands for embedding of an edge $e = (u, v)$ in the dual graph G^*, and $N^{G'}(w)$ denotes neighborhood of a node w in corresponding graph G', which is either G or G^* in our case.

For convenience of setting up the model, we observe that $L_{G^* \to G}$ can be re-written in matrix form, using the incidence matrix $M \in \mathbb{R}^{|V| \times |E|}$, where each element is defined as $M_{u,e} = \mathbb{I}\{\text{node } v \text{ is adjacent to edge } e\}$, then

$$L_{G^* \to G} = \|F - A^{-1}_{\text{degree}} M F^*\|_2^2$$

3) LAPLACIAN REGULARIZATION

We use typical network problem structure knowledge and add some tricks to regularize learned representations and make them more robust. Namely, we draw inspiration from previous researches, such as [43] and [75], and add Laplacian regularization to both node and edge representations.

Intuitively, it seems plausible that a model that places more emphasis on first-order proximity (through direct linkage) is simpler than a model that is more reliant on neighborhood similarities for second-order proximities. Thus, a way to regularize our model and enforce preserving the direct relationship is to add the Laplacian term, with a coefficient controlling the strength of a regularizer. We impose this regularization on both primal and dual GAE by adding appropriately weighted losses L_{reg} and L^*_{reg}, defined as follows:

$$L_{reg}(F) = \sum_{i,j} a_{ij} \left\| \frac{F_i}{d_i} - \frac{F_j}{d_j} \right\|^2 = 2\ \mathrm{tr}(F^T \widetilde{L} F) \tag{10}$$

$$L^*_{reg}(F^*) = \sum_{i,j} a^*_{ij} \left\| \frac{F^*_i}{d^*_i} - \frac{F^*_j}{d^*_j} \right\|^2 = 2\ \mathrm{tr}(F^{*T} \widetilde{L}^* F^*) \tag{11}$$

In (10)-(11) d_i, d^*_i are node degrees in G and G^*, ^{T} stands for transposition, and $\widetilde{L} = I - A^{-\frac{1}{2}}_{\text{degree}} \cdot A \cdot A^{-\frac{1}{2}}_{\text{degree}}$, $\widetilde{L}^* = I - A^{*-\frac{1}{2}}_{\text{degree}} \cdot A^* \cdot A^{*-\frac{1}{2}}_{\text{degree}}$ are corresponding normalized Laplacians.

4) FINAL MODEL

We combine individual GAE loss functions (6)-(7), individual regularizers (10)-(11) and joint components (8)-(9) into the following loss function:

$$\begin{aligned} L(\Theta, \Theta^*) &= L_G(\Theta) + L_{G^*}(\Theta^*) \\ &\quad + \lambda L_{reg}(\Theta) + \lambda^* L^*_{reg}(\Theta^*) + C \cdot L_{G^* \to G}(\Theta, \Theta^*) \\ &\quad + C^* \cdot L_{G \to G^*}(\Theta, \Theta^*) \end{aligned} \tag{12}$$

In our experiments, we have only tested out adding one term $L_{G^* \to G}$, due to computational limitations on large graphs. Thus we omit the $L_{G \to G^*}$ term from descriptions in this work, making $C^* = 0$. However, we believe that a more symmetric model would learn better quality embeddings.

In (12) we denoted by Θ and Θ^* the learnable parameters of *GAE* and *GAE**, respectively. These are essentially the parameters of their GCN encoders, as was defined in (2)-(3). During optimization, we iteratively update these weights with back-propagation gradient descent, synchronously updating them: first, the gradients are computed with respect to fixed weights from the previous update step, and then parameters get updated (for more information on optimization schemes, see Section III-D). The model is shown in Figure 1.

D. OPTIMIZATION

In our implementation, we experiment with different training regimes for joint optimization of two networks. The standard training uses synchronous updates, computing gradients for both *GAE* and *GAE** and renewing both parameter sets with dual parameters held fixed at the previous value. As an alternative to simultaneously updating model weights, we propose to alternate updates for G and G^*, between propagating gradients from the loss associated with model G parameters on even step and with model G^* parameters on odd steps, keeping G's embeddings fixed. The idea is similar to the training of generative adversarial network (GAN), in which training a generator is alternated with training a discriminator. We expect this regime to have a similar effect of stabilizing training.

Following the author's implementation of GAE [59], we take Adam optimizer with a learning rate 0.01. In general, larger graphs (over 10000 nodes) require more training epochs; however, we observe that the order of hundreds or even 100 to 200 epochs worked well on our test cases.

Due to differences in scale, some components of the composite loss may overbalance others and disrupt optimization. To cope with that, we initially perform scaling of the additional part of the loss $\lambda^* L^*_{reg}(\Theta^*) + C \cdot L_{G^* \to G}(\Theta, \Theta^*)$ to make it equal to the general part $L_G(\Theta) + L_{G^*}(\Theta^*)$ times λ and then keep this scaling, effectively normalizing gradients on every step based on the first step. In our tests, we observed that $\lambda = 0.1, \lambda^* = 0.2$ work the best, as was found via hyper-parameter search. We also use standard deep learning regularization tactics, adding a small dropout 0.1 after the ReLU activation following the first convolutional layer. This means that we randomly knock down a specified small share of layer's neurons on each forward pass during training and turn our network into an ensemble, thus preventing excessive co-adaptation of different neurons.

E. JOINT DUAL GRAPH CONVOLUTIONAL NETWORK FOR SEMI-SUPERVISED CLASSIFICATION

Additionally, JONNEE admits modifications for labeled data incorporation in semi-supervised node K-class classification with labeled nodes $V_L \subseteq V$ (dual construction allows to work with semi-supervised edge classification, usually used as entity recognition for knowledge graphs). This model JONNLEE (with L for 'Labeled') is even simpler: instead of reconstructing the adjacency matrix with an inner product decoder, we may use one more GCN layer above representations F with input size d and output size K to obtain correct probabilities, which we train to approximate target class probabilities by softmax normalization, so that class probabilities $\approx \text{Softmax}(\text{GCNLayer}(F)) \in \mathbb{R}^{|V_L| \times K}$.

If we have edge labels in addition to node labels, we can use them in the same way. Otherwise, for the dual model to be semi-supervised as well, we label an edge as 1 if it connects two nodes with the same label, and 0 otherwise or if one of the nodes is not in the training dataset and its label is unknown. In such construction we force an edge to indicate that its adjacent vertices belong to the same class, also using softmax class probabilities $\approx \sigma(\text{GCNLayer}(F^*)) \in \mathbb{R}^{|V^*_L| \times 2}$. This way, we can use our model with almost the same loss function, in which L_G and L_{G^*} terms are now multinomial cross-entropies between the one-hot label distribution and the last layer output in the case of single-label multi-class classification.

