Recurrent Attention Walk for Semi-supervised Classification

Uchenna Akujuobi
King Abdullah University of Science and Technology (KAUST), Saudi Arabia
uchenna.akujuobi@kaust.edu.sa

Qiannan Zhang
King Abdullah University of Science and Technology (KAUST), Saudi Arabia
qiannan.zhang@kaust.edu.sa

Han Yufei
NortonLifelock Research Group, France
yufei_han@symantec.com

Xiangliang Zhang
King Abdullah University of Science and Technology (KAUST), Saudi Arabia
xiangliang.zhang@kaust.edu.sa

ABSTRACT

In this paper, we study the graph-based semi-supervised learning for classifying nodes in attributed networks, where the nodes and edges possess content information. Recent approaches like graph convolution networks and attention mechanisms have been proposed to ensemble the first-order neighbors and incorporate the relevant neighbors. However, it is costly (especially in memory) to consider all neighbors without a prior differentiation. We propose to explore the neighborhood in a reinforcement learning setting and find a walk path well-tuned for classifying the unlabelled target nodes. We let an agent (of node classification task) walk over the graph and decide where to move to maximize classification accuracy. We define the graph walk as a partially observable Markov decision process (POMDP). The proposed method is flexible for working in both transductive and inductive setting. Extensive experiments on four datasets demonstrate that our proposed method outperforms several state-of-the-art methods. Several case studies also illustrate the meaningful movement trajectory made by the agent.

CCS CONCEPTS

• **Theory of computation → Reinforcement learning**; **Semi-supervised learning**; • **Computing methodologies → Partially-observable Markov decision processes**.

ACM Reference Format:
Uchenna Akujuobi, Qiannan Zhang, Han Yufei, and Xiangliang Zhang. 2020. Recurrent Attention Walk for Semi-supervised Classification . In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371853

1 INTRODUCTION

Network data model interactions between entities such as humans [21], genes [32], and publications [29]. Networks with node or edge content information are known as *Attributed Networks*. For example, in an attributed web network, nodes are attributed with full website content and edges are attributed with the mention contexts (the sentence encompassing the website mention). A variety of graph mining tasks on attributed networks have been exploited as popular research topics, such as graph embedding [11, 15, 22, 35], community detection and clustering [10, 24], classification [17, 31, 36], user profiling [23] and recommendation [4, 7]. In this paper, we focus on the problem of semi-supervised node classification on attributed graphs with both nodes and edge contents.

Definition 1.1. Semi-supervised Node Classification: Given an attributed graph $G = \{V, E, X_v, X_e\}$, where node set V contains a small subset of labelled nodes $V_l = < v_i, y_i >, 1 \leq i \leq |V_l|$ and the remaining nodes $V_u = V/V_l = < v_j >, 1 \leq j \leq |V_u|$ are unlabeled. x_v and x_e denote the attributes of nodes and edges in the graph G, respectively. The goal is to infer the labels of the unlabeled nodes V_u based on the available but limited node labels. Learning from the graph content and structure information.

The main solutions to this problem are categorized into two modes: *unsupervised embedding + classifier*, and *semi-supervised learning on graph*. The approaches in the first branch apply a classifier on embeddings of graph nodes learned using methods like Node2Vec [12], DeepWalk [25], or TADW [35]. The algorithms belonging to the second branch directly learn from the graphs, e.g., non-attributed (Label propagation [39] and label spreading [37]), and attributed graphs embedding (GCN [17], Planetoid [36], DGM [2], and [30, 31]). The core ideas behind these approaches are to 1) jointly learn from the graph structure and the node attributes (most of them are not designed to include edge contents); and 2) aggregate the content of neighboring nodes at different levels of relevance, from immediate neighbors to neighbors k-hop away. One limitation of these approaches is the performance downgrade caused by the noisy information from an exponentially increasing number of expanded neighborhood members [38], even though considering high-order structures in graphs might be beneficial for some graph-based problems [20, 26, 27]. Another issue is the high computational cost, especially in memory cost, caused by the exponentially increasing number of expanded neighbors.

Furthermore, most of the previously proposed semi-supervised methods are transductive, and thus cannot fit to the situations where new nodes are observed and inserted to the graph. However, deriving embeddings and conducting classification in an inductive way for new unseen nodes is highly demanding in real-world settings, e.g., classifying a new published paper/website. Inductive approaches also facilitate the generalization across attributed

WSDM '20, February 3–7, 2020, Houston, TX, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6822-3/20/02…$15.00
https://doi.org/10.1145/3336191.3371853

graphs with similar feature spaces [13, 36]. It is thus desirable to design approaches that are flexible for *both transductive and inductive* setting.

To *reduce the scope of neighbors* to be evaluated in the semi-supervised node classification problem and *maintain an inductive property*, we propose a recurrent attention framework to learn to explore neighborhoods. In this way we guide neighborhood exploration to better serve the goal of node classification, compared to purely random walk. We pose the learn-to-walk task as a partially observable markov decision process (POMDP) problem and attack it with reinforcement learning. To summarize, we address the node classification problem by letting an agent make recurrent decisions on next nodes to visit in its walk on the graph. This process can be considered as a recurrent attention-based walk. Therefore, we call our proposed model, *Recurrent Attention Walk (RAW).* Comparing to other popular semi-supervised graph-based node classification approaches, RAW has the following advantages:

- RAW uses a *recurrent-attention* strategy, while attention-based node classification approaches like GAT [31] and AGNN [30] are based on a *self-attention* strategy, which accessing high-order neighbors by iteratively aggregating one-hop neighbors. By contrast, our recurrent-attention strategy learns how to walk and thus can find the walk path well tuned for classifying the target nodes, and thereby minimizing the noisy information obtained.
- RAW thus is more efficient than GCN [17] and GAT like approaches on memory cost, because RAW reduces the number of nodes to aggregate per hop.
- RAW is usable in both transductive and inductive settings. We perform extensive experiments on real-world large datasets. The result shows that RAW has superior performance, significantly on inductive setting.
- The walking path generated by RAW can be used to interpret the decision making process and infer class label dependency, as shown in our case studies.

2 PREVIOUS WORK

In general, solutions for the studied problem (as defined in the previous section) target on minimizing the loss: $E = \mathcal{E}_l(f(x), y) + \mathcal{E}_r(f(x))$, where $\mathcal{E}_l(f(x), y)$ is the supervised loss function and $\mathcal{E}_r(f(x))$ is the regularizer. The regularizer $\mathcal{E}_r(f(x))$ penalizes a model for assigning different labels $f(x_i) \neq f(x_j)$ to similar nodes x_i and x_j, which are close on the graph and have similar content.

