2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

Multi-task Network Embedding with Adaptive Loss Weighting

Fatemeh Salehi Rizi
University of Passau
Faculty of Computer Science and Mathematics
Passau, Germany
Fatemeh.SalehiRizi@uni-passau.de

Michael Granitzer
University of Passau
Faculty of Computer Science and Mathematics
Passau, Germany
Michael.Granitzer@uni-passau.de

***Abstract*—Network embedding is to learn low-dimensional representations of nodes which mostly preserve the network topological structure. In real-world networks, however, nodes are often associated with a rich set of attributes and labels which are potentially valuable in seeking more effective vector representations. To properly utilize this information, we propose a Joint Autoencoders framework for Multi-task network Embedding (JAME), which aims to encode a shared representation of local network structure, node attributes, and available node labels. Jointly embedding via multi-task learning is strongly dependent on the relative weighting between each task's loss function. Tuning these weights by hand is an expensive and difficult process, making multi-task learning prohibitive in practice. Therefore, we define an adaptive loss weighting layer capable of learning an optimal combination of loss weights during model training. This allows to dynamically updates task weights for controlling the influence of individual tasks based on task difficulty. Empirical evaluations on three real-world datasets demonstrate effectiveness of JAME compared to the relevant baseline methods.**

***Index Terms*—Network Embedding, Multi-task Learning, Adaptive Loss Weighting.**

I. Introduction

Network embedding projects nodes in a network to a low-dimensional vector space while preserving network structure and inherent properties. It has been used to solve many practical applications in various domains [1]–[5]. Traditionally, network embedding focused on preserving plain network structure, mapping nodes in the same neighborhood close to each other in the vector space. In real-world scenarios, however, nodes are often associated with a set of auxiliary information such as contents (e.g. profile attributes or textual features) or labels (e.g. community or affiliation group) which form labeled attributed networks. It has been shown that labels are strongly influenced by and inherently correlated to both network structure and attribute information [6]. Therefore, modeling and integrating node attribute proximity and labels into network embedding is potentially helpful in learning more effective and meaningful vector representations.

Recent attributed network embeddings [7]–[11] considered network structure and attributes, but ignored abundant labels associated to the nodes. To the best of our knowledge, LANE [6] represents the first effort to smoothly incorporate labels, attributes and network structure into embeddings preserving their correlations. LANE learns three types of latent representations via spectral techniques, however it faces several drawbacks, like, 1) expensive computations for matrix decomposition, 2) manually tuning weights for multiple objective functions, and 3) relatively low performance in node classification and link prediction.

To overcome the problems of the existing methods, we need to design an efficient multi-task learning framework with adaptive loss weighting. Inspired by joint autoencoders [12]–[15], we propose a **J**oint **A**utoencoders model for **M**ulti-task network **E**mbedding, called JAME, which encodes shared representations for nodes in the network via optimizing multiple objectives. In detail, our end-to-end model consists of three autoencoders: a network structure autoencoder, an attribute autoencoder, and a label autoencoder. In joint training phase, each autoencoder reconstructs its input data accurately by minimizing the corresponding reconstruction loss.

However, the performance of multi-task learning is highly dependent on an appropriate choice of weighting between each task's loss. Manually tuning loss weights is tedious specifically with a large number of learning tasks. To search for optimal weighting, we define an adaptive loss weighting layer additional to the joint model based on task difficulty. This allows to dynamically weight multiple learning tasks during training, where difficulty is the loss ratio between the batch loss and the initial loss. We control weights such that poorly trained tasks with ratio close to 1 contribute more to the overall loss and gradient. In summary, the main contributions of this paper are as follows:

- We propose a novel end-to-end model for labeled attributed networks, JAME, which aims at encoding joint representations of nodes via multi-task learning.
- We define an adaptive loss weighting layer which optimizes loss weights based on task difficulty.
- We empirically evaluate effectiveness of JAME on real-world datasets, showing its superior performance to the baseline methods.

The rest of the paper is organized as follows. Section II calibrates the formal notations and details of the proposed method.

IEEE/ACM ASONAM 2020, December 7-10, 2020
978-1-7281-1056-1/20/$31.00 © 2020 IEEE

Section III presents extensive experiments and evaluations.

