Accepted Version of the paper published on Information Processing & Management
DOI: https://doi.org/10.1016/j.ipm.2020.102443

Rank-based Self-Training for Graph Convolutional Networks

Daniel Carlos Guimarães Pedronette

Department of Statistics, Applied Mathematics and Computing (DEMAC), São Paulo State University (UNESP), Rio Claro, Brazil

Longin Jan Latecki

Department of Computer and Information Sciences, Temple University, Philadelphia, USA

Abstract

Graph Convolutional Networks (GCNs) have been established as a fundamental approach for representation learning on graphs, based on convolution operations on non-Euclidean domain, defined by graph-structured data. GCNs and variants have achieved state-of-the-art results on classification tasks, especially in semi-supervised learning scenarios. A central challenge in semi-supervised classification consists in how to exploit the maximum of useful information encoded in the unlabeled data. In this paper, we address this issue through a novel self-training approach for improving the accuracy of GCNs on semi-supervised classification tasks. A margin score is used through a rank-based model to identify the most confident sample predictions. Such predictions are exploited as an expanded labeled set in a second-stage training step. Our model is suitable for different GCN models. Moreover, we also propose a rank aggregation of labeled sets obtained by different GCN models. The experimental evaluation considers four GCN variations and traditional benchmarks extensively used in the literature. Significant accuracy gains were achieved for all evaluated models, reaching results comparable or superior to the state-of-the-art. The best results were achieved for rank aggregation self-training on combinations of the four GCN models.

Keywords: graph convolutional networks, self-training, rank model, semi-supervised learning

1. Introduction

Mainly grounded by deep-learning approaches, classification methods experimented a remarkable development in last decade. However, in spite of the huge advances achieved, performing classification tasks based on scarce labeled data still remains a challenging task, since deep models often require large amount of data for training. In this scenario, semi-supervised classification re-emerge as a promising approach, capable of also exploiting useful information encoded in the unlabeled data. Actually, semi-supervised learning (SSL) is halfway between supervised and unsupervised, being suitable to operate with large amounts of unlabeled data and a small quantity of labeled data [1, 2]. A direct and central motivation for semi-supervised learning relies on the fact that labeled data is typically much harder to obtain compared to unlabeled data [3].

One of the earliest approaches of semi-supervised learning is self-training, also known as self-labeling or self-supervision [1, 4]. The main idea consists in a wrapper approach that repeatedly exploits a supervised learning method. Firstly, the supervised method is trained based on labeled data only. Subsequently, the unlabeled data is labeled according to the trained supervised model. Then, a selected sub-set of predictions is exploited as additional labeled data to re-train the supervised method [1]. Naturally, the possible combinations between supervised models and approaches to select the additional labeled data open a broad and promising range of self-training approaches [2].

On the other hand, graphs have been employed as a representation tool in a wide range of real-world scenarios, mainly due to their expressive power [5]. Graph analytics approaches enable a better understanding of what is behind

the data, being useful in diverse applications and domains. Tasks as node classification, node recommendation, and link prediction often can benefit from graph-based representations in several areas, from social networks to physical and biological systems [6]. Additionally, the semi-supervised learning literature also exploits graphs as a way to encode the geometry of both labeled and unlabeled data in order to improve supervised methods [1].

For graph-based semi-supervised learning, a remarkable recent development has been achieved by Graph Neural Networks (GNNs) [7] and, more specifically, by Graph Convolutional Networks (GCNs) [8]. The GCN models integrate local node features and graph topology in the convolutional layers [9]. In fact, graph convolution enables the extension of standard convolution from Euclidean to non-Euclidean domain, defined by graph-structured data [5]. Recently, GCNs [8] and subsequent variants [10–14] have achieved state-of-the-art results in diverse applications, specially involving semi-supervised classification tasks. Many variants propose changes in the network structure, mostly by including or removing components. For instance, in [12], the weight matrices between consecutive layers are collapsed and non-linearities removed. In [13], a convolutional layer based on auto-regressive moving average filter is inserted.

In a distinct and promising research direction, other recent works [5, 9, 15, 16] have exploited traditional machine learning strategies for further improving GCN capacities. Mainly motivated by the recent impressive GCN results, some relevant techniques in machine learning have been revisited in conjunction with GCN models [9]. In [15], hypergraphs are exploited to learn deep embeddings on the high-order graph-structured data. A diffusion process is employed in [16] for aggregating information from a larger neighborhood. The neighborhood is constructed based on the sparsification of a generalized form of graph diffusion.

In this context, some few self-training approaches have been investigated

for GCNs very recently [5, 9, 17]. The GCN model is described as a special form of Laplacian smoothing in [9], discussing scenarios of success and fails of the model. While the smoothing is pointed as the key reason for successful scenarios, potential over-smoothing can occur for many convolutional layers. A self-training approach is proposed to overcome such limitations. In a more recent work [5], the method is extended to multiple self-training stages. An aligning mechanism is used on the output of a deep-clustering approach. The clusters are used to assign pseudo-labels for each unlabeled data point in the embedding space.

In this paper, we propose a novel Rank-Based Self-Training approach for Graph Convolutional Networks. Our work focuses on the selection of data points in the embedding space to be used as additional labeled data. How to accurately select effective predictions is a challenging task for self-training approaches, few exploited in recent related work [5, 9]. Typically, the selection is performed by considering the top softmax scores per class [5, 9]. We propose to employ a margin-based selection score inspired by active learning approaches [18]. The nodes are represented according to the score in a rank-based model for the whole dataset, allowing to handle unbalanced additional labeled data per class. Based on the rank model, an expanded labeled set is defined for a second-stage training and classification. The main contributions, differences and novelties with respect to related works can be summarized as follows:

- The proposed self-training approach is not restricted to a specific GCN model. Different from recent related self-training methods [5, 9, 17], which are focused on a GCN model, the proposed method was validated on four GCN models [8, 11–13] with a log-soft-max as the last layer. Actually, the rank-based formulation is versatile and can be easily extended to other learning tasks;

- The proposed margin score addresses the challenging task of self-labeled data selection, which has a central problem in self-training methods. Most of approaches consider the *softmax* score of the predicted class to estimate the confidence of prediction. In contrast, the proposed margin confidence considers not only the class predicted, but the relationship between the first and second higher scores. The motivation is based on the conjecture that even a high *softmax* score can provide a low accurate estimation when first and second top prediction scores are similar. The proposed score provides a simple, yet effective label prediction estimation;

- Our approach employ a global ranking of nodes, being able to handle data with unbalanced classes. In contrast, other self-training methods [5, 9, 17] often select a fixed number of nodes per class as the expanded labeled set. However, a forced equal amount per class can add low-accurate predictions to certain classes;

- A rank aggregation approach is proposed to exploit and fuse the information from distinct GCN models and training executions. Based on the fused confidence which consider different models, an aggregated and more effective ranking of nodes is computed in order to obtain a more accurate expanded labeled set. To the best of our knowledge, it is the first approach which fuses information from distinct GCN models in a self-training setting.