Zhu and Ghahramani [39] proposed a transductive label propagation model following the theoretical framework of Gaussian Random Fields to classify nodes in a nearest neighbor graph of a semi-supervised data set. Some other works follow a two-step solution by first learning node embeddings with unsupervised methods [12, 25, 28], and then building classifiers on the learned node embedding to infer the unknown labels. Since the embedding is learned in a unsupervised way, it is general enough to be deployed across different tasks (e.g., clustering and link prediction). However, it is not tailored to fit the use in node classification[1].

[1]We are aware of a big group of related work to our study in graph embedding. In this section, we focus on the most relevant ones solving node classification in semi-supervised learning. Comprehensive discussion of other unsupervised graph embedding for both plain and attributed graphs can be found at [5].

Recent decades have witnessed a new trend of research on node classification, which focuses on conducting semi-supervised learning on graphs. Yang et al. [36] proposed a node embedding method to jointly predict the neighborhood context and labels of graph nodes. Kipf et al. [17] proposed the use of graph convolutional networks (GCN) for graph-based semi-supervised learning. Zhuang and Ma [40] extended the idea of GCN by considering global and local consistency. Akujuobi et al. [2] studied the use of deep generative models for graph-based semi-supervised learning. Hamilton et al. [13] proposed GraphSAGE, an inductive method that computes a node representation by applying an aggregation function over a fixed sample length of node neighbors.

In general, few of the above-discussed approaches attentively selects the relevant neighboring nodes. The relevance of all neighboring nodes may be implicitly encoded in the aggregation procedure. However, the action on all neighboring nodes without prior preference introduces noisy information due to the exponentially increasing of nodes as the exploration range of the neighborhood extends. To suppress the potential impact of noisy information during aggregating the node neighbors, we propose an attention-based reinforcement learning method for node classification. Next, we survey the use of attention mechanisms and use of reinforcement learning on graph-based problems.

2.1 Attention-based Node Classification

We can consider selecting the relevant neighboring nodes to visit from the perspective of attention mechanism. Introducing attention mechanism allows the models to focus on the relevant areas of graphs for a given learning task, such as node classification [19]. Abu-El-Haija et al. [1] extends deepwalk by using the attention to guide random walk. Thekumparampil et al. [30] introduced attention to the GCN propagation layers to assign more weight to relevant neighbors of each node. Velicovic et al. [31], extend the idea of GraphSAGE by introducing the use of attention in the node neighbor sampling. Note that the attention neighboorhood per node in the papers, as mentioned above, are the nodes one-hop away from a given node. Our model removes this restriction and thus can achieve better graph exploration. Also, most of these proposed methods do not scale well on large graphs with non-sparse feature vectors as node attributes (i.e., continuous vectors). Furthermore, all these attention models share a *self-attention* strategy. Specifically, hidden states of each node are computed by attending their neighbors. Thus, by stacking more layers (i.e., k-layers), the nodes aggregate information from neighbors up to k-hop away. We consider a *recurrent-attention* strategy, where hidden states of each node are computed by enforcing attention on a recurrent walk on the graph. This strategy reduces the number of nodes to be considered per hop and thereby, minimizing the noisy information obtained. Also, it enables us to evaluate which nodes are more useful based on the information it already gathered from previous hops, and which areas of the graph to explore.

2.2 Reinforcement on Graph-Structured Data

Several works have studied the application of reinforcement learning on graph-structured data. Hoshen [14] applied soft attention on

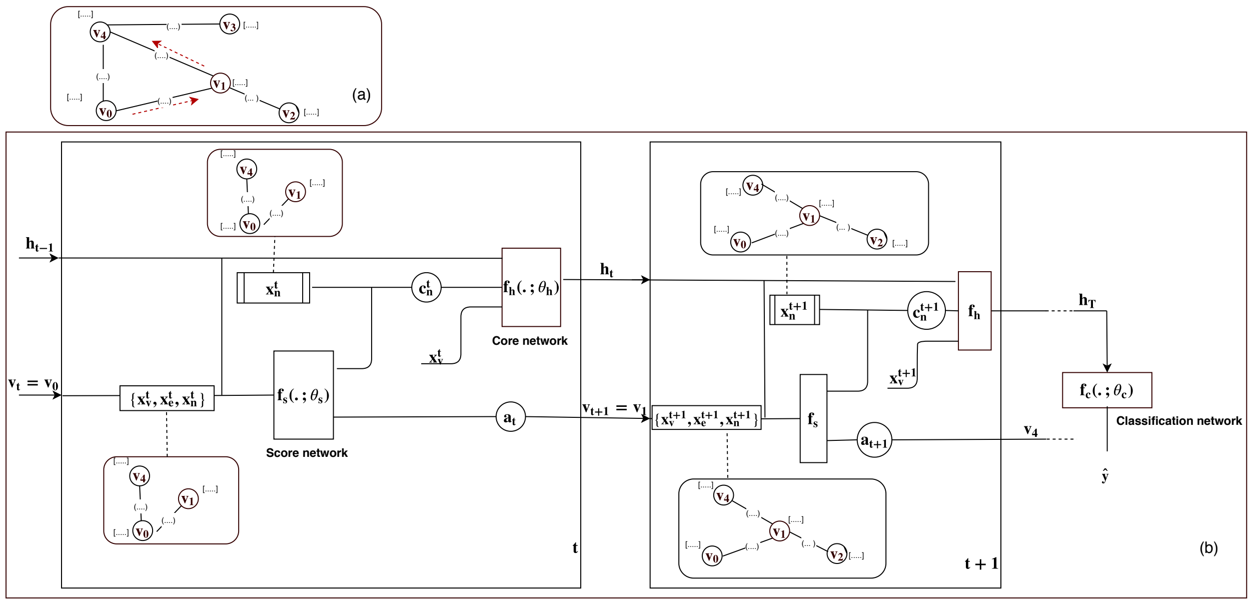

Figure 1: The proposed RAW model. See section 3.1 for description.

the matrix pair-wise interactions between game agents to select information from relevant agents. Jiang et al. [16] introduced a graph convolutional reinforcement learning method to learn multi-agent cooperation. Xiong et al. [34] proposed a model for finding multi-hop relation paths in knowledge graphs. None of these models are designed to select the optimal movement trajectory (path) for node classification. Although with the similar motivation, our parallel work [3] focuses on multi-label problem, rather than multi-class problem.