II. Proposed Framework

Let $G = (V, E)$ be a labeled attributed network, where V is a set of n nodes and E is the set of edges. Each node in the network is associated with a set of attributes and label information indicating its affiliation group. We denote $\mathbf{A} \in \{0,1\}^{n \times n}$ as adjacency matrix, $\mathbf{X} \in \{0,1\}^{n \times f}$ as the attribute matrix with f attributes, and $\mathbf{Y} \in \{0,1\}^{n \times k}$ as label matrix with k categories. Every row x_i in the attribute matrix $\mathbf{X}$ describes the attributes associated with node i. $\mathbf{Y}_{ij} = 1$ in the label matrix indicates node i belongs to category j. The final embedding matrix is denoted as $\mathbf{U} \in \mathbb{R}^{n \times d}$ where $d \ll n$ is the embedding dimension size. Figure 1 illustrates our JAME model with three major parts: network structure autoencoder, attribute autoencoder, and label autoencoder. These autoencoders are jointly trained by sharing the learned structure $\mathbf{U}$ across different embedding tasks. JAME receives the adjacency matrix $\mathbf{A}$, the attribute $\mathbf{X}$, and the binary label $\mathbf{Y}$ as input to reconstruct $\hat{\mathbf{A}}$, $\hat{\mathbf{X}}$, and $\hat{\mathbf{Y}}$ in output. In the multi-task setting, JAME offers an adaptive loss weighting layer $\mathbf{L}$ capable of learning a weighted combination of several objectives. Each autoencoder learns the model parameters by minimizing the cross-entropy loss formulation.

To encapsulate the non-linear structure of the input graph, we employ non-linear activation functions into the basic autoencoders. More formally, we define the graph structure autoencoder as:

$$\mathbf{H}_a = \mathrm{ReLU}(\mathbf{A}W_1^{(1)} + b_1^{(1)}), \tag{1}$$

$$\mathbf{U} = \sigma(\mathbf{H}_a + \mathbf{H}_x + \mathbf{H}_y)W^{(2)} + b^{(2)}), \tag{2}$$

$$\hat{\mathbf{A}} = \sigma(\mathbf{U}W_1^{(3)} + b_1^{(3)}), \tag{3}$$

where $\mathbf{H}_a \in \mathbb{R}^{n \times s}$ is the hidden representation from the structure encoder with dimension size s. $W_r^{(l)}$ is the trainable weight matrix and $b_r^{(l)}$ is the bias of r^{th} autoencoder in l^{th} layer. Sigmoid $\sigma(.)$ and ReLU are non-linear element-wise activation functions. $\mathbf{H}_x \in \mathbb{R}^{n \times s}$ is the hidden representation from the attribute encoder, $\mathbf{H}_y \in \mathbb{R}^{n \times s}$ is the hidden representation from the label encoder, and $\mathbf{U} \in \mathbb{R}^{n \times d}$ is the shared representation matrix with dimension size d.

To capture attribute proximity associated to nodes, the attribute autoencoder is defined as:

$$\mathbf{H}_x = \mathrm{ReLU}(\mathbf{X}W_2^{(1)} + b_2^{(1)}), \tag{4}$$

$$\mathbf{U} = \sigma(\mathbf{H}_a + \mathbf{H}_x + \mathbf{H}_y)W^{(2)} + b^{(2)}), \tag{5}$$

$$\hat{\mathbf{X}} = \sigma(\mathbf{U}W_2^{(3)} + b_2^{(3)}), \tag{6}$$

Label information plays an essential role in determining the inscape of each node with strong intrinsic correlations to network structure and node attributes. Formally, the label autoencoder transformation is defined as:

$$\mathbf{H}_y = \mathrm{ReLU}(\mathbf{Y}W_3^{(1)} + b_3^{(1)}), \tag{7}$$

$$\mathbf{U} = \sigma(\mathbf{H}_a + \mathbf{H}_x + \mathbf{H}_y)W^{(2)} + b^{(2)}), \tag{8}$$

$$\hat{\mathbf{Y}} = \sigma(\mathbf{U}W_3^{(3)} + b_3^{(3)}), \tag{9}$$

In our joint embedding framework, we combine loss functions of different learning tasks in the loss weighting layer. The final loss $\mathcal{L}_f$ is calculated by linear combination of multiple weighted losses as follows:

$$\mathcal{L}_f = \mathrm{ReLU}(w_a\mathcal{L}_a + w_x\mathcal{L}_x + w_y\mathcal{L}_y), \tag{10}$$

where w_a, w_x, and w_y are weights for the network structure, attribute, and label embedding tasks respectively.