The proposed approach was evaluated for semi-supervised classification tasks on citation datasets broadly used as benchmark in the literature [5, 8, 9, 11–13, 15]. The experiments indicate significant accuracy gains of the proposed self-training method applied to four different GCN models. Accuracy results on semi-supervised classification outperform most of state-of-art methods.

The remainder of this paper is organized as follows. Section 2 discusses related work and a formal definition of the problem setting. Section 3 presents the proposed self-training approach. Section 4 describes the conducted experimental evaluation and, finally, Section 5 discusses the conclusions.

2. Problem Definition and Preliminaries

2.1. Graph-based Semi-Supervised Learning

In this section, we first discuss a formal definition of the semi-supervised learning classification task using graph convolution networks, mostly following [8, 9].

Let $\mathcal{G}$ denotes an undirected graph represented by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V}$ is the node set, $\mathcal{E}$ is the edge set and $\mathbf{X}$ is a feature matrix. The node set is given by $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ and the edge set is defined by a set of pairs $(v_i, v_j) \in \mathcal{E}$, which can be represented by a non-negative adjacency matrix $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{n \times n}$. The feature matrix is defined as $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T \in \mathbb{R}^{n \times d}$, where $\mathbf{x}_i$ is a d-dimensional feature vector which represents the node v_i.

Let $\mathcal{Y} = \{y_1, y_2, \ldots, y_c\}$ be a set of labels which can be assigned to nodes $v_i \in \mathcal{V}$. In this way, the node set can be more specifically defined as $\mathcal{V} = \{v_1, v_2, \ldots, v_L, v_{L+1}, \ldots, v_n\}$, which denotes a partially labeled data set, where $\mathcal{V}_L = \{v_i\}_{i=1}^{L}$ is the labeled data items subset and $\mathcal{V}_U = \{x_i\}_{i=L+1}^{n}$ is the unlabeled data items subset. For semi-supervised classification, as a general rule, we have $|\mathcal{V}_L| \ll |\mathcal{V}_U|$. Formally, the training set can be seen as a labeling function $l : \mathcal{V}_L \rightarrow \mathcal{Y}$, where $y_i = l(v_i) \forall v_i \in \mathcal{V}_L$. The goal is to learn a function $\hat{l} : \mathcal{V}_U \rightarrow \mathcal{Y}$ to predict the labels of unlabeled nodes in $\mathcal{V}_U$.

2.2. Graph Convolutional Networks

Recently, much effort has been made on exploiting deep learning approaches for graph data [6]. In this context, Graph Convolutional Networks

(GCN) represent a relevant graph-based neural network model, introduced in [8]. In a simplified way, GCN learns the embedding (representation) of each node by iteratively aggregating the embeddings of its neighbors, encoding the graph structure directly on a neural network model. A two-layer GCN model is used for semi-supervised node classification in [8], taking into account a graph represented by a symmetric adjacency matrix $\mathbf{A}$.

The network model can be depicted as a function both on the feature data $\mathbf{X}$ and on the adjacency matrix $\mathbf{A}$, as:

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}), \tag{1}$$

where $\mathbf{Z}$ denotes an embedding matrix, such that $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n]^T \in \mathbb{R}^{n \times c}$ and $\mathbf{z}_i$ is a c-dimensional embedded representation learned for the node v_i.

The degree matrices are computed as a pre-processing step, defined as $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$, where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$. Then, the function $f(\cdot)$ which represents the two-layer GCN model assumes the form:

$$\mathbf{Z} = log(softmax(\hat{\mathbf{A}}ReLU(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)})\mathbf{W}^{(1)})) \tag{2}$$

The matrix $\mathbf{W}^{(0)} \in \mathbb{R}^{d \times H}$ defines the neural network weights for an input-to-hidden layer with H feature maps, while $\mathbf{W}^{(1)} \in \mathbb{R}^{H \times c}$ is a hidden-to-output matrix. Both matrices $\mathbf{W}^{(0)}$ and $\mathbf{W}^{(1)}$ are trained using gradient descent, considering the cross-entropy error over all labeled nodes $v_l \in \mathcal{V}_L$. The softmax activation function is applied row-wise and yields the probability distribution over the c class labels for each row, i.e., the probability values sum up to 1 for each row. After log function, the label assigned to a node v_i is defined according to the class with the less negative value in the embedded representation $\mathbf{z}_i$.

Mostly grounded by the success of the GCN [8], various related graph convolutional network models have been recently proposed [9–13, 15, 16].

While some approaches focus in the structure of network models [11–13, 15], others present contributions involving training steps and manifold information [9, 16]. Different network models [8, 11–13] can be used in conjunction with the proposed self-training approach. The only condition is that a log-soft-max operation is kept as the last layer.

3. Rank-based Self-Training

We propose a self-training approach focused on better exploiting the unlabeled data by taking into account the information encoded in the embedding computed by a first stage semi-supervised classification. The method post-processes the predicted labels computed through a GCN classification, by identifying high-confidence predictions to be used for pseudo-labeled data expansion.

A challenging task for different self-training methods [5, 9, 17] is how to identify high-accurate label predictions. This task is a central problem of self-training methods. Most approaches consider the soft-max score of the predicted class to estimate the confidence of prediction, which is often not sufficient. In this regard, we present a margin prediction confidence, inspired by active learning approaches [18, 19], which consider the gap between the first and second higher scores. While active learning methods aims at identifying the most informative samples, we are interested in the most confident labels. Our score is based on the conjecture that the log-soft-max output tends to provide similar values to both first and second classes when the classification is not correct.

Different from other self-training methods, which keep a confidence score update at each epoch [17] or multi-stage [5], our method employs a simple two-stage approach, i.e., we train the GCN twice, first with original label set, and then with the augmented label set. A direct advantage is improved

efficiency and simplicity. A two-state training approach requires the computation of the margin score and the global ranking of vertices. Considering a two-stage approach, such steps are computed only once, reducing substantially the computational efforts required. Hence a two-stage approach keeps a favorable trade-off between effectiveness and efficiency.