The GAM (Graph Attention Model) proposed by Lee et al. [18] is an RNN model for graph classification (not node classification), through attention on the graph structural composition. The graph classification differs from node classification on the prediction goal. Thus, the GAM model cannot be applied for node classification as the embedding learned from the graph classification is based on recurrent attention on nodes with random starting nodes. It is not designed to encode a linear combination of the node embeddings. Secondly, the GAM method evaluate the graph label prediction per step iteratively, which is not feasible for node classification in large graphs. GAM also assumes that all the nodes know node types (labels), which does not hold in the settings of semi-supervised node classification.

3 METHODOLOGY

3.1 Model Description

We model the sequential decision making of which next node to visit by Recurrent Neural Networks (RNN) [8] to capture the recurrent dependency in the walk path on the graph. Sequential decision making describes a situation where the decision maker takes its action upon successive observations. The choice of action depends on the expected benefit that can be potentially gained in the future. Given this setting, Markov Decision Process (MDP) provides a coherently appropriate solution to the sequential decision making problem. Nevertheless, exploration of the walk path in an attributed graph violates the Markov property: the observations of the agent at each step should be rich enough to distinguish states of the agent from one to another. In the walk over the graph, observing only the attributes and the neighbors of the current node is not enough to capture all topological information. Therefore, the neighborhood exploration task reduces to a Partially Observable MDP (POMDP) problem. To attack this issue, we encode the past histories of walk paths with RNN to augment state representation of the agent, which facilitates the process of policy learning.

As illustrated in Figure 1, the proposed RAW is composed of 3 networks: the core network, the score network, and the classification network. With a small example of an attributed graph in Figure 1 (a), the whole process can be explained as follows. At the current time t, the agent is at node v_0 and deciding the next visit at time $t+1$, thus $v_t = v_0$. In the left of Figure 1 (b), the score network $f_s(.;\theta_s)$ takes as input the previous history h_{t-1}, the current node attribute x_v^t, and the attributes of the current neighborhood observation of the agent, which includes the attributes of the immediate node neighbors and edges $\{x_n^t, x_e^t\}$. The job of the score network is to generate a score for each node neighbor. The generated score in range [0, 1] denotes the relevance of a node neighbor to the given node. After the relevance score is normalized, the next node v_{t+1} to visit is sampled from its neighbors in proportion to their relevance. The core network takes over after relevance score is generated. By selectively aggregating the embeddings of neighboring nodes x_n^t based on the score network, an immediate neighborhood information c_n^t is formed (see section 3.1.2 for more details). The core network $f_h(.;\theta_h)$ takes as input the neighborhood aggregation c_n^t , the previous history h_{t-1}, and the current node embedding x_v^t, and outputs the current walk history h_t. This process leads the agent to v_1 at time $t+1$, and it repeats to make a move to

v_4 at $t+2$, etc. After a fixed number of steps T, the **final** vector h_T summarizing the information obtained from the graph walk is passed to the classification network $f_c(.;\theta_c)$ for the label prediction of the starting node. See algorithm 1 for more details.

RAW is also applicable in inductive setting, where the walk policy is learned based on the nodes available in the graph. Given a new node added to the graph, the agent initiates a walk from the new unlabelled node guided by the learned policy based on $f_s(.;\theta_s)$ and $f_h(.;\theta_h)$, and finally uses $f_c(.;\theta_c)$ for classification.

Algorithm 1: Classifying node v_1

```
Input: Graph $G$, start node $v_1$, history vector $h_0$ (a vector of
       zeros), node and edge embeddings $x_v$, $x_e$
Result: label prediction for node $v_1$
1 for $t \leftarrow 1 \cdots T$ do
2     Obtain the current node embeddings $x_v^t$ of the current node $v_t$;
        $x_e^t$ for edges connecting to $v_t$; and $x_n^t$ for neighboring nodes ;
3     Assign relevance value to each neighbor observation
        $\varphi^t = f_s(h_{t-1}, x_v^t, x_e^t, x_n^t; \theta_s)$;
4     Sample next node $v_{t+1}$ from a categorical distribution
        $Cat(.|P_t(\varphi^t))$ over the neighbors ;
5     Extract the relevant neighbor information $c_n^t$ ;
6     update the history vector $h_t = f_h(h_{t-1}, x_v^t, c_n^t; \theta_h)$ ;
7 end
8 Obtain the label prediction of the start node $y_{v_1} = f_c(h_T; \theta_c)$
```

3.1.1 *Information Flow.*

The information flow in RAW has been described above as a sequential decision process, formulated as POMDP. At the time t, the agent, which can only observe its one-hop neighbors at the current node, cannot capture the complete topological information in the large graph. Formulating as POMDP allows for a careful treatment to the incomplete observation problem, which is necessary in our case.

To address the uncertainty of observation, we augment the observation by integrating the information from the previous walk path. This information is encoded recurrently by RNN and updated as the agent traverses.

At each step, the agent takes action based on its observation, including the previous history h_{t-1}, the current node attribute x_v^t, and attributes of its immediate node and edge neighbors, x_n^t and x_e^t respectively, transiting to the next node v_{t+1}. The history h_{t-1} acts as a summary of the previous observations in the graph walk, combined with the current observation, the history is updated by the core network $h_t = f_h(h_{t-1}, x_v^t, c_n^t; \theta_h)$, which has GRU at its core and is formulated as:

$$\begin{aligned} z_t &= \sigma_g(W^z[x_v^t + c_n^t] + U^z h_{t-1} + b^z), \\ r_t &= \sigma_g(W^r[x_v^t + c_n^t] + U^r h_{t-1} + b^r), \\ h_t' &= \sigma_{h'}(W[x_v^t + c_n^t] + r_t \circ U h_{t-1} + b), \\ h_t &= z_t \circ h_t' + (1 - z_t) \circ h_{t-1}. \end{aligned} \tag{1}$$

where $\circ$ and $+$ denote element-wise multiplication and vector concatenation respectively. The variable z_t is the update gate which determines the amount of past information to overwrite, r_t is the reset gate which decides the amount of past information to compute a new memory content, h_t' is the current memory content, and h_t is the output vector containing information from the current unit and previous units. The variables W and U are the weights; x_v^t is the node attribute of the current node, c_n^t is the aggregated attribute of the relevant current node neighbors (see section 3.1.2), and b^z, b^r, b are the bias vectors.