To control the influence of each task, we calculate the task difficulty as a ratio between the current loss and the initial loss in each batch. Task difficulty is a measure how well the model has trained for that task. During model training, we update task weights such that poorly trained tasks receive higher weights to contribute more to the final loss and gradient. The detail of our approach is described in Algorithm 1.

Algorithm 1 JAME Training with Adaptive Loss Weighting

```
Given a set of tasks {1, 2, ··· , T}
Initialize task weights w_t = 1 ∀t
Initialize model weights W^(l) ∀l
for each iteration i do
  for each input batch B do
    Compute each task loss L_(B) ∈ R^T
    Get the first batch loss L_(0,i) ∈ R^T
    Compute each task difficulty γ_t = L_(B) / L_(0,i) ∈ R^T
    for each task t do
      Update the task weight w_t = exp(γ_t) / Σ_k exp(γ_k)
    end for
    Compute final loss L_f = Σ_t w_t L_(B,t)
    Update W^(l) with respect to L_f
  end for
end for
```

III. Empirical Evaluation

In this section, we conduct experiments to evaluate the effectiveness of JAME on node classification, link prediction and attribute inferring tasks. Evaluations via graph analysis tasks can reveal whether our shared embeddings retain certain type of information. We first introduce the datasets, baseline methods, and experimental settings before the model training. We then investigate how the joint model with loss weighting affects the performance of each individual task.

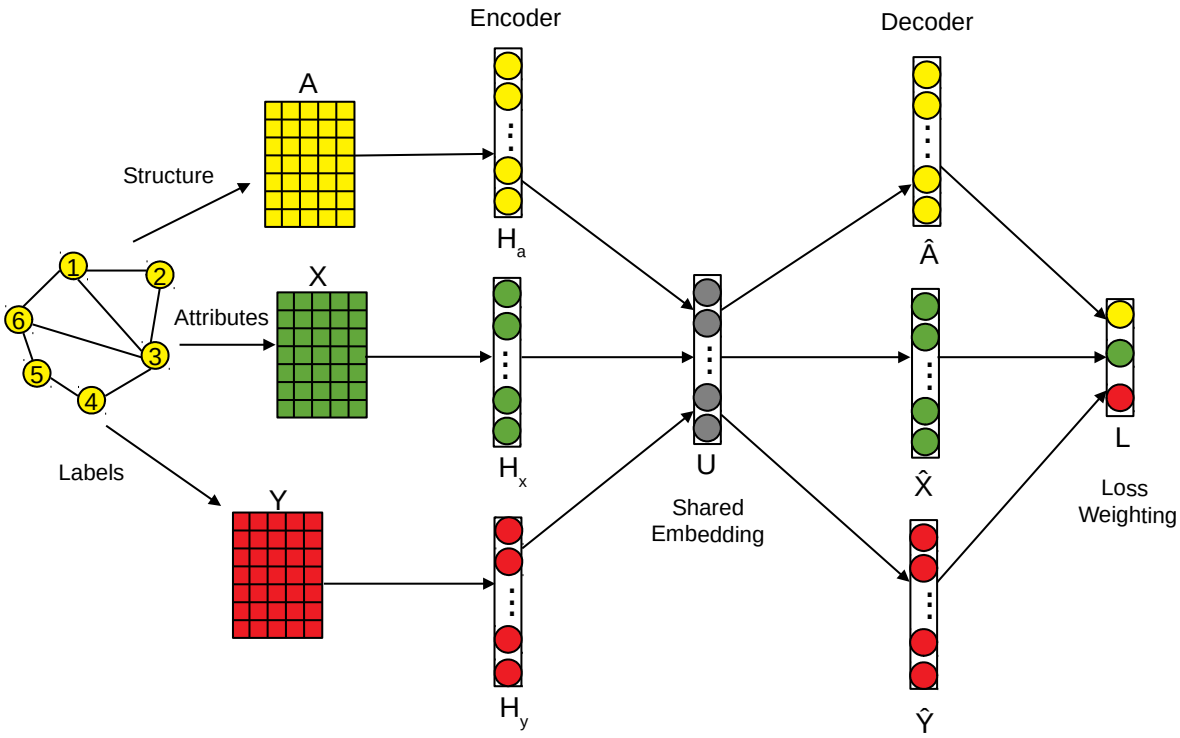

Fig. 1: The framework of our proposed JAME model. The model takes the adjacency matrix **A**, the attribute matrix **X**, and the label matrix **Y** as input and reconstructs $\hat{\mathbf{A}}$, $\hat{\mathbf{X}}$, and $\hat{\mathbf{Y}}$ as output. The shared embedding layer **U** aggregates information from structure, attribute and labels while loss weighting layer **L** learns optimal weights for loss functions based on task difficulty.