For the training set, we take the nodes that have labels. This way, we learn node and edge representations consistent with the training labels and also have them preserve first-order proximities with Laplacian regularization.

IV. EXPERIMENTS WITH THE MODEL

In this section, we report on the experiments conducted to validate the proposed model. We start with describing experiment setup, model implementation, and machine learning settings. Next, we provide description of extensive experiments on the model components impact and hyper-parameter optimization. Finally, we showed the value of two training modes on the model performance.

A. EXPERIMENTAL SETUP

1) MODEL IMPLEMENTATION AND HARDWARE

The model is written in Python3 with PyTorch framework, version `python`0.3.1.post2. The code to reproduce the results will become available after the paper is accepted. All computations have been conducted on a single laptop computer MacBook Pro 2013, with 2.3 GHz Intel Core i7 processor with 4 cores and 16 GB RAM, with only one core used in order to not exploit differences in models' ability to parallelize.

2) EVALUATION FOR MACHINE LEARNING TASKS ON GRAPHS

Since we maintain that edges can include additional information that cannot be presented by incident nodes embeddings alone, we mainly focus on the link prediction problem to guide our model evaluation. The link prediction problem is one of the core machine learning tasks on graphs, which can be solved by our model better than by other state-of-the-art embedding-based solutions. Secondly, we test our model on a multi-class node classification problem.

In these experiments, we run JONNEE for 100 epochs on 5 random 2000-node subgraphs with 5 random train-validation-test splits of all datasets except for HSE and Cora, on which only different splits were used. All seeds are kept the same for different algorithms and hyperparameter choices within one test suite, while sigmoid activation of output reconstruction layer was thresholded using validation set and set to 0.6.

3) DATASETS

In Table 1, we provide a summary of datasets we use for experiments.

TABLE 1. Summary of datasets.

	$\mid V\mid$	$\mid E\mid$	# classes
Blog-Catalog [76]	10312	333983	2 classes, 39 labels/node
Ca-GrQc [76]	5242	14496	-
Cit-HepTh [76]	27770	352807	-
HSE [77]	1491	2947 weighted	-
Cora [45]	2708, features	5429	7 classes

B. COMPONENTS TESTING: FEATURE GENERATION

One of the central components of the model is seeding the autoencoder with informative node features by learning them with a sequence-based algorithm from the training data. In this set of experiments, we verify that this procedure is indeed useful and observe some other details regarding the choice of sequence-based algorithm for feature generation.

1) GENERATED VS. DUMMY FEATURES

By *dummy* features, we refer to the identity matrix I_n used in place of node features and which can be viewed as a set of uninformative features, with each feature only expressing the identity of its node. Other approaches are based on random walks from node2vec (n2v in Tables) and diff2vec (d2v in Tables) models as described previously in Section III. The results are shown in Table 2.

TABLE 2. Feature generation evaluation.

Dataset \ model	Dummy	N2v	D2v	Metric
Cit-HepTh	0.569 ± 0.038	0.612 ± 0.060	$\mathbf{0.722 \pm 0.036}$	Acc
	0.593 ± 0.063	0.751 ± 0.074	$\mathbf{0.771 \pm 0.050}$	ROC
	0.573 ± 0.063	$\mathbf{0.813 \pm 0.055}$	0.791 ± 0.052	AP
	0.666 ± 0.031	0.663 ± 0.059	$\mathbf{0.702 \pm 0.054}$	F1
HSE	0.523 ± 0.029	$\mathbf{0.686 \pm 0.027}$	0.543 ± 0.033	Acc
	0.513 ± 0.031	$\mathbf{0.859 \pm 0.031}$	0.807 ± 0.020	ROC
	0.589 ± 0.015	$\mathbf{0.889 \pm 0.026}$	0.866 ± 0.014	AP
	0.548 ± 0.030	$\mathbf{0.739 \pm 0.026}$	0.642 ± 0.015	F1

These results show that the proposed procedure significantly improves performance over dummy features. The choice between random walks and diffusion, in general, depends on the graph's properties: sparser graphs could be better handled by diffusion, whereas random walks generally yield better and more stable results on dense graphs.

2) CASE OF WEIGHTED GRAPH

Next, we analyze the performance of feature generation on a weighted network. We observe that while a biased random walk performed by node2vec can use edge weights by placing additional weight onto 'heavier' edges where appropriate, the original diff2vec does not use weights. We were able to slightly improve diff2vec performance on weighted graphs by modifying its diffusion graph construction stage to account for weights: we make propagation probabilities proportional to respective edge weights, as shown in Table 3. The HSE dataset itself was collected in [13] for HSE University co-authorship recommender system, while weights represent the aggregated quality of overall papers published by researchers.

TABLE 3. Comparison of regular and weighted diffusions.

Dataset \ model	D2v	Weighted D2v	Metric
HSE	0.552 ± 0.033	$\mathbf{0.556 \pm 0.037}$	Acc
	0.806 ± 0.013	$\mathbf{0.811 \pm 0.030}$	ROC
	0.863 ± 0.017	$\mathbf{0.866 \pm 0.014}$	AP
	0.642 ± 0.013	$\mathbf{0.648 \pm 0.024}$	F1

3) CASE OF NONTRIVIAL ATTRIBUTIVE NODE FEATURES X_0

In cases when additional information for each node is available, we face a potential trade-off: our learned sequence features may dilute the information provided by these extra features, which are valuable as they contain information additional to the structure data. Thus, we compare different options for making features on the first stage: we either concatenate two kinds of features or do not use sequence-based features at all (for the reasons mentioned above, we assume these features are valuable). Results on the Cora dataset are reproduced in Table 4. We find that using learned features X_1 in addition to extrinsic ones X_0 allows us to improve the results over all metrics by a significant margin.

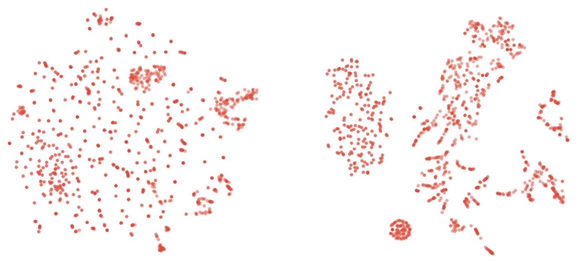
TABLE 4. Importance of domain-specific features X_0 on Cora dataset.