At the end of the walk ($t = T$), the core network $f_h(.;\theta_h)$ produces h_T, the embedding of the full trajectory started from the target node. To classify the target node, h_T is given to the classification network $f_c(.;\theta_c)$, modeled as a 2-layer neural network, to predict the class label.

3.1.2 *Action.*

The agent is expected to take actions to choose the most relevant nodes to visit, and finally collect sufficient information for classifying the target node. Therefore, we can determine the next node to select as an action a_t based on the output $\varphi^t = f_s(h_{t-1}, x_v^t, x_e^t, x_n^t; \theta_s)$ of the score network. The output φ^t is a measure of relevance between node v_t and its neighbors, and thus, is useful for deciding which of the neighboring nodes are relevant to the current node v_t. φ^t will be used for the next node selection, and also serve for the history aggregation update.

The score network is modeled using a sigmoid activation function. Values in φ^t are thus between 0 and 1 for each neighboring node. For the sake of better exploration, a stochastic policy π is adopted to make the choice of the next node v_{t+1} to visit via sampling under the categorical distribution $P = Cat(.|\varphi^t)$, after normalizing φ^t: $P = Cat(.|\varphi^t) = \frac{1}{\sum_{v_k} \varphi_{v_k}^t} \times \varphi_{v_k}^t$.

Then the aggregation of relevant neighboring nodes is conducted as:

$$c_n^t = \sum_{v_k} x_k^t \times \mathbb{1}(\varphi_{v_k}^t - 0.5); \quad v_k \in N_r(v_t), \tag{2}$$

where $N_r(v_t)$ is the set of nodes in the one-hop neighborhood of the current node v_t, x_k^t is the node attribute of node v_k in the set, and $\varphi_{v_k}^t$ is the relevance score of v_k. The indicator function $\mathbb{1}(.)$ outputs 1 when positive and 0 otherwise.

3.1.3 *Reward.*

In our model, the performance of a graph walk path (trajectory) would be measured at the end, like evaluating a student passing or failing a course in the final exam after one-semester recurrent study. Specifically, the agent gets an immediate reward $r_t = 1$ at the last step T, if the label prediction at the end ($t = T$) is correct and $r_t = 0$ otherwise. The goal of the agent is to take actions with large reward to go, $R = \sum_{t=1}^{T} r_t$. This reward encourages the agent to explore nodes on the graph that improve the final predictive performance.

The setting of T is application dependent. A large T allows for long-run exploration but increases computational cost, while a small T limits the knowledge to aggregate. We have a sensitivity analysis about T in the experimental section.

3.2 Training

The final target of our model is to classify an unknown node. Given a trained model, the agent starts from the unknown node, follows the policy to traverse the graph and assigns a label to the given node by the classification network at the end of the graph walk. To fulfill the goal of the model, it is required to learn a good walk policy and classification network. And we conduct the training process in a semi-supervised manner integrating both labeled nodes V_l and unlabeled ones V_u efficiently.

We augmented the observation to tackle the partial observation problem. But to indicate the property of POMDP, we adopt $o_{1:t}$ to represent the partial observations along the path until time t, while in our study augmented observation $\{h_{t-1}, x_v^t, x_e^t, x_n^t\}$ acts as $o_{1:t}$. We would like to train the policy $\pi(a_t|o_{1:t};\theta)$ to learn the mapping from the observation space to the action space. Since the policy will take its history from previous transitions as one part of its input, the training of policy will in fact result in an improved core network to provide better history embedding and a score network for more accurate score generation. Therefore, we train the parameters $\theta = \{\theta_s, \theta_h\}$ together for the policy. The policy objective is the reward in the future over the expectation of the graph walk paths following the current policy, which is $\mathcal{J}(\theta) = \mathbb{E}_{(\pi;\theta)}\big[\sum_{t=1}^{T} r_t\big]$.

However, computing the objective function is tough in practice. The expectation over joint probability distribution of walk paths is hard to measure. Therefore, adopting the trick of log derivative to change the gradient of the expectation to the expectation of the gradient, the algorithm REINFORCE for POMDP in [33] could take gradients of the objective as following:

$$\begin{aligned}\nabla_\theta \mathcal{J} &= \sum_{t=1}^{T} \mathbb{E}_{p(o_{1:T};\theta)}[\nabla_\theta \log \pi(a_t|o_{1:T};\theta)R] \\ &\approx \frac{1}{M}\sum_{i=1}^{M}\sum_{t=1}^{T-1} \nabla_\theta \log \pi(a_t^i|o_{1:t}^i;\theta)\gamma^{T-t}R^i. \end{aligned} \tag{3}$$

The o^i's are the roll-out sequences obtained from running the agent π_θ for $i = 1, ..., M$ episodes, and $\gamma \in (0, 1]$ is a discount factor that gives more preference to actions performed closer to the time the final prediction is made (i.e., $t = T$). R^i is the reward to go of the episode i. We only adjust the log-probabilities for steps $1 \cdots T-1$ since there is no choice of next node to visit at time T.

On breaking down the joint distribution of the trajectory, the gradient could be estimated by sampling different roll-outs, each running the agent for T limited time steps, from which obtaining the rewards of observations and actions for the estimate. This trial-and-error method is conducted under the current policy, thus providing feedback to the policy and guiding it towards better regions in the parameter space. The information of policy gradients will be back propagated to update parameters of the policy. The differentiable score network f_s and core network f_h, represented as neural networks, will be updated. This intuition follows: any gradients of the policy that correspond to high rewards are higher weighted, making roll-outs with higher rewards more likely.

The expected reward of the roll-out only depends on the classification at the end of the walk. Therefore, with the roll-outs starting at labeled nodes but traversing over unlabelled nodes, the training is allowed in a semi-supervised manner to use the unlabeled ones as the transitional nodes. It effectively integrates labeled and unlabeled nodes to utilize their information to the maximal extent. Besides, high variance from sampling still exists, though the estimate is an unbiased one. The reward setting alleviates this problem in sampled trajectories to some degree by reducing the reward collected at the intermediate steps of roll-outs.

Table 1: Statistics of datasets used in the evaluations.

	# Nodes	#Edges	#labels	# Labeled nodes
CoraL1	31,314	133,491	10	21,112
CoraIDA	31,314	133,491	23	9,743
DBLP	1,037,692	7,371,345	6	238,350
DELVE	1,229,280	4,322,275	7	665,495

For the classification $f_c(.;\theta_c)$ network, we define the loss to include the classification error (cross-entropy) and L2 regularization. The classification network is trained in supervised way via gradient descent by itself and provides reward signals to the agent, while score network is trained using REINFORCE. The whole model is trained end-to-end.