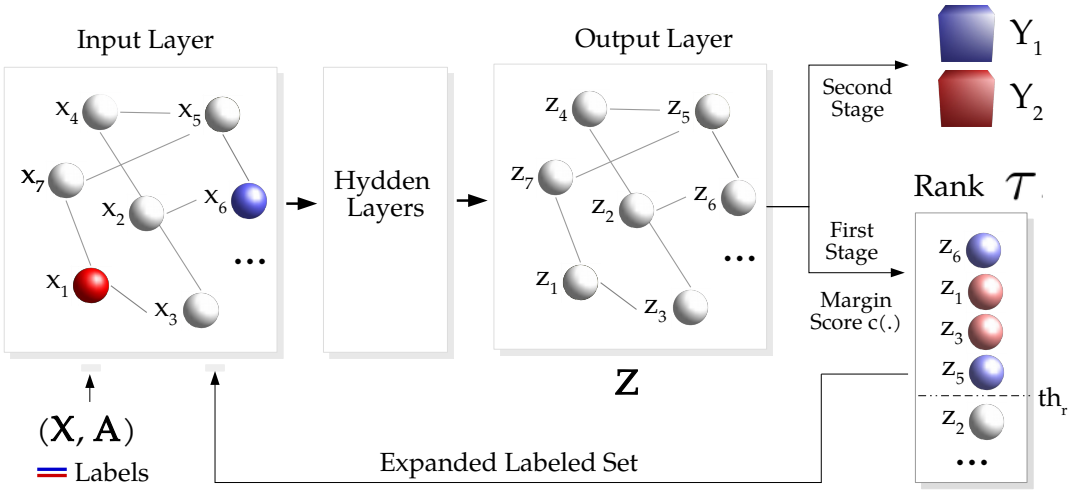


Figure 1: Overview of the proposed Rank-based Self-Training for Graph Convolutional Networks.

The proposed approach is illustrated in Figure 1. Firstly, a first-stage classification is performed by a GCN model. Subsequently, the proposed approach can be divided into three main steps:

1. *Margin Confidence Score:* a margin-based score is computed for each graph node aiming to estimate the confidence on the computed embedding;

2. *Labeled Set Expansion:* the nodes are represented through a rank model, defined according to the margin score. Subsequently, some nodes are selected for an expanded labeled set;

3. *Second Stage Semi-Supervised Classification:* the GCN model is re-trained by taking into account the expanded labeled set.

The three main steps of the proposed method are detailed and formally defined in next sub-sections. A general outline of the proposed method is presented in Algorithm 1. Line 1 performs the firs-stage classification. Lines 2-4 define the computation of the margin confidence, discussed in Section 3.1. The labeled set expansion step is defined in lines 5-12 and detailed in Section 3.2. Line 13 performs the second-stage classification, discussed in Section 3.3.

Algorithm 1 Rank-based Self-Training for GCNs

Require: Feature matrix $\mathbf{X}$, adjacency matrix $\mathbf{A}$, labeled data $\mathcal{V}_L$, α
Ensure: Embedding matrix $\mathbf{Z}$, learned function $\hat{l}^{(2)}$

1: $\mathbf{Z} = f_{GCN}(\mathbf{X}, \mathbf{A}, \mathcal{V}_L)$ # first-stage classification: learned function $\hat{l}^{(1)}$
2: **for all** $v_i \in \mathcal{V}$ **do**
3: $c_i = c(v_i, \mathbf{Z})$ # margin score computation
4: **end for**
5: $\tau = sort(\mathcal{V}, \mathbf{c})$ # ranked list computation
6: $th_r = |\mathcal{V}| \times \alpha$ # threshold definition
7: $\mathcal{V}_E = \mathcal{V}_L$ # labeled set expansion
8: **for all** $v_e \in \mathcal{V}$ **do**
9: **if** $\tau(v_e) \leq th_r$ **then**
10: $\mathcal{V}_E = \mathcal{V}_E \cup v_e$
11: **end if**
12: **end for**
13: $\mathbf{Z} = f_{GCN}(\mathbf{X}, \mathbf{A}, \mathcal{V}_E)$ # second-stage classification: learned function $\hat{l}^{(2)}$

3.1. Margin Confidence Score

The margin confidence score is defined based on the learned embeddings. Given a c-dimensional vector which defines the embedded representation $\mathbf{z}_i$

for a node v_i, a set $\mathcal{S}_i$ is computed containing the $\mathbf{z}_i$ absolute values. The set $\mathcal{S}_i$ is formally defined as:

$$\mathcal{S}_i = \{|\mathbf{Z}_{ij}| : j \in \{0, 1, \dots c\}\} \tag{3}$$

Once a log function is applied to the soft-max layer and the set $\mathcal{S}_i$ contains its absolute values, we are interested in the smallest values in $\mathcal{S}_i$. Such values are associated with the most likely classes. Therefore, we consider a function $m(\mathcal{S}_i, k)$ that returns the k-th smallest element in $\mathcal{S}_i$. Formally, the function $m : \mathcal{S}_i \times \mathbb{N} \to \mathbb{R}$ can be defined as:

$$m(\mathcal{S}_i, k) = \begin{cases} min(\mathcal{S}_i), & k = 1 \\ min(\{v : v \in \mathcal{S}_i, v > m(\mathcal{S}_i, k-1)\}), & k > 1. \end{cases} \tag{4}$$

More specifically, the value of interest is given by the normalized difference between the smallest and the second smallest values. Therefore, the confidence estimation $c(v_i)$ of the predicted class for a given node v_i is defined as:

$$c(v_i, \mathbf{Z}) = \frac{m(\mathcal{S}_i, 2) - m(\mathcal{S}_i, 1)}{\sum_{v \in \mathcal{S}_i} v}. \tag{5}$$

3.2. Labeled Set Expansion

Once a confidence estimation is defined, the nodes are ranked according to the function $c(\cdot)$ in order to obtain a ranked list τ. The ranked list τ can be formally defined as a permutation $(v_1, v_2, \dots, v_n)$ of the node set $\mathcal{V}$. A permutation τ is a bijection from the set $\mathcal{V}$ onto the set $[N] = \{1, 2, \dots, n\}$.

For a permutation τ, we interpret $\tau(v_i)$ as the position (or rank) of node v_i in the ranked list τ. If v_i is ranked before v_j in the ranked list τ, i.e., $\tau(v_i) < \tau(v_j)$, then $c(v_i, \mathbf{Z}) \geq c(v_j \mathbf{Z})$. The rank $\tau(v_i)$ is used to decide if the node v_i is included in the expanded training set.

Let $\alpha \in [0, 1]$ denote a hyper-parameter that regulates the extend of labeled set expansion. Based on α, a ranking threshold th_r is defined as:

$$th_r = |\mathcal{V}| \times \alpha \tag{6}$$

The threshold th_r defines a position in the ranked list τ, until which the nodes are included in the expanded training set. Let $\mathcal{V}_E$ denotes the expanded labeled set, it is formally defined as:

$$\mathcal{V}_E = \{v_e \in \mathcal{V} | \tau(v_e) \leq th_r\} \cup \mathcal{V}_L \tag{7}$$

3.3. Second Stage Semi-Supervised Classification

Once an expanded labeled set $\mathcal{V}_E$ is defined, a second training stage is performed. Let $\hat{l}^{(1)}$ denotes the labeled function learned by the first training stage. The second stage training uses the same network model and same hyper-parameters of the first stage. However, the second stage training considers the set $\mathcal{V}_E$ as the labeled set to learn a new function $\hat{l}^{(2)}$. Finally, the function $\hat{l}^{(2)}$ is used for performing the definitive classification.