Dataset \ model	X	$[X_0, \mathrm{N2v}]$	$[X_0, \mathrm{D2v}]$	Metric
Cora	0.540	0.696	$\mathbf{0.697}$	Acc
	0.642	$\mathbf{0.871}$	0.860	ROC
	0.661	$\mathbf{0.889}$	0.879	AP
	0.659	$\mathbf{0.750}$	$\mathbf{0.750}$	F1

4) Node2vec VS diff2vec

Although node2vec proves to be more stable and reliable, our experiments reveal that diffusion-based features result in better clustering in the embedding space. Below, we visualize $d = 16$ dimensional embeddings in 2 dimensions using

TSNE algorithm [78], which is a common dimensionality reduction tool for vector-valued data. We also compute *silhouette score* with K-Means detected clusters for the original JONNEE embeddings on 1000 nodes subgraph of HSE dataset (see Figure 2). The observation that clustering is induced by diffusion is indeed not surprising: when a truncated random walk with generic parameters $p, q \approx 1$ most often leaves the starting node in a few hops, diffusion stays local due to the same probability of nodes being sampled in diffusion graph.



(a) random walk features, silhCoef 0.256 (b) diffusion features, silhCoef 0.664

FIGURE 2. **Diffusion features aid clustering in the embedding space of Weighted HSE dataset. Higher silhouette scores correspond to better clusters.**

C. COMPONENTS TESTING: β-MASKING

In these experiments, we aim to find out whether masking aids learning. For this purpose, both positive and negative values were tested, with positive employed to smooth learning and negative to force closer fitting on training data. The results for selected representative mask intensities for Cit-HepTh and Ca-GrQc are shown in Table 5. It shows that a small positive β-value improves generalization by regularizing the model, but negative masking is unstable and requires further investigation.

We also try taking larger masks for the dual optimizer since, in the node-based tasks, it is only required to learn representations consistent with the primal graph rather than perform well on edge-based tasks. The Table 6 shows some results for varying β^* with the same optimal βs on every dataset. We find that by taking a $\beta^* > \beta$, it is possible to improve quality. Overall, however, parameter tuning for masking coefficients offers only marginal improvements when the appropriate order of magnitude is found, so defaulting it to $\beta = 0.01, \beta^* = 0.02$ is a good choice.

TABLE 5. **Testing β values.**

Dataset \ model	No masking	$\beta = -0.01$	$\beta = 0.01$	Metric
Cit-HepTh	0.594 ± 0.044	$\mathbf{0.605 \pm 0.084}$	0.600 ± 0.064	Acc
	0.741 ± 0.060	0.740 ± 0.068	$\mathbf{0.757 \pm 0.072}$	ROC
	0.812 ± 0.050	0.811 ± 0.054	$\mathbf{0.821 \pm 0.053}$	AP
	0.651 ± 0.040	0.649 ± 0.048	$\mathbf{0.654 \pm 0.056}$	F1
Ca-GrQc	$\mathbf{0.660 \pm 0.043}$	0.653 ± 0.036	0.656 ± 0.020	Acc
	0.802 ± 0.043	0.801 ± 0.034	$\mathbf{0.804 \pm 0.033}$	ROC
	0.845 ± 0.032	0.839 ± 0.025	$\mathbf{0.851 \pm 0.025}$	AP
	0.703 ± 0.038	0.705 ± 0.033	$\mathbf{0.714 \pm 0.027}$	F1

TABLE 6. **Changing β^* with fixed $\beta = 0.01$.**

Dataset \ model	$\beta^* = 0.1\beta$	$\beta^* = 2\beta$	$\beta^* = 10\beta$	Metric
Cit-HepTh	0.619 ± 0.048	$\mathbf{0.620 \pm 0.056}$	0.600 ± 0.066	Acc
	0.733 ± 0.055	$\mathbf{0.750 \pm 0.067}$	0.730 ± 0.081	ROC
	0.818 ± 0.046	$\mathbf{0.820 \pm 0.049}$	0.809 ± 0.057	AP
	0.660 ± 0.043	$\mathbf{0.666 \pm 0.049}$	0.655 ± 0.062	F1
Ca-GrQc	0.662 ± 0.008	$\mathbf{0.669 \pm 0.008}$	0.665 ± 0.015	Acc
	0.804 ± 0.020	$\mathbf{0.805 \pm 0.018}$	0.803 ± 0.022	ROC
	0.851 ± 0.021	0.854 ± 0.019	$\mathbf{0.855 \pm 0.016}$	AP
	0.706 ± 0.011	0.708 ± 0.006	$\mathbf{0.711 \pm 0.013}$	F1

TABLE 7. **Laplacian regularization for tied λs.**

Dataset \ model	No Laplacian	$\lambda = 0.1$	$\lambda = 1.$	Metric
Cit-HepTh	0.630 ± 0.017	$\mathbf{0.642 \pm 0.020}$	0.600 ± 0.043	Acc
	0.784 ± 0.018	$\mathbf{0.794 \pm 0.014}$	0.786 ± 0.023	ROC
	0.780 ± 0.012	0.788 ± 0.025	$\mathbf{0.832 \pm 0.023}$	AP
	0.683 ± 0.016	$\mathbf{0.691 \pm 0.013}$	0.684 ± 0.019	F1
Ca-GrQc	0.594 ± 0.054	0.592 ± 0.058	$\mathbf{0.597 \pm 0.043}$	Acc
	0.736 ± 0.062	$\mathbf{0.768 \pm 0.071}$	0.757 ± 0.053	ROC
	0.811 ± 0.044	$\mathbf{0.809 \pm 0.055}$	0.808 ± 0.042	AP
	0.649 ± 0.046	$\mathbf{0.681 \pm 0.048}$	0.673 ± 0.036	F1

TABLE 8. **Choosing λ^*.**

Dataset \ model	$\lambda^* = 0.1\lambda$	$\lambda^* = 2\lambda$	$\lambda^* = 10\lambda$	
Cit-HepTh, $\lambda = 0.1$	0.612 ± 0.052	$\mathbf{0.612 \pm 0.061}$	0.609 ± 0.055	Acc
	0.740 ± 0.070	$\mathbf{0.745 \pm 0.082}$	0.743 ± 0.074	ROC
	0.812 ± 0.050	$\mathbf{0.814 \pm 0.059}$	0.813 ± 0.057	AP
	0.658 ± 0.049	$\mathbf{0.659 \pm 0.060}$	0.655 ± 0.057	F1
Ca-GrQc	0.651 ± 0.012	0.657 ± 0.021	$\mathbf{0.665 \pm 0.028}$	Acc
	0.793 ± 0.018	0.802 ± 0.018	$\mathbf{0.804 \pm 0.018}$	ROC
	0.843 ± 0.011	$\mathbf{0.858 \pm 0.011}$	0.855 ± 0.011	AP
	0.699 ± 0.015	0.707 ± 0.015	$\mathbf{0.713 \pm 0.019}$	F1