4 EXPERIMENTS

In this section, we present and discuss the extensive experiments and results obtained. We first introduce the four used datasets, the comparison methods, the implementation details, and parameters used for all the models. Finally, we report the results obtained and also present a case study.

4.1 Datasets

The evaluation datasets are citation networks constructed from Cora, DBLP, and Delve datasets. For each of the resulting paper in the citation networks, we extract the titles (and abstract when available). We also extract the citation context (sentences encompassing the citation) of the references from the papers when available. The statistics of the datasets are shown in Table 1.

CoraL1: The Cora dataset is extracted from the original Cora data[2]. We excluded papers with missing titles and papers with no citation and references (isolated papers). We use the top level labels provided in the dataset.

CoraIDA: The CoraIDA dataset is constructed as in the CoraL1. However, we only train and test on the papers under Artificial Intelligence, Databases, and Information Retrieval.

DBLP: The DBLP dataset was extracted from the DBLP dump[3]. This dump is composed of the full DBLP data at the time of download. We extracted papers published in preselected conferences and journals with a focus on predefined topics. Thus, if a paper X is published in one of the database focused conference or journal, paper X is assigned the label "database". We constructed a citation network by selecting the neighbors (1 hop away) of each paper. For each of the resulting paper, we extract the title (and abstract when available). This dataset has no edge attribute since the DBLP has no full-text content information.

Delve: The delve dataset is extracted from the delve website[4]. Just as in the DBLP data, we extracted papers published in preselected conferences/journals targeting some predefined topics. The citation graph and paper labeling were constructed in the same ways as in DBLP.

[2]https://people.cs.umass.edu/~mccallum/data.html
[3]https://dblp.uni-trier.de/xml/
[4]http://adatahub.com

Table 2: Accuracy results on the citation datasets. The percentage values signify the amount of training data used.

	CORAL1					CORAIDA				
	10%	20%	30%	40%	50%	10%	20%	30%	40%	50%
	Transductive					Transductive				
Random	21.1	21.1	20.6	20.5	20.8	14.8	14.3	15.2	14.8	14.7
Node2Vec	72.9	75.0	75.6	75.8	75.8	67.8	69.6	71.1	71.8	72.7
Deepwalk	71.8	74.2	75.2	75.3	75.3	66.5	69.8	71.0	72.1	72.0
TADW	71.3	73.8	74.7	75.5	75.9	68.4	70.7	71.9	72.8	74.1
Planetoid-T	48.0	54.3	55.7	63.0	63.4	43.1	-	-	-	-
GCN_MLP	73.2	76.2	77.3	77.7	78.3	69.4	72.4	74.3	74.9	75.1
GCN	**80.3**	**82.4**	**83.3**	**83.8**	**84.2**	**76.4**	**78.1**	**79.6**	**80.2**	**80.7**
GCN_cheb	**80.4**	**81.9**	**82.9**	**83.6**	**84.2**	**76.1**	**78.2**	**79.6**	**80.0**	**80.7**
GAT	75.5	75.9	76.6	76.1	76.4	68.4	69.3	70.6	70.8	70.6
RAW-T	**80.1**	**81.8**	**82.4**	**83.7**	**84.4**	**76.1**	**78.0**	**79.7**	**79.9**	**80.6**
	Inductive					Inductive				
Feature	70.9	72.5	73.8	74.0	74.8	66.5	69.5	71.7	71.6	72.1
GraphSAGE-mean	73.4	77.0	77.3	78.7	79.6	68.1	71.7	78.0	74.6	74.9
GraphSAGE-GCN	73.9	77.3	77.6	78.5	79.2	65.1	66.1	66.8	74.3	68.3
GraphSAGE-maxpool	71.0	76.1	77.4	78.5	79.7	73.8	68.9	75.9	79.5	76.7
GraphSAGE-meanpool	71.4	76.3	76.6	78.3	79.5	64.9	70.1	74.7	79.3	80.3
GraphSAGE-LSTM	71.0	75.7	76.1	77.5	78.9	64.9	77.0	78.1	79.1	75.0
Planetoid-I	61.9	71.0	71.8	71.5	73.5	57.1	62.9	65.5	67.1	67.8
FastGCN-importance	76.3	78.5	79.8	80.9	81.6	5.3	74.5	76.8	77.6	78.4
FastGCN-uniform	75.7	78.1	79.1	80.2	81.3	6.1	74.0	76.4	77.4	78.0
RAW-I	**80.1**	**81.8**	**82.4**	**83.6**	**84.3**	**76.1**	**78.0**	**79.2**	**79.9**	**80.4**

	DBLP					DELVE				
	10%	20%	30%	40%	50%	10%	20%	30%	40%	50%
	Transductive					Transductive				
Random	20.6	20.6	20.5	20.6	20.6	24.8	24.8	24.8	24.8	24.7
Node2Vec	78.4	78.6	78.6	78.6	78.6	58.7	58.8	58.8	58.8	58.8
Deepwalk	78.4	78.4	78.6	78.6	78.5	58.6	58.8	58.8	58.8	58.8
RAW-T	**80.9**	**81.6**	**81.7**	**82.0**	**82.1**	**81.6**	**82.9**	**83.5**	**83.6**	**84.2**
	Inductive					Inductive				
Feature	73.6	73.9	74.1	74.1	74.2	80.8	81.0	81	81.1	81.1
GraphSAGE-mean	72.6	73.1	73.6	74.6	74.9	74.4	76.8	78	79.1	79.8
GraphSAGE-GCN	76.7	77.5	77.8	78.3	78.2	65.1	66.1	66.8	67.7	68.3
GraphSAGE-maxpool	73.8	74.9	75.9	76.6	76.7	74.8	77.8	78.8	79.5	80.4
GraphSAGE-meanpool	73.2	74.3	74.7	75.5	75.8	75.2	77.4	78.4	79.3	80.3
GraphSAGE-LSTM	72.2	73.9	74.8	75.6	75.0	74.5	77.0	78.1	79.1	80.2
Planetoid-I	73.8	74.6	74.5	74.6	74.9	78.8	78.7	79.5	79.5	79.6
FastGCN-importance	76.4	77.9	78.8	79.2	79.3	75.5	75.7	74.9	74.3	73.9
FastGCN-uniform	76.4	77.7	78.3	78.3	78.5	75.1	75.1	74.1	73.1	72.7
RAW-I	**80.9**	**81.5**	**81.6**	**81.9**	**82.1**	**81.5**	**82.8**	**83.5**	**83.6**	**84.0**