3.4. Complexity Analysis

This section presents a brief discussion about the complexity of proposed rank-based self-training approach. The first and second stage classifications are directly associated with the GCN model used. Therefore, the complexity of our method is given by the computation of the margin confidence and the labeled set expansion procedure.

The margin score is computed based on the c-dimensional vector which defines the embedded representation, where c denotes the number of classes. Therefore, the complexity is $O(c)$. The labeled set expansion requires a sorting procedure of the vertices, which has $O(n \log n)$ complexity. Since $c \ll n$, the general complexity of the proposed model is given by $O(n \log n)$.

3.5. Rank Aggregation for Prediction Confidence Fusion

Effectively exploiting the useful information encoded in the unlabeled data is a central issue in semi-supervised learning. More specifically in self-training, the use of unlabeled data is closely associated with the identification of high-accurate predictions.

A true power of the proposed self-training approach becomes unlocked when a few diverse GCN models are exploited. The labeled set expansion is defined through the ranked list of vertices, which is computed based on the margin score. Once the margin score varies according to the GCN model, distinct GCN models yield different expanded labeled sets. Therefore, ranked lists from different GCN models present diversity, which can be exploited and aggregated for a more accurate label expansion step.

Additionally, even the same GCN model can give rise to distinct expanded labeled sets through distinct training executions, due to the stochastic characteristics of optimization procedures and random parameters initialization involved. In this scenario, we propose a rank aggregation approach to exploit such diversity. A fused prediction confidence score is computed by aggregating the ranks in two dimensions: distinct GCN models and different training executions of each model. The fused scores gives rise to a more effective aggregated ranking and, therefore, a more accurate expanded labeled set. Based on the expanded labeled set, a second stage classification is performed.

More formally, each training execution of each GCN model assigns a different prediction score to given pair (node, class). As a result, a different margin-based confidence score is computed for each execution. Let $\mathcal{M}$ denotes a set of GCN models, such that a model $M_j \in \mathcal{M}$ and $|\mathcal{M}| = m_f$. Let t denotes the current training execution, such that $t \in \{1, 2, \ldots, n_f\}$. Let $c_{t,j}(v_i)$ denotes a confidence score computed by a GCN model M_j on the training execution t for a node v_i. A fused confidence score $c_f(v_i)$ is

computed based on a multiplicative rank aggregation formulation [20], as:

$$c_f(v_i) = \prod_{j=1}^{m_f} \prod_{t=1}^{n_f} (1 + c_{t,j}(v_i)) \tag{8}$$

The fused confidence score $c_f(v_i)$ combines information from different GCN models and training executions and can be used in place of function $c(\cdot)$ (Equation 5) to define the ranked list τ. In this way, the aggregated ranking τ is exploited to define a more accurate expanded labeled set for a second-stage classification. The second stage classification can be performed by any model $M_j \in \mathcal{M}$. In order to ensure a consistent aggregation, the fused score $c_f(v_i)$ can only be considered if the the same label is assigned to node v_i by all GCN models on all training executions. Otherwise, we set $c_f(v_i) = 0$.

4. Experimental Evaluation

In this section, we describe the experiments conducted to assess the accuracy of the proposed method in the task of semi-supervised node classification.

4.1. Graph Convolutional Network Models

As previously discussed, the proposed self-training approach allows different GCN models and variants. The margin-based score requirement is restricted only to the last layer, expected to be a log-soft-max operation. We validated the proposed method on four GCN models with a log-soft-max as the last layer, described in the following:

- *GCN*: Graph Convolution Network [8], a seminal GCN model broadly defined as an efficient variant CNNs on graphs (detailed in Section 2.2);

- *SGC*: Simple Graph Convolution [12], a simplification of GCN models obtained by removing nonlinearities and collapsing weight matrices between consecutive layers;

- *APPNP*: Approximate Personalized Propagation of Neural Predictions [11], an algorithm which exploits the relationship between GCNs and PageRank, deriving a propagation strategy based on personalized PageRank;

- *ARMA*: ARMA Filter Convolutions [13], a GCN variant which defines a convolutional layer based on Auto-Regressive Moving Average (ARMA) filters.

All four network models are used in the rank aggregation fusion to boost the correctness of the extended label sets. Diverse combinations of pairs and all models are considered on the experimental evaluation.

4.2. Datasets

The experimental evaluation was conducted on three citation network datasets[1]: Cora [21, 22], Citeseer [22, 23] and Pubmed [24]. Such datasets have been largely used in the literature [10, 12, 15, 25–27] as benchmark for semi-supervised classification tasks, including some representative [8, 24] and recent [15, 26] works. Table 1 presents some statstics of the three datasets, briefly detailed in the following.

The three datasets are composed by textual documents and a list of citation links between them. A graph is defined, where each node represents a document and each edge represents a citation link. Additionally, a sparse bag-of-words feature vector is associated to each document. The Cora dataset contains scientific publications divided into 7 categories. Each publication is described by a binary bag-of-word representation, where 0 (or 1) indicates

[1] https://linqs.soe.ucsc.edu/data

Table 1: Citation network datasets statistics.

Dataset	Nodes	Edges	Classes	Features	Train/Val/Test
Cora	2708	5429	7	1433	140/500/1000
CiteSeer	3327	4732	6	3703	120/500/1000
PubMed	19717	44338	3	500	60/500/1000

the absence (or presence) of the corresponding word from the dictionary. The dictionary consists of 1,433 unique words (features). The Citeseer dataset employs an analogous representation but of 3, 703 dimensions. The Pubmed dataset is divided into 3 classes and uses a vectorial representation using Term Frequency-Inverse Document Frequency (TF-IDF), based on a dictionary of 500 terms. For all datasets, each document has a single class label.

4.3. Experimental Protocol and Implementation Details

The experiments were conducted according to the protocol initially used in [24] and followed by other works [8, 25]. The training is performed using 20 labels per class, but all feature vectors (labeled and unlabeled data). The accuracy prediction accuracy is evaluated on a test set of 1,000 nodes. The dataset splits follows [8, 24, 25], with a validation set of 500 labeled samples used by [8, 25]. The labels of the validation set are not used for training.