TABLE 9. **Joint training comparison.**

Dataset \ model	No joint loss	$C = 0.01$	$C = 1.$	Metric
Cit-HepTh	0.590 ± 0.060	$\mathbf{0.612 \pm 0.048}$	0.557 ± 0.038	Acc
	0.740 ± 0.063	0.752 ± 0.060	$\mathbf{0.759 \pm 0.044}$	ROC
	0.810 ± 0.052	$\mathbf{0.818 \pm 0.048}$	0.786 ± 0.041	AP
	0.650 ± 0.050	0.661 ± 0.046	$\mathbf{0.672 \pm 0.021}$	F1
Ca-GrQc	0.640 ± 0.025	$\mathbf{0.661 \pm 0.018}$	0.645 ± 0.019	Acc
	0.793 ± 0.024	$\mathbf{0.803 \pm 0.020}$	0.787 ± 0.024	ROC
	0.846 ± 0.015	$\mathbf{0.854 \pm 0.013}$	0.843 ± 0.013	AP
	0.697 ± 0.023	$\mathbf{0.709 \pm 0.016}$	0.700 ± 0.019	F1

D. COMPONENTS TESTING: LAPLACIAN REGULARIZATION

Similar to the previous experiment, we test Laplacian regularization for $\lambda^* = \lambda$ choosing the best parameter (see Table 7) and then try varying the ratio between the two coefficients (see Table 8). Experiments show that adding Laplacian regularization slightly improved the quality of our model. The improvement that λ^*-tuning offers is fairly small, so it is enough to set $\lambda = 0.1$ and $\lambda^* = 0.2$.

E. COMPONENTS TESTING: LEARNING WITH DUAL GRAPH CONSTRAINT

In this set of experiments, we test one of the central features of our model — learning joint embedding for nodes and edges as nodes of the dual graph. From the results reported in Table 9, we see that dual learning with a small parameter coefficient provides an improvement over a node-based link prediction.

Remark 3: It is important to say a few words on hyperparameter choice and scaling. We observe that imposing a joint loss benefits the quality of resulting node embeddings.

However, training with it should be managed more carefully by tuning the step size so that one component of the loss function does not outweigh the others. In particular, the common loss should not overbalance the respective losses of the two autoencoders. Otherwise, the model would still learn to approximate averages, but these would be of low quality, not fit to reconstruct edges, and useless altogether. For that, we compute the ratio of loss components L_G and the remaining part and then scale the remainder down by that constant factor on each subsequent iteration. This way, we achieve the initial composition of losses on the order of $1 : \lambda$ for the primal model's weights and on the order of $1 : \lambda^*$ for the dual model's weights, with optimization weighing gradient directions in the same way. During the process of optimization, these ratios may change, but the initial calibration is essential.

F. ALTERNATING OPTIMIZATION

We conduct two tests of the alternating optimization (see Table 10) and observe that, although the mean results for classification scores are generally worse, sample variance across different runs is consistently reduced for alternating optimization. This leads us to the idea that by selecting an appropriate number of steps performed by each optimizer before passing over to dual and learning rates, one could perform better with this procedure.

TABLE 10. Optimization modes comparison.

Dataset \ mode	Simultaneous	Alternating	Metric
Cit-HepTh	$0.565 \pm \mathbf{0.09}$	0.562 ± 0.092	Acc
	0.661 ± 0.124	$0.641 \pm \mathbf{0.087}$	ROC
	0.734 ± 0.103	$0.731 \pm \mathbf{0.089}$	AP
	0.612 ± 0.095	$0.591 \pm \mathbf{0.088}$	F1
Ca-GrQc	0.586 ± 0.074	$0.582 \pm \mathbf{0.067}$	Acc
	0.681 ± 0.091	$0.685 \pm \mathbf{0.078}$	ROC
	$0.750 \pm \mathbf{0.042}$	0.762 ± 0.072	AP
	0.641 ± 0.051	$0.627 \pm \mathbf{0.042}$	F1

V. EVALUATION FOR GRAPH MACHINE LEARNING

We perform a comparison with state-of-the-art graph embedding models on link prediction (unsupervised setting), node classification (unsupervised and semi-supervised settings), and network visualization (unsupervised setting) problems. Each Table shows the results of our best model after testing of components. The compared models are chosen with default hyperparameters, as stated in corresponding research papers. A unique pipeline for training and evaluation of all the models was applied in each experiment compared here.

A. LINK PREDICTION OVERALL

We compare our model to a selection of other popular structural embedding models based on different principles: HOPE [16] and GraphFactorization (GrF in Tables) [79] based on matrix factorizations, node2vec (N2v in Tables) and unmodified diff2vec (D2v in Tables) learning representations from sampled sequences, and to GAE [59], GAT [61], GraphSAGE (GrS in Tables) [60], a deep learning models for graphs. With this, we want to verify that our JONNEE can be compared favorably along all dimensions, against a diverse set of models, according to various metrics and datasets. Results of our experiments are shown in Table 11. In the experiments, we take the embedding dimension to be $d = 16$ and sample several subgraphs while holding a train/validation/test split constant with proportion 0.85/0.05/0.1. Other hyperparameters were default for benchmark models. We use implementation of GAT, and GraphSAGE from DGL framework [80]. The default number of layers here for both models is 2. GraphSAGE uses GCN aggregation. The results show that tuned JONNEE with diffusion features and 400 epochs mostly outperforms baselines.

Despite the larger number of parameters (due to accounting for Line Graph), Figure 3 shows that our model converges faster than GAT and GraphSAGE on HSE, Cit-HepTh, Ca-GrQc datasets.

B. MULTI-CLASS CLASSIFICATION FOR NODES

In this set of experiments, we verify that the model's performance generalizes well from link prediction to other tasks such as multi-class node classification. Additionally, we offer a simplified version of our Dual Autoencoder JONNEE, named JONNLEE, directly adapted to incorporate available node labels during training in a semi-supervised manner.

Evaluating our model on multi-class classification, we no longer specify sample standard deviations to keep the tables clean. We train models on the 80% subsample and validate on the rest 20%.

1) UNSUPERVISED REPRESENTATION EXTRACTION

In these experiments, we train models on the entire graph G in an unsupervised way to generate node embeddings and then fit a Random forest with default parameters for multi-class classification on a random subsample of node representations to evaluate the quality of embeddings. By producing embeddings, we reduce the node classification problem to a standard classification problem with vector input. We choose Random forest as one baseline method proved to show better performance compared to other standard methods like support vector machine (SVM) or gradient boosting (XGBoost) on multi-class classification problem on embedding vectors for co-authorship networks [14], while leaving comparison on other methods for JONNEE representation for the future work. All models, except for Graph Factorizations, perform better than random (which in the case of Cora is ≈ 0.14 accuracy). The results are reported in Table 12.