4.2 Experimental setup

The experiments were conducted on a Linux system using Python. Our method is implemented using the Tensorflow library. Each GPU based experiment was conducted on an Nvidia 1080TI GPU. When the abstract is available, a paper (node) attribute is given as a concatenation of both the title and abstract else only the title is used. Each citation relationship (edge) attribute is given as the concatenation of all its citation contexts (i.e., sentences where the reference is mentioned in the citing paper). The paper and citation attributes are then converted to a vector by applying the latent semantic analysis (LSI) method on the document-term matrix features, resulting in 300-dimension features vectors. We complete the missing citation attributes with zero vectors and assume no missing paper attribute. In all the experiments, the attribute vector is normalized to unit norm.

For our proposed model, we performed a grid search over the length of walk $T = \{5, 10, 20, 40\}$ and the number of walks per node $M = \{1, 5, 10, 20\}$. For each neural network based model, we performed a grid search over the learning rate $lr = \{1e^{-2}, 5e^{-2}, 1e^{-3}, 5e^{-4}, 1e^{-4}\}$ and hidden layer dimension $d = \{32, 64, 128\}$. We performed the parameter grid search by training on the CoraL1 dataset with 10% labeled samples. The best parameters per model from the grid search are then used in all experiments. The RAW models are trained for 30 epochs with a parameter set ($d = 128, T = 10$, $lr = 4e^{-4}$, and $M = 10$). The GCN and FastGCN models are trained for 200 epochs with $lr = 1e^{-2}$ and $d = 64$ and 128 respectively. The GraphSAGE and GAT models are trained for 20 and 100 epochs respectively with a parameter set of ($d = 128, lr = 1e^{-2}$). The Planetoid models are trained for 5000 epochs with a parameter set of ($d = 64, lr = 0.1$). We used the Scikit-Learn implementation of Linear SVM with default settings for embedding based evaluations. All experiment results reported in this paper are averaged from running on each dataset five times on random samples. For each experiment, we separate 30% of the labeled data for testing. We then vary the number of labeled training data, with the remaining labeled samples assumed to be unlabeled (included in the set of unlabeled samples). In all our experiments, we assume the graph to be undirected.

4.3 Comparison Methods

To evaluate the performance of our model, we compare RAW with several state-of-the-art semi-supervised graph-based methods using classification accuracy as the performance metric. We selected the most competitive baselines that are also publicly available online to avoid unfair evaluations due to faulty implementation. The baseline methods are from different groups:

Unsupervised embedding + classifier: we generate embeddings using several unsupervised embedding methods, which we then give as input to the Linear SVM model for training and classification. The embedding methods include: Node2Vec [12], DeepWalk [25], Latent Semantic Analysis [9], and TADW [35].

Semi-supervised learning on graph: we selected the most popular models including Planetoid [36], and GCN [17].

Supervised learning on graph: in the inductive setting, we evaluate against several variants of FastGCN [6] and GraphSAGE [13] which are supervised learning models for inductive node classification. Note, however, that our proposed method works in a semi-supervised manner in both the inductive and transductive settings.

Semi-supervised learning on graph with attention: like our RAW, GAT [31] and AGNN [30] employed attention mechanism when aggregating the neighbors. Note that we only show the results of GAT due to the poor performance of AGNN on our datasets.

4.4 Results

4.4.1 ***Transductive.*** Table 3 shows the classification performance of our proposed model and other state-of-the-art models. Our proposed model exhibited similar performance compared with GCN in the transductive settings, but it outperforms all other baseline methods in all settings. For the GAT models, we use the sparse version (SpGAT) as the original implementation gave an out of memory error (OOM) on even the CoraL1 with 10% labeled samples. We could only evaluate the GCN and GAT on the Cora datasets as we got out of memory error when applying them to the other large datasets due to the dense LSI vectors. We will evaluate the scalability of these methods and show the memory usage analysis in section 4.4.4. In summary, RAW is usable on large-scale graphs and produce the best node classification results, with no significant difference to GCN, but more efficient than GCN.

4.4.2 ***Inductive.*** Table 3 also shows the comparison of RAW and other inductive models. RAW outperforms all the baseline methods in all settings. In the inductive setting, the testing nodes are removed from the training graph and thus are not seen during training. The agent learns the optimal policy for the graph walk during training that will be generalized to unseen nodes. The test nodes are only added to the graph during testing. The agent (guided by the policy learned after training), starts a walk from the added nodes to learn the embedding for the new nodes. We compare RAW against GraphSAGE, FastGCN and Planetoid inductive model. GraphSAGE and FastGCN are supervised learning algorithms and thus do not use the unlabeled and test nodes during training. The superior performance of RAW shows that walks starting from the new nodes guided by the learned policy aggregated the most useful information for classifying the starting node (the target to classify).

4.4.3 ***Trajectory Analysis.*** Furthermore, we analyze the returned walk trajectory from RAW. This study is performed on the CoraIDA dataset with $T = 30$. We extract the trajectories learned for nodes in each class, and then get the distribution of labels for all nodes visited on these trajectories. The 23 columns in the whole heatmap plot correspond to 23 class labels given in Table 3. From figure 2, we can see that the walk sequences for each class mostly visit the nodes in the same class as the target class (the light squares on diagonal). This verifies that RAW agent tends to walk to nodes in the same class for accomplishing the classification task. It is worth mentioning that RAW agent has no information about label when walking, neither the target label (label of the starting node), nor the label of neighboring nodes.

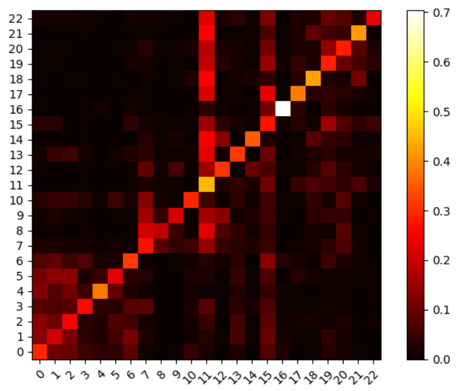

Figure 2: A heatmap whose d-th column demonstrates the RAW agent starting from nodes with label d moved to nodes with what label distribution. The visiting frequency rate is shown in color. Brighter color indicates more visits.