The implementation of the proposed Rank-based Self-Training was made upon PyTorch Geometric (PyG) [25], a geometric deep learning extension library for PyTorch [28]. We also used the PyG [25] implementation of the four network models previously discussed: Graph Convolution Network (GCN) [8]; Simple Graph Convolution (SGC) [12]; Approximate Personalized Propagation of Neural Predictions (APPNP) [11]; and ARMA Filter Convolutions (ARMA) [13].

All the models were trained for 200 epochs using using Adam [29] optimization. During the training process, for both first and second stages, the

model is selected according to the lowest validation loss. The hyperparameters and network configurations for each model followed the default values given by the benchmark provided in PyG [25][2]. The learning rate is defined as 0.01 for all networks except SGC, which used 0.1. The dropout parameter is set to 0.5 also for all networks except SGC, which does no employ dropout. The early stop window size was defined to 10 for all networks except ARMA, which used 100. The rank-based Self Training approach has only one hyper-parameter α, which, is discussed in next sub-section.

Regarding feature normalization, the PyG implementation includes a normalization procedure, which impacts positively on most of network models. However, our self-training approach performs better without the normalization. Therefore, we report results before self-training on both situations: with and without feature normalization.

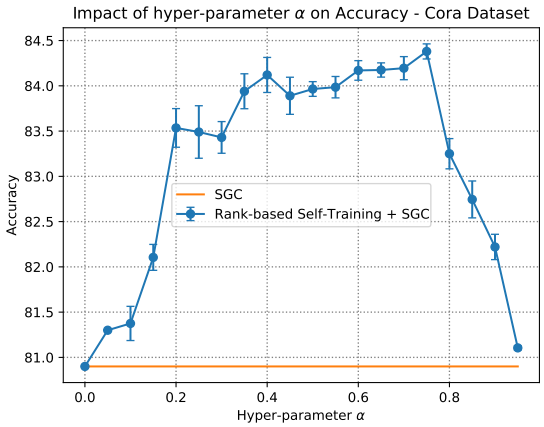
4.4. Parameter Space Analysis

This section presents an analysis of the parameter space and definition of parameter settings. The proposed rank-based Self Training requires only one hyper-parameter α, which defines a rank threshold. The impact of α on accuracy was evaluated on Cora dataset considering an average of 20 executions. Figure 2 presents the results for SGC and GCN networks.

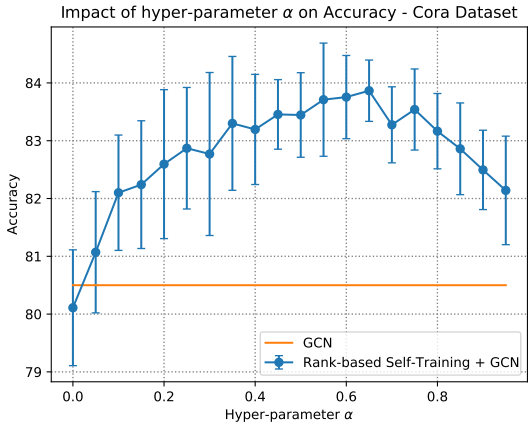
We can observe that very low and high values of α lead to small gains in comparison with the network model in isolation (orange line). However, a large intermediary region tends to produce higher accuracy gains, indicating the robustness of our approach to small parameter variations. In all experiments we used $\alpha = 0.4$, which approximates the beginning of intermediary values with high associated accuracy gains.

The aggregation of network models also considers a parameter n_f, which defines the aggregation of executions on first stage and the average of execu-

[2] `https://github.com/rusty1s/pytorch_geometric/tree/master/benchmark`



(a) SGC



(b) GCN

Figure 2: Evaluation of the impact of hyper-parameter α on accuracy.

tions on second stage classifications. We varied n_f in the interval [5,50] and evaluated the impact on accuracy. Figure 3 shows the results, which are very stable to different settings. The value of $n_f = 20$ was empirically defined and used in all aggregation experiments.

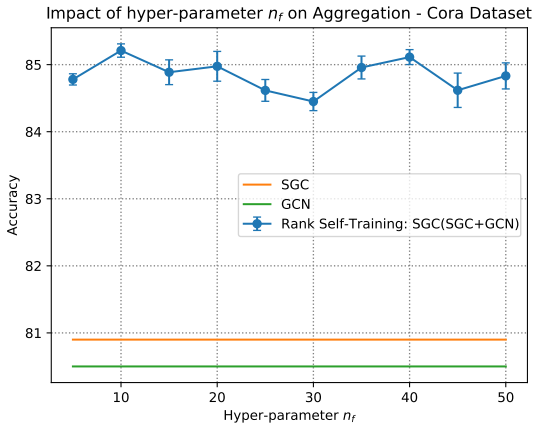


Figure 3: Evaluation of the impact of hyper-parameter n_f on aggregation.

4.5. Results

This section present the accuracy results on semi-supervised classification tasks for the citation datasets. Firstly, we report the accuracy results of the four network models in isolation, considering feature normalization. Table 2 present the results reported according to PyG [25], with higher accuracy in bold.

Table 2: Accuracy of semi-supervised classification on citation datasets as reported in [25] with feature normalization.

Method	**Cora**	**CiteSeer**	**PubMed**
GCN [8]	81.5 ± 0.6	71.1 ± 0.7	79.0 ± 0.6
SGC [12]	81.7 ± 0.1	71.3 ± 0.2	78.9 ± 0.1
ARMA [13]	82.8 ± 0.6	**72.3** ± 1.1	78.8 ± 0.3
APPNP [11]	**83.3** ± 0.5	71.8 ± 0.5	**80.1** ± 0.2

Table 3 present the accuracy results for the proposed Rank-based Self-Training method. We reported the results of each network model in isolation and jointly with the proposed self-training approach. All results consider an

average of 100 executions and do not use feature normalization. The best accuracy results are highlighted in bold. We can observe significant gains obtained for most network models and datasets. All the highest accuracy results (in bold) are achieved by the Rank-based Self-Training, even considering results of Table 2.

Table 3: Accuracy of Rank-based Self-Training on citation datasets. Results without feature normalization.

Method	**Cora**	**CiteSeer**	**PubMed**
GCN [8]	80.5 ± 0.6	66.9 ± 0.7	78.9 ± 0.3
Rank-based Self-Trainig + GCN	83.3 ± 0.9	69.4 ± 1.9	80.3 ± 0.4
SGC [12]	80.9 ± 0.0	69.3 ± 0.0	78.9 ± 0.0
Rank-based Self-Trainig + SGC	84.1 ± 0.2	**73.1** ± 0.2	75.9 ± 0.0
ARMA [13]	80.1 ± 1.0	64.6 ± 2.2	78.2 ± 0.4
Rank-based Self-Trainig + ARMA	82.9 ± 1.0	68.0 ± 4.4	79.5 ± 0.6
APPNP [11]	82.8 ± 0.7	70.1 ± 0.7	79.9 ± 0.2
Rank-based Self-Trainig + APPNP	**84.7** ± 0.6	71.2 ± 0.7	**81.2** ± 0.7

Results for fusion of confidence prediction are reported on Table 4. Different pairs combinations and the aggregation of all network models are considered. SGC and APPNP network models are considered for the second stage classification, once have achieved the best results on individual results reported in Table 3. As it can be observed, the results of fusion reached the high accuracy results for all datasets. The highest results for each segment of the table is highlighted in bold.