2) JONNLEE: SEMI-SUPERVISED ANALOGUE OF JONNEE

By exploiting knowledge of labels during the training stage, we incorporate task knowledge into the embeddings and expect better performance on the relevant task. For a fair comparison, we take another semi-supervised model GCN and compare it to JONNLEE, a semi-supervised modification of JONNEE. From this experiment, we verify that the

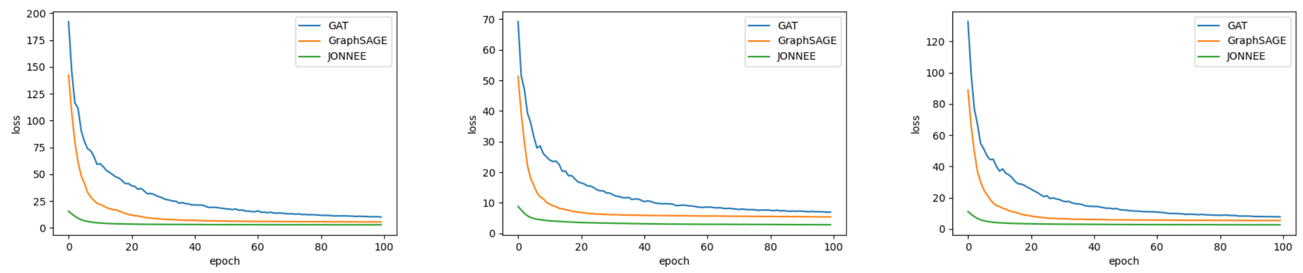


FIGURE 3. Convergence of loss function for GAT, GraphSAGE and JONNEE models.

TABLE 11. Link prediction evaluation.

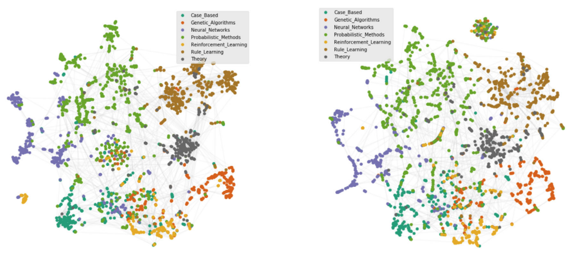
	GrF	HOPE	N2v	D2v	GAE	GAT	GrS	Ours	Metric
Cit-HepTh	0.555 ± 0.035	0.741 ± 0.012	0.616 ± 0.024	0.720 ± 0.023	0.633 ± 0.010	0.725 ± 0.012	0.722 ± 0.010	$\mathbf{0.751 \pm 0.010}$	Acc
	0.564 ± 0.044	0.730 ± 0.014	0.758 ± 0.023	0.799 ± 0.012	0.778 ± 0.018	0.9095 ± 0.017	$\mathbf{0.936 \pm 0.015}$	0.817 ± 0.007	ROC
	0.554 ± 0.024	0.759 ± 0.012	0.757 ± 0.019	0.817 ± 0.007	0.845 ± 0.017	0.8995 ± 0.015	$\mathbf{0.927 \pm 0.015}$	0.841 ± 0.004	AP
	0.583 ± 0.032	0.682 ± 0.018	0.698 ± 0.016	0.730 ± 0.018	0.677 ± 0.011	$\mathbf{0.7830 \pm 0.012}$	0.782 ± 0.011	0.748 ± 0.010	F1
Ca-GrQc	0.596 ± 0.017	0.689 ± 0.008	0.557 ± 0.025	0.615 ± 0.027	0.602 ± 0.016	$\mathbf{0.721 \pm 0.015}$	0.690 ± 0.013	0.642 ± 0.027	Acc
	0.661 ± 0.025	0.690 ± 0.009	0.732 ± 0.010	0.735 ± 0.011	0.670 ± 0.016	$\mathbf{0.927 \pm 0.014}$	0.916 ± 0.013	0.797 ± 0.011	ROC
	0.682 ± 0.022	0.693 ± 0.008	0.800 ± 0.009	0.795 ± 0.003	0.718 ± 0.023	$\mathbf{0.930 \pm 0.019}$	0.909 ± 0.017	0.852 ± 0.004	AP
	0.631 ± 0.016	0.557 ± 0.018	0.657 ± 0.010	0.644 ± 0.013	0.630 ± 0.015	$\mathbf{0.782 \pm 0.014}$	0.763 ± 0.012	0.700 ± 0.020	F1
HSE	0.665 ± 0.000	0.777 ± 0.000	0.477 ± 0.005	0.463 ± 0.000	0.566 ± 0.000	0.684 ± 0.000	0.699 ± 0.000	$\mathbf{0.793 \pm 0.006}$	Acc
	0.745 ± 0.000	0.781 ± 0.000	0.674 ± 0.009	0.751 ± 0.000	0.678 ± 0.000	0.907 ± 0.000	$\mathbf{0.925 \pm 0.000}$	0.876 ± 0.003	ROC
	0.766 ± 0.000	0.783 ± 0.000	0.743 ± 0.009	0.842 ± 0.000	0.706 ± 0.000	0.910 ± 0.000	$\mathbf{0.920 \pm 0.000}$	0.906 ± 0.002	AP
	0.715 ± 0.000	0.717 ± 0.000	0.631 ± 0.005	0.601 ± 0.000	0.558 ± 0.000	0.760 ± 0.000	$\mathbf{0.769 \pm 0.000}$	0.748 ± 0.005	F1
Blog -Catalog	0.598 ± 0.027	$\mathbf{0.734 \pm 0.012}$	0.601 ± 0.008	0.698 ± 0.009	0.662 ± 0.011	0.647 ± 0.007	0.649 ± 0.008	0.699 ± 0.010	Acc
	0.539 ± 0.035	0.798 ± 0.001	0.703 ± 0.021	0.733 ± 0.010	0.830 ± 0.005	0.778 ± 0.009	0.767 ± 0.008	$\mathbf{0.890 \pm 0.004}$	ROC
	0.477 ± 0.020	0.839 ± 0.006	0.631 ± 0.017	0.672 ± 0.017	0.880 ± 0.005	0.681 ± 0.006	0.681 ± 0.005	$\mathbf{0.904 \pm 0.003}$	AP
	0.660 ± 0.036	0.750 ± 0.006	0.707 ± 0.005	0.752 ± 0.005	0.719 ± 0.007	0.737 ± 0.006	0.739 ± 0.005	$\mathbf{0.753 \pm 0.005}$	F1

TABLE 12. Node classification evaluation.