All the experiments report the mean results after 5-fold cross validation. Our reference code and data are available at https://github.com/fatemehsrz/JAME.

A. Datasets

We conduct our experiments on three widely used attributed network datasets: Cora, Citeseer and PubMed [9]–[11]. These datasets are citation networks where nodes are publications and edges are citation links. Attributes of each node are bag-of-words representations of the corresponding publications. The labels indicate research topics of the publications. The statistics of the datasets are provided in Table I.

TABLE I: Statistics of datasets.

Dataset	Nodes	Edges	Attributes	Labels
Cora	2,708	5,429	1,433	7
Citeseer	3,312	4,660	3,703	6
PubMed	19,717	44,338	500	3

B. Baselines

We compare JAME with the following relevant network embedding baselines:

- **LANE** is an attributed network embedding model which integrates labels into the final representations through eigen-decomposition of the graph affinity, attribute affinity and label affinity matrices [6].
- **CAN** learns network representations via a variational autoencoder model which consists of an inference model for encoding attributes into Gaussian distributions and a generative model for reconstructing both edges and attributes [9].
- **attri2vec** integrates the network structure and node attributes together seamlessly by a defined transformation function [11].
- **DANE** captures the network non-linearity and preserves the proximity of both topological structure and node attributes via a joint autoencoder [7].

The parameters of all baselines are tuned to be optimal as default settings.

C. Multi-task Weighting

We first train JAME to observe the influence of each task to encode the network information, then we evaluate the shared representations through graph analysis tasks. According to the algorithm 1, all loss weights are initialized to 1.0 and the model adjusts these weights based on task difficulty. Table II demonstrates in each training step loss weights are optimized to give a higher weight to the hardest task. In our datasets, reconstructing the group labels is found to be the hardest task and gains consistently the highest weight in 10 iterations. Figure 2 shows change of final loss after each iteration on

TABLE II: Multi-task weighting based on task difficulty for different datasets.

Iteration	Cora			Citeseer			PubMed		
	w_a	w_x	w_y	w_a	w_x	w_y	w_a	w_x	w_y
1	0.933	0.942	**0.946**	0.921	0.932	**0.935**	0.913	0.926	**0.933**
2	0.876	0.889	**0.922**	0.842	0.857	**0.925**	0.868	0.886	**0.928**
3	0.801	0.823	**0.919**	0.762	0.783	**0.926**	0.799	0.875	**0.922**
4	0.735	0.762	**0.917**	0.673	0.711	**0.924**	0.660	0.764	**0.920**
5	0.672	0.708	**0.919**	0.596	0.645	**0.923**	0.520	0.567	**0.917**
6	0.615	0.634	**0.916**	0.516	0.568	**0.921**	0.483	0.553	**0.911**
7	0.533	0.583	**0.913**	0.432	0.482	**0.922**	0.455	0.542	**0.908**
8	0.464	0.535	**0.914**	0.358	0.436	**0.919**	0.423	0.517	**0.901**
9	0.403	0.477	**0.912**	0.264	0.348	**0.918**	0.342	0.432	**0.898**
10	0.335	0.418	**0.910**	0.221	0.283	**0.916**	0.313	0.415	**0.897**

Cora, Citeseer and PubMed. From Figure 2 and Table II can be seen that our loss weighting layer assigns optimal weights to each individual task while the final loss naturally drops and the model converges.

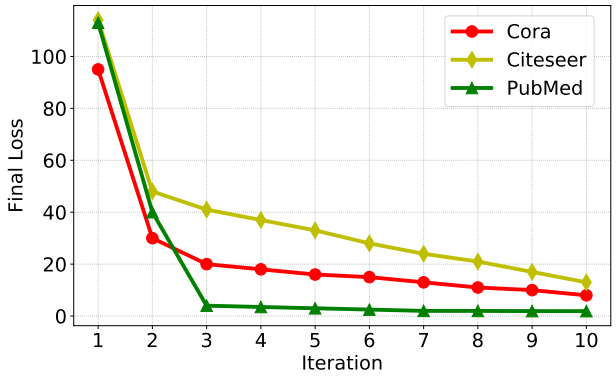

Fig. 2: Change of final loss over several iterations during training on all datasets.