4.6. Visual Analysis

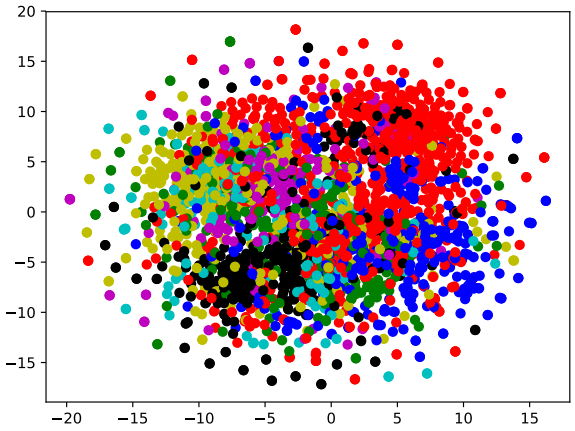
In order to enrich the discussion about the impact of the proposed method, we present a qualitative visualization of the feature space. The visual analysis illustrates the outcome of the method through a 2-D projection of data fea-

Table 4: Accuracy results for confidence prediction fusion based on **Rank Aggregation Self-Training**.

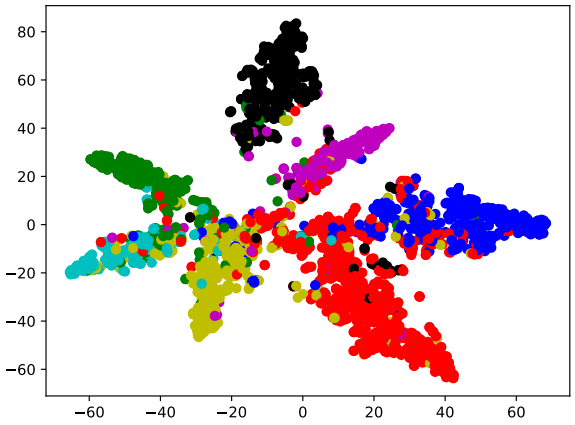
Method	Cora	CiteSeer	PubMed
Rank-based Self-Trainig + GCN	83.3 ± 0.9	69.4 ± 1.9	80.3 ± 0.4
Rank-based Self-Trainig + SGC	84.1 ± 0.2	**73.1** ± 0.2	75.9 ± 0.0
Rank-based Self-Trainig + ARMA	82.9 ± 1.0	68.0 ± 4.4	79.5 ± 0.6
Rank-based Self-Trainig + APPNP	**84.7** ± 0.6	71.2 ± 0.7	**81.2** ± 0.7
Rank Aggregation Self-Training: SGC for second stage classification			
GCN+SGC	84.2 ± 0.2	73.2 ± 0.0	76.9 ± 0.0
GCN+ARMA	83.7 ± 0.0	73.3 ± 0.0	79.1 ± 0.0
GCN+APPNP	84.2 ± 0.0	73.3 ± 0.1	78.9 ± 0.0
SGC+ARMA	84.2 ± 0.2	73.4 ± 0.0	77.8 ± 0.0
SGC+APPNP	**84.4** ± 0.2	72.8 ± 0.1	77.8 ± 0.0
ARMA+APPNP	84.1 ± 0.0	73.5 ± 0.1	**79.0** ± 0.0
GCN+SGC+ARMA+APPNP	84.3 ± 0.0	**74.1** ± 0.0;	78.3 ± 0.0
Rank Aggregation Self-Training: APPNP for second stage classification			
GCN+SGC	84.0 ± 0.4	71.4 ± 0.4	79.3 ± 0.3
GCN+ARMA	84.6 ± 0.3	71.6 ± 0.5	80.3 ± 0.4
GCN+APPNP	84.9 ± 0.5	71.4 ± 0.5	80.8 ± 0.2
SGC+ARMA	84.7 ± 0.4	72.1 ± 0.3	79.3 ± 0.2
SGC+APPNP	**85.1** ± 0.4	**72.5** ± 0.5	79.6 ± 0.2
ARMA+APPNP	85.0 ± 0.3	71.9 ± 0.5	**80.9** ± 0.2
GCN+SGC+ARMA+APPNP	85.0 ± 0.5	72.1 ± 0.5	79.9 ± 0.2

tures and their respective computed embeddings. The analysis is conducted on three datasets with the 2-D projected space computed by the t-SNE [30] algorithm.

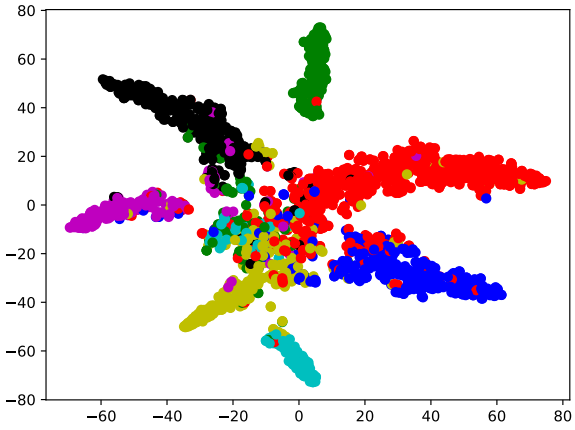
Figure 4 shows three t-SNE visualizations on the Cora dataset: (a) depicting the raw dataset features; (b) the embeddings computed by SGC [12] model; and (c) the embeddings computed by the proposed self-training ap-