	GrF	HOPE	N2v	D2v	GAE	GAT	GrS	Ours	Metric
Cora	0.087	0.577	0.714	0.681	0.625	0.449	0.503	**0.760**	Acc
	0.154	0.689	0.804	0.779	0.734	0.371	0.457	**0.831**	F1
Blog -Catalog	0.060	0.060	0.065	0.065	0.063	0.074	0.054	**0.085**	Acc
	0.021	0.010	0.020	0.020	0.020	0.016	0.019	0.017	F1

TABLE 13. JONNLEE in semi-supervised mode.

Dataset \ model	GCN	GAT	GrS	Ours	Metric
Cora	0.840	0.862	0.854	**0.870**	Acc
	0.820	0.838	0.833	**0.843**	F1
Blog-Catalog	0.140	0.158	0.155	**0.160**	Acc
	0.037	0.081	0.079	**0.089**	F1



(a) JONNEE with d=100 (b) diff2vec with d=100

FIGURE 4. Comparison of visualization for diffusion and JONNEE.

proposed modification of JONNEE for the semi-supervised setting outperforms the model without supervision and also compares favorably to the original GCN (see Table 13).

C. VISUALIZATION AND COMMUNITY DETECTION

Finally, we briefly demonstrate that the embeddings learned by JONNEE are suitable for visualization. For this, we compare it to diff2vec [37], known for accurate and highly modular cluster visualization. We use the Cora dataset, which is convenient due to having only a single label per node, which can be interpreted as a community/cluster label. In this case, communities are topic groups of papers on the ''machine learning'' topic, with features containing their TF-IDF encoded content and links representing citations.

We learn 100-dimensional embeddings to retain more information by both models and use TSNE [78] to obtain planar embeddings. We then plot the embeddings and edges between them and color the points according to their class labels. The results for both models are displayed in Figure 4, where we see that embeddings learned by diff2vec in an unsupervised way are nicely aligned with true labels, while JONNEE not only identifies thematic communities by label but also forms a cleaner and more modular cluster.

D. DISCUSSION

We see that the quality of the embeddings learned by JONNEE is almost always superior to state-of-the-art structural models. Moreover, it is exceptionally flexible and is able to beneficially incorporate both node and edge features and weights while being stable to hyperparameter perturbations. However, our model has a certain drawback of longer training, similar to matrix factorizations but less parallelizable.

Our model has complexity $O(|E|^2)$ in the worst case, which is arguably far from efficient. However, the paper is intended as a proof of concept for (edge-to-vertex) 'dual' graphs to be used and explored in tandem with regular graphs to learn more accurate representations. Moreover, for sparse real-world networks, this complexity is lower than the complexity of some factorizations, as mentioned above. Additional memory consumption is at worst $|E|^2$ at runtime, but storing all of the extra data is unnecessary. These problems we believe to be solvable with clever approximations to the dual graph similar to [11], [12], for which we have not found open-source code to conduct comparison. Code of our model and experiments could be found on GitHub.[1]

VI. CONCLUSION

In this work, we have developed a new embedding model JONNEE which learns high-quality node representations in tandem with edge representations. We have implemented the model and conducted an extensive range of experiments demonstrating that the model performs well compared to other state-of-the-art benchmarks.

The quality of the embeddings learned by JONNEE is almost always superior in the link prediction problem to those learned by state-of-the-art models of all different types: matrix factorization based, sequence-based. However, it shows competitive results to other deep learning methods. The model performs well in both unsupervised and semi-supervised settings for the node classification task. In addition, embeddings learned by JONNEE have a well-clustered structure and are suitable for visualization. JONNEE is exceptionally flexible and is able to beneficially incorporate both node and edge features and weights while being stable to hyperparameter perturbations.

However, our model has a certain drawback of longer training similar to matrix factorizations but less parallelizable. It can be overcome via changing core architecture components: graph autoencoders may be changed to graph sampling models or inductive models so that JONNEE will be able to process large and temporal networks. Also, it is interesting to see how the construction of joint constraints may be generalized for arbitrary embedding operators that can be approximated by a deep neural network of incident nodes embeddings. We believe that presented via Line graph self-supervision on learning network representation is a promising approach generalizing for complex networks.

ACKNOWLEDGMENT

The authors are grateful to Daniil Tikhomirov for his invaluable contribution to polishing the article presentation and text.

AUTHOR CONTRIBUTION

Ilya Makarov: Experiment design, evaluation, paper preparation, and research supervision. Ksenia Korovina: Model design and implementation, paper preparation, and coding experiments. Dmitrii Kiselev: Design of revision experiments, additional evaluation, and paper revision update.

[1]https://github.com/mkiseljov/JONNEE

REFERENCES

[1] L. G. Moyano, "Learning network representations," *Eur. Phys. J. Special Topics*, vol. 226, no. 3, pp. 499–518, Feb. 2017. [Online]. Available: https://link.springer.com/10.1140/epjst/e2016-60266-2

[2] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," 2017, *arXiv:1709.05584*. [Online]. Available: https://arxiv.org/abs/1709.05584

[3] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 9, pp. 1616–1637, Sep. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8294302/

[4] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 5, pp. 833–852, May 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8392745/

[5] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowl.-Based Syst.*, vol. 151, pp. 78–94, Jul. 2018.

[6] H. Chen, B. Perozzi, R. Al-Rfou, and S. Skiena, "A tutorial on network embeddings," 2018, *arXiv:1808.02590*. [Online]. Available: https://arxiv.org/abs/1808.02590

[7] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[8] I. Makarov, D. Kiselev, N. Nikitinsky, and L. Subelj, "Survey on graph embeddings and their applications to machine learning problems on graphs," *PeerJ Comput. Sci.*, vol. 7, p. e357, Feb. 2021.

[9] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 855–864, New York, NY, USA, doi: 10.1145/2939672.2939754.

[10] S. Abu-El-Haija, B. Perozzi, and R. Al-Rfou, "Learning edge representations via low-rank asymmetric projections," in *Proc. ACM Conf. Inf. Knowl. Manage.*, Nov. 2017, pp. 1787–1796. New York, NY, USA, doi: 10.1145/3132847.3132959.

[11] P. Goyal, H. Hosseinmardi, E. Ferrara, and A. Galstyan, "Capturing edge attributes via network embedding," *IEEE Trans. Comput. Social Syst.*, vol. 5, no. 4, pp. 907–917, Dec. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8536422/

[12] F. Monti, O. Shchur, A. Bojchevski, O. Litany, S. Günnemann, and M. M. Bronstein, "Dual-primal graph convolutional networks," 2018, *arXiv:1806.00770*. [Online]. Available: https://arxiv.org/abs/1806.00770

[13] I. Makarov, O. Gerasimova, P. Sulimov, and L. E. Zhukov, "Recommending co-authorship via network embeddings and feature engineering: The case of national research university higher school of economics," in *Proc. 18th ACM/IEEE Joint Conf. Digit. Libraries*, May 2018, pp. 365–366, New York, NY, USA, doi: 10.1145/3197026.3203911.