Table 3: Class label IDs of the CoraIDA dataset

Class	ID	Class	ID
DB/Object Oriented	0	AI/Machine Learning	11
DB/Query Evaluation	1	AI/NLP	12
DB/Relational	2	AI/Data Mining	13
DB/Temporal	3	AI/Speech	14
DB/Concurrency	4	AI/Knowledge Representation	15
DB/Performance	5	AI/Theorem Proving	16
DB/Deductive	6	AI/Games and Search	17
IR/Retrieval	7	AI/Vision and Pattern Recognition	18
IR/Filtering	8	AI/Planning	19
IR/Extraction	9	AI/Agents	20
IR/Digital Library	10	AI/Robotics	21
		AI/Expert Systems	22

More importantly, we observe in Figure 2 the relationship between the classes (note again RAW agent moves without any label information). For instance, we can observe that papers under some topics in a research field tend to visit other papers in the same research field more often. *Database* papers (with label 0-6) form a block in the left-bottom corner. The other two blocks, although not obvious but observable, correspond to *information retrial* and *artificial intelligence*. Figure 2 also highlights the important topics. We can notice the influence of the *Machine Learning* class on the *Artificial Intelligence* and *Information Retrieval* community. This influence is shown by the ratio of times the walk sequence of nodes in each class under *AI* visits the *machine learning* nodes. It is interpretable as an individual usually needs to read some machine learning papers/books to understand these topics better. We also notice the versatility of the classes. We see the walk sequence of

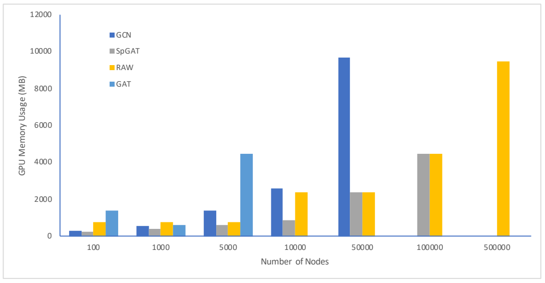

Figure 3: GPU memory usage. Missing bars indicate an out of memory error (OOM).

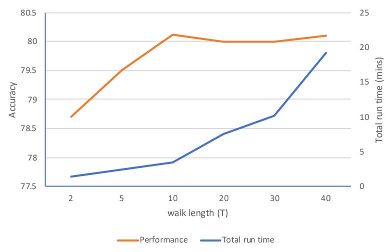

Figure 4: Effect of the walk length on the predictive performance and running time.

the class *Theorem Proving* mostly visit nodes in its class. This result shows that the research area is quite narrow while *Machine Learning* and *Knowledge representation* are broader topics and therefore, more versatile.

To further analyze the performance of RAW, we compare the path made by RAW and random walk. By setting $T = 10$, we obtain a set of trajectories returned by RAW, and another set of trajectories by random walk starting from the same node. For each trajectory, we calculate the *path label diversity* for each walking step t:

$$\delta_t = 1 - \frac{\Sigma_{i=0}^{t} \mathbb{1}(l_i = l_0)}{t}, \tag{4}$$

where l_0 is the label of the starting node, l_i is the label of the i-th node on the path, and $\mathbb{1}()$ is the indicator function. The value δ_t is low (to 0) when nodes on the path have the same label as the starting node, indicating that the agent learned to explore neighboring nodes with the same label as the target label. Note that the agent has no label knowledge during the walk. Figure 5 shows the mean and variance of the *path label diversity* when starting at two different selected nodes. We can see that RAW agent walks with a much lower diversity than random walk.

4.4.4 ***Parameter and Memory Analysis.*** We study the effect of the walk length $T = \{2, 5, 10, 20, 30, 40\}$ on the performance of the model. We train the model on the CoraL1 dataset with 10% training samples. In Figure 4, it can be observed that the model already performs well after ten steps as there is no much improvement with an increase in the number of walks. Meanwhile, more number of walks causes higher time cost.

Figure 3 shows the GPU memory utilization of our proposed model and several semi-supervised state-of-the-art transductive

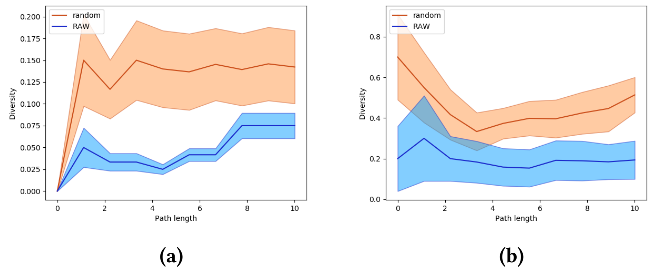

Figure 5: The mean and variance of the *path label diversity* defined in Eq.(4), measured on ten paths starting from two randomly sampled paper, (a) in class *"Concurrency"* and (b) in class *"Vision and pattern recognition"*

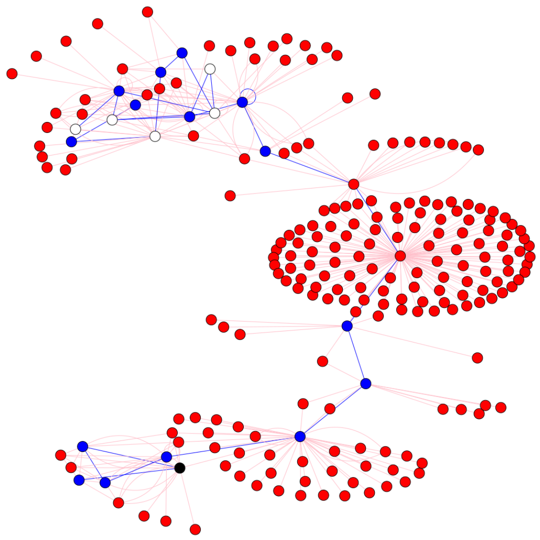

Figure 6: Case study of a sampled walk trajectory, starting from the node (black node) of a paper entitled "*A Critique of Structure from motion Algorithm*" classified as "Vision and Pattern Recognition" on a subgraph of the CoraIDA graph.

models. We randomly generated Erdős-Rényi graphs in size of 100, 1K, 5K, 10K, 50K, 100K and 500K (the number of nodes), and set the number of edges in each graph to be ten times the number of nodes. We randomly generate 300-dimension attributes for the nodes and edges. We then measure the GPU memory consumption using the *nvidia-smi* Linux command on each graph and compare RAW with GCN, GAT and SpGAT. GCN and GAT do not scale with the number of nodes and edges (shown with zero bars in Figure 3 after the graph gets larger than 50K).