(a) Cora Raw features

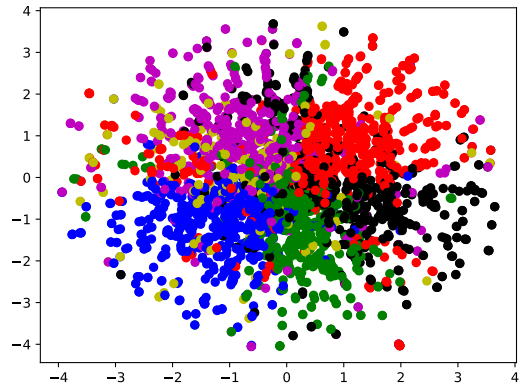


(b) Cora SGC embeddings

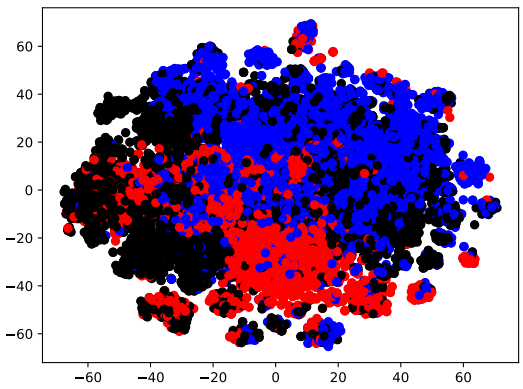


(c) Cora Self-Training SGC embeddings

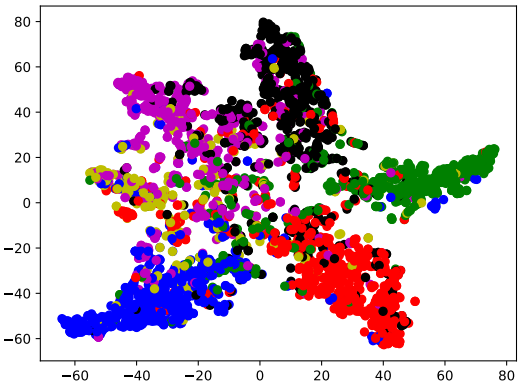
Figure 4: t-SNE [30] visualization of initial features and computed embeddings for the Cora dataset.



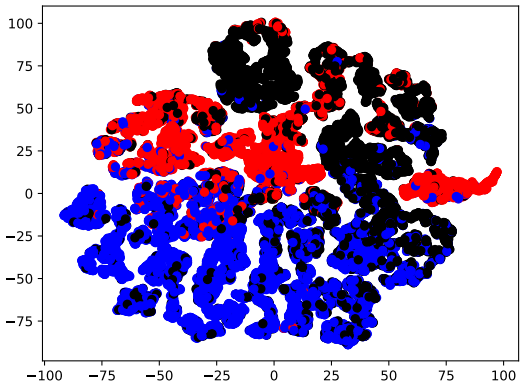
(a) CiteSeer Raw features



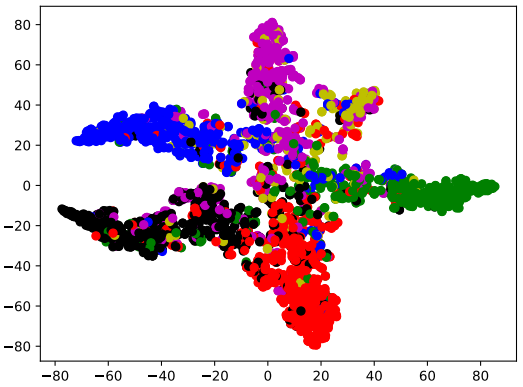
(d) PubMed Raw features



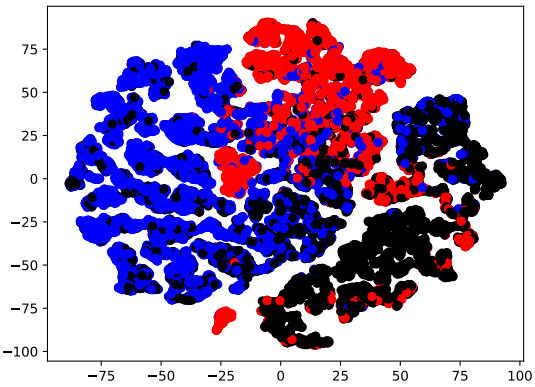
(b) CiteSeer SGC embeddings



(e) PubMed SGC embeddings



(c) CiteSeer Self-Training SGC embeddings



(f) PubMed Self-Training SGC embeddings

Figure 5: t-SNE [30] visualization of initial features and computed embeddings for the CiteSeer and PubMed datasets.

proach based on SGC [12]. Despite of the subjectivity associated to the visualization, we can observe a higher cohesion of data items according to the seven topic classes of Cora and better separation of different classes for the proposed self-training embedding.

Figure 5 illustrates analogous t-SNE visualizations on CiteSeer and PubMed datasets. The impact of self-training is more pronounced on the CiteSeer dataset, where the embeddings representations of each class are thinned. On the PubMed dataset, the difference is slight, but a better separation between blue and black classes can be observed.

4.7. Comparison with State-of-the-art

We compared the proposed method with various state-of-the-art algorithms on semi-supervised classification tasks. The comparison was conducted on Cora, Citesser, and Pubmed datasets. The experimental setting and data splits followed the protocol used in [8, 24], which have been established as a standard benchmark protocol in the area. The classification accuracy of the compared methods is directly quoted from the literature. Several methods were considered, from traditional semi-supervised learning methods [31, 32] to more recent baselines [24, 33]. Representative works, as GCN [8] and GAT [10], were also considered in addition to very recent network models [15, 26]. Related self-training approaches were also included in the comparison, considering the higher accuracy results reported in [9]. Other self-training methods [5, 17] were not included due to distinct experiments settings and data splits from [24].

Table 5 presents the accuracy results on the three datasets. We report the results for Rank-based Self-Training considering the network models which presented the best accuracy results in isolation: APPNP [11] and SGC [12]. For the Rank Aggregation Self-Training, we considered two selected combination of pairs reported in Table 4: ARMA [13] + APPNP [11] with SGC [12]

as second stage classification and SGC [12] + APPNP [11] with APPNP [11] for second stage. As we can observe, the the proposed approach achieved the best accuracy results on the three datasets in comparison with all state-of-art methods (in bold).

Table 5: Comparison with state-of-the-art methods in terms of classification accuracy (%).

Method	**Cora**	**Citeseer**	**Pubmed**
Manifold Regularization [34]	59.5	60.1	70.7
Semi-Supervised Embedding [35]	59.0	59.6	71.1
Label Propagation (LP) [31]	68.0	45.3	63.0
DeepWalk [33]	67.2	43.2	65.3
Iterative Classification Algorithm (ICA) [32]	75.1	69.1	73.9
Planetoid [24]	75.7	64.7	77.2
Graph Convolution Netork (GCN) [8]	81.5	70.3	79.0
Graph Attention Network (GAT) [10]	83.0	72.5	79.0
Variance Reduction [27]	82.0	72.9	79.0
Self-Training [9]	80.5	69.9	78.3
Simple Graph Convolution (SGC) [12]	81.0	71.9	78.9
Deep Graph Infomax (DGI) [26]	82.3	71.8	76.8
Hypergraph Convolution and Attention [15]	82.7	71.2	78.4
Auto-Regressive Moving Average Filters (ARMA) [13]	83.4	72.5	78.9
Approx. Pers. Propagation of Neural Pred. (APPNP) [11]	83.3	71.8	80.1
Rank-based Self-Traing + SGC	84.1	73.1	75.9
Rank-based Self-Traing + APPNP	84.7	71.2	**81.2**
R. Agreg. Self-Training: SGC (ARMA+APPNP)	84.1	**73.5**	79.0
R. Agreg. Self-Training: APPNP (SGC+APPNP)	**85.1**	72.5	79.6

5. Conclusion

In this paper, we proposed a simple and effective self-training approach for Graph Convolutional Networks. Our approach uses a margin-based score

computed over log-soft-max layer of GCNs and analyzed through a rank-based model. Considering the crucial role of unlabeled data on semi-supervised learning, the proposed Rank-based Self Training approach allows an effective labeled set expansion and more accurate results on a second-stage classification. Evaluated on different GCN models, our approach achieved state-of-the-art results on benchmarks widely used in the literature. In future work, we intend to investigate the use of our approach on graph classification tasks, in addition to node classification.