[14] I. Makarov, O. Gerasimova, P. Sulimov, and L. E. Zhukov, "Dual network embedding for representing research interests in the link prediction problem on co-authorship networks," *PeerJ Comput. Sci.*, vol. 5, p. e172, Jan. 2019. [Online]. Available: https://peerj.com/articles/cs-172

[15] I. Makarov, O. Gerasimova, P. Sulimov, K. Korovina, and L. E. Zhukov, "Joint node-edge network embedding for link prediction," in *Analysis of Images, Social Networks and Texts* (Lecture Notes in Computer Science), vol. 11179. Moscow, Russia: Springer, Jul. 2018, pp. 20–31, doi: 10.1007/978-3-030-11027-7_3.

[16] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2016, pp. 1105–1114, doi: 10.1145/2939672.2939751.

[17] J. Kruskal and M. Wish, *Multidimensional Scaling*, no. 11. Newbury Park, CA, USA: Sage, 1978.

[18] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *J. Amer. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, 1990.

[19] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," in *Linear Algebra*. Springer, 1971, pp. 134–151, ch. 10, doi: 10.1007/978-3-662-39778-7_10.

[20] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, Dec. 2000.

[21] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, Dec. 2000.

[22] X. He and P. Niyogi, "Locality preserving projections," in *Advances in Neural Information Processing Systems*, vol. 16, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA, USA: MIT Press, 2004. [Online]. Available: https://proceedings.neurips.cc/paper/2003/file/d69116f8 b0140cdeb1f99a4d5096ffe4-Paper.pdf

[23] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2002, pp. 585–591. [Online]. Available: https://www-2.cs.cmu.edu/Groups/NIPS/NIPS2001/papers/psgz/AA42.ps.gz

[24] B. Shaw and T. Jebara, "Structure preserving embedding," in *Proc. 26th Annu. Int. Conf. Mach. Learn. (ICML)*, New York, NY, USA, 2009, pp. 937–944.

[25] D. Luo, F. Nie, H. Huang, and C. H. Ding, "Cauchy graph embedding," in *Proc. 28th Int. Conf. Mach. Learn. (ICML)*, 2011, pp. 553–560. [Online]. Available: https://icml.cc/2011/papers/353_icmlpaper.pdf

[26] S. Cao, W. Lu, and Q. Xu, "GraRep: Learning graph representations with global structural information," in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage.*, New York, NY, USA, Oct. 2015, pp. 891–900.

[27] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, vol. 26, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2013. [Online]. Available: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec0 39965f3c4923ce901b-Paper.pdf

[28] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2014, pp. 701–710, New York, NY, USA, doi: 10.1145/2623330.2623732.

[29] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena, "Don't walk, skip! Online learning of multi-scale network embeddings," 2016, *arXiv:1605.02115*. [Online]. Available: https://arxiv.org/abs/1605.02115

[30] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 1067–1077.

[31] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, "Harp: Hierarchical representation learning for networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 2127–2134. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11849

[32] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, New York, NY, USA, 2009, pp. 817–826. [Online]. Available: https://portal.acm.org/citation.cfm?doid=1557019.1557109

[33] R. Feng, Y. Yang, W. Hu, F. Wu, and Y. Zhang, "Representation learning for scale-free networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 282–289. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11256

[34] B. Rozemberczki, R. Davies, R. Sarkar, and C. Sutton, "GEMSEC: Graph embedding with self clustering," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining*, New York, NY, USA, Aug. 2019, pp. 65–72.

[35] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. A. Alemi, "Watch your step: Learning node embeddings via graph attention," in *Advances in Neural Information Processing Systems*, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/8a94ecfa54dcb88a2f a993bfa6388f9e-Paper.pdf

[36] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, L. Song, and Y. Qi, "Geniepath: Graph neural networks with adaptive receptive paths," in *Proc. 33rd AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4424–4431.

[37] B. Rozemberczki and R. Sarkar, "Fast sequence-based embedding with diffusion graphs," in *Proc. Int. Workshop Complex Netw.* Cham, Switzerland: Springer, 2018, pp. 99–107.

[38] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.

[39] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proc. IEEE*, vol. 106, no. 5, pp. 808–828, May 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8347162/

[40] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009. [Online]. Available: https://ieeexplore.ieee.org/document/4700287/

[41] K. Li, J. Gao, S. Guo, N. Du, X. Li, and A. Zhang, "LRBM: A restricted Boltzmann machine based approach for representation learning on linked data," in *Proc. IEEE Int. Conf. Data Mining*, Dec. 2014, pp. 300–309.

[42] X. Li, N. Du, H. Li, K. Li, J. Gao, and A. Zhang, "A deep learning approach to link prediction in dynamic networks," in *Proc. SIAM Int. Conf. Data Mining*, 2014, pp. 289–297.

[43] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2016, pp. 1225–1234, doi: 10.1145/2939672.2939753.

[44] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Proc. AAAI Conf. Artif. Intell.*, vol. 30, 2016, pp. 1145–1152. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10179

[45] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*. [Online]. Available: https://arxiv.org/abs/1609.02907

[46] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, J. Dy and A. Krause, Eds. Stockholm, Sweden: PMLR, Jul. 2018, pp. 942–950. [Online]. Available: https://proceedings.mlr.press/v80/chen18p.html

[47] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–15. [Online]. Available: https://openreview.net/forum?id=rytstxWAW

[48] M. Lei, Y. Shi, and L. Niu, "The applications of stochastic models in network embedding: A survey," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. (WI)*, Dec. 2018, pp. 635–638.