D. Node Classification

Node classification is a widely adopted task for validating quality of network embeddings [16]. Similar to LANE, we first randomly divide all of the n nodes into a training set $(\mathbf{A}_{train}, \mathbf{X}_{train}, \mathbf{Y}_{train})$ and a test set $(\mathbf{A}_{test}, \mathbf{X}_{test}, \mathbf{Y}_{test})$. We then employ JAME architecture itself for node classification by randomly observing 10% to 90% of labels as training set and predicting the rest. The goal is to predict the label of each instance in the test set, given its network structure and attributes. For the other baselines, we conduct node classification directly applying logistic regression to the latent representations. For fair comparisons, we report Macro-F1 scores as a metric to measure the performance after 5-fold cross validation. From Table IV, we can find that our proposed JAME consistently outperforms the baselines on all of the datasets. It shows in a multi-task setting, tasks compensate each other by learning from the shared information knowing that links, attribute and labels are correlated. As we control the task weights, label embedding can trained as effective as other tasks which explains our better results compared to baseline models. DANE leverages two autoencoders to jointly encode link and attribute information, hence it also presents fairly good performance in all training percentages. Overall, the presented results indicate effectiveness and robustness of JAME to capture and encode the label information associated to the network.

E. Link Prediction

Link prediction is a commonly used task to demonstrate the meaningfulness of the vector representations for preserving local connections in the network. To carry out the link prediction, we follow Grover et al. [17] experiments which gather positive and negative edges. The positive examples are obtained by randomly removing 50% of the existing edges from the original graph whereas negative examples are node pairs which are not connected by edges (non-existing edges). We construct edge features from node representations by the Hadamard product to feed a logistic regression. Table III shows the link prediction performance of JAME and the baselines for different methods measured by area under ROC curve (AUC) and average precision (AP). As shown for all datasets, JAME obtains highly competitive performance with CAN to predict the missing links in the networks. CAN exploits variational autoencoders along with graph convolutions to encode non-linearity in the graph structure. JAME differently takes the advantage of observing links, attribute and labels during representation learning. This allows shared embeddings to encode more meaningful integrated information into the vector space.

TABLE III: Comparison of the baseline methods for link prediction.

Method	Cora		Citeseer		PubMed	
	AUC	AP	AUC	AP	AUC	AP
JAME	**0.987**	**0.972**	**0.985**	**0.979**	**0.983**	**0.980**
LANE	0.860	0.857	0.771	0.764	0.883	0.879
CAN	0.985	0.971	0.984	0.977	0.980	0.977
DANE	0.897	0.886	0.938	0.923	0.969	0.951
attri2vec	0.872	0.867	0.796	0.791	0.890	0.885

TABLE IV: Node classification performance of different methods observing different percentages of labeled nodes.

Dataset	Method	Macro-F1								
		10%	20%	30%	40%	50%	60%	70%	80%	90%
Cora	JAME	**0.816**	**0.824**	**0.839**	**0.847**	**0.855**	**0.868**	**0.879**	**0.885**	**0.891**
	LANE	0.663	0.671	0.684	0.715	0.726	0.739	0.753	0.785	0.799
	CAN	0.733	0.752	0.768	0.773	0.788	0.794	0.806	0.814	0.822
	DANE	0.778	0.795	0.812	0.822	0.837	0.854	0.861	0.869	0.877
	attri2vec	0.695	0.713	0.729	0.732	0.746	0.767	0.788	0.792	0.806
Citeseer	JAME	**0.731**	**0.739**	**0.755**	**0.778**	**0.786**	**0.790**	**0.798**	**0.804**	**0.810**
	LANE	0.503	0.549	0.570	0.579	0.581	0.591	0.606	0.624	0.636
	CAN	0.577	0.606	0.613	0.619	0.628	0.632	0.638	0.641	0.642
	DANE	0.604	0.633	0.671	0.678	0.696	0.705	0.723	0.735	0.745
	attri2vec	0.556	0.571	0.614	0.650	0.656	0.662	0.670	0.666	0.682
PubMed	JAME	**0.868**	**0.873**	**0.879**	**0.881**	**0.884**	**0.886**	**0.888**	**0.894**	**0.898**
	LANE	0.787	0.792	0.795	0.799	0.820	0.824	0.828	0.833	0.837
	CAN	0.793	0.796	0.801	0.809	0.812	0.816	0.820	0.825	0.829
	DANE	0.857	0.864	0.870	0.873	0.874	0.876	0.878	0.880	0.882
	attri2vec	0.843	0.850	0.853	0.858	0.860	0.862	0.863	0.865	0.866