5 CASE STUDY

In this section, we train the RAW model on the CoraIDA dataset with a trajectory length of $T = 30$ and present a case study of a sampled walk trajectory of a paper. Figure 6 shows the walk sequence extracted from a paper entitled "*A Critique of Structure from* *motion Algorithm*" classified as "Vision and Pattern Recognition" on a subgraph of the CoraIDA graph. The thickness of the edges signifies the ratio of times the edge was traversed during the walk. The color of the nodes signifies the class relationship of the node to the target class. The blue color signifies that a node has the same label as the target label, the red signifies that a node has a label different to the target node, and the black color signifies the start node. Note that the target class is the class of the start node.

We see from Figure 6 that *the agent can selectively make decisions to visit nodes with the same labels as that of the start node.* The agent also visits unlabeled nodes (white nodes in the right-bottom corner). Interestingly, the visited unlabeled nodes have similar topics as the start paper, e.g., entitled "*new statistical models for randoms-precorrected pet scans*", "*fast monotonic algorithms for transmission tomography*", etc.

6 CONCLUSION

In this paper, we propose to address the semi-supervised node classification problem in attributed networks by letting an agent choose the most relevant nodes in a recurrent walk framework. The decision of where to visit is determined by considering the previous visiting history, the current node content, node content of node one-hop neighbors, and the edge content between the current node and its linked neighbors. The accumulated information from the nodes in the sequence is finally used for classification. We show by several experiments and analysis that the proposed model outperforms several state-of-the-art methods in both transductive and inductive settings. The analysis of the obtained walk sequences also confirms that our model selects the most relevant nodes to visit and thus leads to higher classification accuracy than others.

ACKNOWLEDGMENTS

This work was partially supported and funded by King Abdullah University of Science and Technology (KAUST), under award number FCC/1/1976-19-01, and NSFC No. 61828302.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. 2017. Watch your step: Learning graph embeddings through attention. *arXiv preprint arXiv:1710.09599* (2017).

[2] Uchenna Akujuobi, Ke Sun, and Xiangliang Zhang. 2018. Mining top-k Popular Datasets via a Deep Generative Model. In *IEEE BigData*. IEEE, 584–593.

[3] Uchenna Akujuobi, Han Yufei, Qiannan Zhang, and Xiangliang Zhang. 2019. Collaborative Graph Walk for Semi-supervised Multi-Label Node Classification. In *ICDM*.

[4] Basmah Altaf, Uchenna Akujuobi, Lu Yu, and Xiangliang Zhang. 2019. Dataset Recommendation via Variational Graph Autoencoder. In *ICDM*.

[5] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[6] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

[7] Yujun Chen, Yuanhong Wang, Yutao Zhang, Juhua Pu, and Xiangliang Zhang. 2019. AMENDER: an Attentive and Aggregate Multi-layered Network for Dataset Recommendation. In *ICDM*.

[8] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[9] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41, 6 (1990), 391–407.

[10] Issam Falih, Nistor Grozavu, Rushed Kanawati, and Younès Bennani. 2018. Community detection in attributed network. In *Companion Proceedings of the The Web Conference*. 1299–1306.

[11] Hongchang Gao and Heng Huang. 2018. Deep Attributed Network Embedding.. In *IJCAI*. 3364–3370.

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. ACM, 855–864.

[13] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.

[14] Yedid Hoshen. 2017. Vain: Attentional multi-agent predictive modeling. In *NIPS*. 2701–2711.

[15] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label informed attributed network embedding. In *WSDM*. ACM, 731–739.

[16] Jiechuan Jiang, Chen Dun, and Zongqing Lu. 2018. Graph Convolutional Reinforcement Learning for Multi-Agent Cooperation. *arXiv preprint arXiv:1810.09202* (2018).

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

[18] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *SIGKDD*. ACM, 1666–1674.

[19] John Boaz Lee, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh. 2018. Attention models in graphs: A survey. *arXiv preprint arXiv:1807.07984* (2018).

[20] John Boaz Lee, Ryan A Rossi, Xiangnan Kong, Sungchul Kim, Eunyee Koh, and Anup Rao. 2018. Higher-order Graph Convolutional Networks. *arXiv preprint arXiv:1809.07697* (2018).

[21] Jure Leskovec, A Krevl, and SNAP Datasets. 2014. Stanford large network dataset collection, 2014. *URL: http://snap. stanford. edu/data/index. html* (2014).

[22] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. 2018. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering* 30, 12 (2018), 2257–2270.

[23] Zaiqiao Meng, Shangsong Liang, Hongyan Bao, and Xiangliang Zhang. 2019. Co-Embedding Attributed Networks. In *WSDM*.

[24] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. 2014. Focused clustering and outlier detection in large attributed graphs. In *SIGKDD*. ACM, 1346–1355.

[25] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. ACM, 701–710.

[26] Ryan A Rossi, Nesreen K Ahmed, and Eunyee Koh. 2018. Higher-order network representation learning. In *Companion Proceedings of the The Web Conference 2018*. 3–4.

[27] Ryan A Rossi, Rong Zhou, and Nesreen K Ahmed. 2018. Estimation of Graphlet Counts in Massive Networks. *IEEE Transactions on Neural Networks and Learning Systems* (2018).

[28] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.

[29] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *SIGKDD*. ACM, 990–998.

[30] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. 2018. Attention-based Graph Neural Network for Semi-supervised Learning. *arXiv preprint arXiv:1803.03735* (2018).

[31] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* 1, 2 (2017).

[32] Jianlong Wang, Sridhar Rao, Jianlin Chu, Xiaohua Shen, Dana N Levasseur, Thorold W Theunissen, and Stuart H Orkin. 2006. A protein interaction network for pluripotency of embryonic stem cells. *nature* 444, 7117 (2006), 364.

[33] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[34] Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690* (2017).

[35] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *IJCAI*.

[36] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861* (2016).

[37] Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *NIPS*. 321–328.

[38] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *arXiv preprint arXiv:1812.08434* (2018).

[39] Xiaojin Zhu and Zoubin Ghahramani. 2002. Learning from labeled and unlabeled data with label propagation. (2002).

[40] Chenyi Zhuang and Qiang Ma. 2018. Dual graph convolutional networks for graph-based semi-supervised classification. In *WWW*. 499–508.