[49] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013, *arXiv:1312.6203*. [Online]. Available: https://arxiv.org/abs/1312.6203

[50] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, vol. 28, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/f9be311e65d81a9ad8 150a60844bb94c-Paper.pdf

[51] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," 2015, *arXiv:1506.05163*. [Online]. Available: https://arxiv.org/abs/1506.05163

[52] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, M. F. Balcan and K. Q. Weinberger, Eds. New York, NY, USA: PMLR, 20–22 Jun. 2016, pp. 2014–2023. [Online]. Available: https://proceedings.mlr.press/v48/niepert16.html

[53] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, vol. 29, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/04df4d434d481c5bb7 23be1b6df1ee65-Paper.pdf

[54] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "CayleyNets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, Jan. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8521593/

[55] 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. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 5115–5124. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2017/html/Monti_Geometric%_ Deep_Learning_CVPR_2017_paper.html

[56] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, "SplineCNN: Fast geometric deep learning with continuous B-spline kernels," in *Proc. IEEE CVPR*, Jun. 2018, pp. 869–877. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/html/Fey_SplineCNN_%Fast_Geometric_CVPR_2018_paper.html

[57] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 29, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/390e982518a50e280d8e2b535462ec1f-Paper.pdf

[58] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–16. [Online]. Available: https://openreview.net/forum?id=SJiHXGWAZ

[59] T. N. Kipf and M. Welling, "Variational graph auto-encoders," 2016, *arXiv:1611.07308*. [Online]. Available: https://arxiv.org/abs/1611.07308

[60] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf

[61] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–12. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[62] C. Tu, H. Liu, Z. Liu, and M. Sun, "CANE: Context-aware network embedding for relation modeling," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics*. Vancouver, BC, Canada, Jul. 2017, pp. 1722–1731.

[63] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *Proc. 10th ACM Int. Conf. Web Search Data Mining*, New York, NY, USA, Feb. 2017, pp. 731–739, doi: 10.1145/3018661.3018667.

[64] H. Gui, J. Liu, F. Tao, M. Jiang, B. Norick, and J. Han, "Large-scale embedding learning in heterogeneous event data," in *Proc. 16th Int. Conf. Data Mining (ICDM)*, Dec. 2016, pp. 907–912.

[65] C. Moon, P. Jones, and N. F. Samatova, "Learning entity type embeddings for knowledge graph completion," in *Proc. ACM Conf. Inf. Knowl. Manage.*, Nov. 2017, pp. 2181–2187. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/9491

[66] I. Makarov and O. Gerasimova, "Link prediction regression for weighted co-authorship networks," in *Proc. Int. Work-Conf. Artif. Neural Netw.* Cham, Switzerland: Springer, 2019, pp. 667–677.

[67] I. Makarov and O. Gerasimova, "Predicting collaborations in co-authorship network," in *Proc. 14th Int. Workshop Semantic Social Media Adaptation Personalization (SMAP)*, Jun. 2019, pp. 1–6.

[68] I. Makarov, O. Gerasimova, P. Sulimov, and L. E. Zhukov, "Co-authorship network embedding and recommending collaborators via network embedding," in *Proc. Int. Conf. Anal. Images, Social Netw. Texts* (Lecture Notes in Computer Science), vol. 11179. Cham, Switzerland: Springer, 2018, pp. 32–38, doi: 10.1007/978-3-030-11027-7_4.

[69] I. Makarov, O. Bulanov, O. Gerasimova, N. Meshcheryakova, I. Karpov, and L. E. Zhukov, "Scientific matchmaker: Collaborator recommender system," in *Analysis of Images, Social Networks and Texts* (Lecture Notes in Computer Science), vol. 10716. Cham, Switzerland: Springer, 2018, pp. 404–410, doi: 10.1007/978-3-319-73013-4_37.

[70] Y. Liu, S. Pan, M. Jin, C. Zhou, F. Xia, and P. S. Yu, "Graph self-supervised learning: A survey," 2021, *arXiv:2103.00111*. [Online]. Available: https://arxiv.org/abs/2103.00111

[71] Y. Xie, Z. Xu, Z. Wang, and S. Ji, "Self-supervised learning of graph neural networks: A unified review," 2021, *arXiv:2102.10757*. [Online]. Available: https://arxiv.org/abs/2102.10757

[72] P. Goyal, H. Hosseinmardi, E. Ferrara, and A. Galstyan, "Embedding networks with edge attributes," in *Proc. 29th Hypertext Social Media*, New York, NY, USA, Jul. 2018, pp. 38–42, doi: 10.1145/3209542.3209571.

[73] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proc. World Wide Web Conf. WWW)*, 2018, pp. 499–508.

[74] I. Makarov, M. Makarov, and D. Kiselev, "Fusion of text and graph information for machine learning problems on graphs," *PeerJ Comput. Sci.*, vol. 7, pp. 1–26, May 2021.

[75] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, M. F. Balcan and K. Q. Weinberger, Eds. New York, NY, USA: PMLR, Jun. 2016, pp. 40–48. [Online]. Available: https://proceedings.mlr.press/v48/yanga16.html

[76] J. Leskovec. (2004). *Snap Project*. [Online]. Available: https://snap.stanford.edu/data/

[77] HSE-DIT. (2017). *HSE Publications*. [Online]. Available: https://publications.hse.ru/en

[78] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008. [Online]. Available: https://www.jmlr.org/papers/volume9/vandermaaten08a/

[79] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proc. 22nd Int. Conf. World Wide Web*, New York, NY, USA, 2013, pp. 37–48. [Online]. Available: https://dl.acm.org/citation.cfm?doid=2488388.2488393

[80] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," 2019, *arXiv:1909.01315*. [Online]. Available: https://arxiv.org/abs/1909.01315

ILYA MAKAROV received the Specialist degree in mathematics from the Lomonosov Moscow State University, Moscow, Russia. He is currently pursuing the Ph.D. degree in computer science with the University of Ljubljana, Ljubljana, Slovenia.

Since 2011, he has been a full-time Lecturer with the School of Data Analysis and Artificial Intelligence, HSE University, where he was the School Deputy Head, from 2012 to 2016. He is also the Program Director of BigData Academy MADE from VK, and a Researcher at Samsung-PDMI Joint AI Center, St. Petersburg Department of V.A. Steklov Mathematical Institute, Russian Academy of Sciences, Saint Petersburg, Russia, and also with the Artificial Intelligence Research Institute, Moscow. He is also a Lecturer at the Moscow Institute of Physics and Technology and a Machine Learning Engineer and the Head of Data Science Tech Master Program in NLP at the National University of Science and Technology MISIS.

KSENIA KOROVINA received the B.S. degree in mathematics from HSE University, Moscow, Russia, in 2018, and the M.S. degree in machine learning from Carnegie Mellon University, Pittsburgh, PA, USA, in 2019.

From 2016 to 2019, she was a Research Assistant with HSE University and Carnegie Mellon University, and an Intern at Twitter. She is currently a Software Engineer at Google.

DMITRII KISELEV received the master's degree in applied mathematics and informatics from HSE University, Moscow, Russia, where he is currently pursuing the Ph.D. degree in computer science.

Since 2018, he has been a part-time Lecturer with the School of Data Analysis and Artificial Intelligence, HSE University. He is currently a Researcher in the field of application of graph neural networks to the Industrial AI at the Artificial Intelligence Research Institute, Moscow.