F. Attribute Inference

Attribute inference aims at predicting the value of attributes associated to the nodes in the network. We follow CAN [9] experiments for attribute inference which conduct a binary classification task applying logistic regression. To evaluate our learned embeddings, we randomly divide nodes into a training (80%), and a test set (20%). We employ AUC and AP to measure the attribute inference performance since attributes of the nodes are 0/1-valued. Table V presents the performance of JAME against the baseline models on all datasets. As can be seen, JAME achieves improvements in both AUC and AP over all of our datasets. The superiority may result from the fact that our model optimizes a loss function consisting of the reconstruction error of all the attributes.

TABLE V: Comparison of the baseline methods for attribute inference.

Method	Cora		Citeseer		PubMed	
	AUC	AP	AUC	AP	AUC	AP
JAME	**0.968**	**0.961**	**0.971**	**0.963**	**0.681**	**0.677**
LANE	0.872	0.867	0.880	0.875	0.590	0.587
CAN	0.932	0.916	0.954	0.939	0.670	0.652
DANE	0.917	0.910	0.928	0.921	0.639	0.635
attri2vec	0.881	0.878	0.889	0.883	0.610	0.598

ACKNOWLEDGMENT

The presented work was developed within the Provenance Analytics project funded by the German Federal Ministry of Education and Research, grant agreement number 03PSIPT5C.

REFERENCES

[1] K. Skorniakov, D. Turdakov, and A. Zhabotinsky, "Make social networks clean again: Graph embedding and stacking classifiers for bot detection." in *CIKM Workshops*, 2018.

[2] F. S. Rizi, J. Schloetterer, and M. Granitzer, "Shortest path distance approximation using deep learning techniques," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2018, pp. 1007–1014.

[3] F. S. Rizi and M. Granitzer, "Predicting event attendance exploring social influence," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM, 2019, pp. 2131–2134.

[4] F. S. Rizi and M. R. Khayyambashi, "Profile cloning in online social networks," *International Journal of Computer Science and Information Security*, vol. 11, no. 8, p. 82, 2013.

[5] F. Salehi Rizi and M. Granitzer, "Properties of vector embeddings in social networks," *Algorithms*, vol. 10, no. 4, p. 109, 2017.

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

[7] H. Gao and H. Huang, "Deep attributed network embedding." in *IJCAI*, vol. 18, 2018, pp. 3364–3370.

[8] X. Huang, J. Li, and X. Hu, "Accelerated attributed network embedding," in *Proceedings of the 2017 SIAM international conference on data mining*. SIAM, 2017, pp. 633–641.

[9] Z. Meng, S. Liang, H. Bao, and X. Zhang, "Co-embedding attributed networks," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 2019, pp. 393–401.

[10] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, and C. Zhang, "Binarized attributed network embedding," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 1476–1481.

[11] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Attributed network embedding via subspace discovery," *arXiv preprint arXiv:1901.04095*, 2019.

[12] B. E. Meir, T. Michaeli *et al.*, "Joint auto-encoders: a flexible multi-task learning framework," *arXiv preprint arXiv:1705.10494*, 2017.

[13] E. Banijamali, A.-H. Karimi, A. Wong, and A. Ghodsi, "Jade: Joint autoencoders for dis-entanglement," *arXiv preprint arXiv:1711.09163*, 2017.

[14] C. Cadena, A. R. Dick, and I. D. Reid, "Multi-modal auto-encoders as joint estimators for robotics scene understanding." in *Robotics: Science and Systems*, vol. 5, 2016, p. 1.

[15] F. Zhuang, D. Luo, X. Jin, H. Xiong, P. Luo, and Q. He, "Representation learning via semi-supervised autoencoder for multi-task learning," in *2015 IEEE International Conference on Data Mining*. IEEE, 2015, pp. 1141–1146.

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

[17] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 855–864.