Acknowledgments

The authors are grateful to Fulbright Commission, São Paulo Research Foundation - FAPESP (grants #2018/15597-6 and #2017/25908-6), Brazilian National Council for Scientific and Technological Development - CNPq (grant #308194/2017-9) and Microsoft Research. This work was also partly supported by the National Science Foundation Grant No. IIS-1814745.

References

[1] O. Chapelle, B. Schlkopf, A. Zien, Semi-Supervised Learning, 1st Edition, The MIT Press, 2010.

[2] I. Triguero, S. García, F. Herrera, Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study, Knowl. Inf. Syst. 42 (2) (2015) 245–284.

[3] Qi Tian, Jie Yu, Qing Xue, N. Sebe, A new analysis of the value of unlabeled data in semi-supervised learning for image retrieval, in: 2004 IEEE International Conference on Multimedia and Expo (ICME), Vol. 2, 2004, pp. 1019–1022.

[4] H. J. Scudder, Probability of error of some adaptive pattern-recognition machines, IEEE Trans. Inf. Theory 11 (3) (1965) 363–371.

[5] K. Sun, Z. Lin, Z. Zhu, Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-20), AAAI Press, 2020.

[6] H. Cai, V. W. Zheng, K. C. Chang, A comprehensive survey of graph embedding: Problems, techniques, and applications, IEEE Trans. Knowl. Data Eng. 30 (9) (2018) 1616–1637.

[7] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu, A comprehensive survey on graph neural networks, CoRR abs/1901.00596 (2019).

[8] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[9] Q. Li, Z. Han, X. Wu, Deeper insights into graph convolutional networks for semi-supervised learning, in: S. A. McIlraith, K. Q. Weinberger (Eds.), Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), AAAI Press, 2018, pp. 3538–3545.

[10] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[11] J. Klicpera, A. Bojchevski, S. Günnemann, Predict then propagate: Graph neural networks meet personalized pagerank, in: International Conference on Learning Representations, ICLR 2019, 2019.

[12] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, K. Weinberger, Simplifying graph convolutional networks, in: International Conference on Machine Learning (ICML), Vol. 97, 2019, pp. 6861–6871.

[13] F. M. Bianchi, D. Grattarola, L. Livi, C. Alippi, Graph neural networks with convolutional ARMA filters, CoRR abs/1901.01343 (2019).

[14] H. Pei, B. Wei, K. C. Chang, Y. Lei, B. Yang, Geom-gcn: Geometric graph convolutional networks, in: International Conference on Learning Representations, ICLR 2020, 2020.

[15] S. Bai, F. Zhang, P. H. S. Torr, Hypergraph convolution and hypergraph attention, CoRR abs/1901.08150 (2019).

[16] J. Klicpera, S. Weißenberger, S. Günnemann, Diffusion improves graph learning, in: Advances in Neural Information Processing Systems, NeurIPS 2019, 2019, pp. 13333–13345.

[17] Z. Zhou, S. Zhang, Z. Huang, Dynamic self-training framework for graph convolutional networks, ArXiv abs/1910.02684 (2019).

[18] B. Settles, Active learning literature survey, Computer Sciences Technical Report 1648, University of Wisconsin–Madison (2009).
URL `http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf`

[19] T. Scheffer, C. Decomain, S. Wrobel, Active hidden markov models for information extraction, in: International Conference on Advances in Intelligent Data Analysis, IDA '01, Springer-Verlag, Berlin, Heidelberg, 2001, p. 309–318.

[20] D. C. G. Pedronette, R. d. S. Torres, Image re-ranking and rank aggregation based on similarity of ranked lists, Pattern Recognition 46 (8) (2013) 2350–2360.

[21] S. K. McCallum, K. Nigam, J. Rennie, K. Seymore, Automating the construction of internet portals with machine learning, Information Retrieval 3 (2000) 127–163.

[22] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, T. Eliassi-Rad, Collective classification in network data, AI Magazine 29 (3) (2008) 93.

[23] C. L. Giles, K. D. Bollacker, S. Lawrence, Citeseer: An automatic citation indexing system, 1998.

[24] Z. Yang, W. W. Cohen, R. Salakhutdinov, Revisiting semi-supervised learning with graph embeddings, in: International Conference on International Conference on Machine Learning, ICML'16, JMLR.org, 2016, p. 40–48.

[25] M. Fey, J. E. Lenssen, Fast graph representation learning with pytorch geometric, CoRR abs/1903.02428 (2019).

[26] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, R. D. Hjelm, Deep graph infomax, in: International Conference on Learning Representations, ICLR 2019, 2019.

[27] J. Chen, J. Zhu, L. Song, Stochastic training of graph convolutional networks with variance reduction, in: International Conference on Machine Learning, ICML 2018, Vol. 80, 2018, pp. 941–949.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in pytorch, in: NIPS-W, 2017.

[29] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), International Conference on Learning Representations, 2015.

[30] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, Journal of Machine Learning Research 9 (2008) 2579–2605.

[31] X. Zhu, Z. Ghahramani, J. Lafferty, Semi-supervised learning using gaussian fields and harmonic functions, in: Proceedings of the Twentieth International Conference on International Conference on Machine Learning, ICML'03, 2003, p. 912–919.

[32] Q. Lu, L. Getoor, Link-based classification, in: International Conference on International Conference on Machine Learning, ICML'03, AAAI Press, 2003, p. 496–503.

[33] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, 2014, p. 701–710.

[34] M. Belkin, P. Niyogi, V. Sindhwani, Manifold regularization: A geometric framework for learning from labeled and unlabeled examples, Journal of machine learning research 7 (2006) 2399–2434.

[35] J. Weston, F. Ratle, R. Collobert, Deep learning via semi-supervised embedding, in: International Conference on Machine Learning, ICML '08, 2008, p. 1168–1175.