A Study of Neural Networks for Network Representation

Man Wu

Abstract

Network structure is very common in our daily applications, especially in the systems involving users. Due to the particularity of the network structure, it is challenging to deal with network relationships (finding suitable network structure representations), especially for large network representation, it may require much more computational complexity. I have implemented a recently proposed methods DNGR(Cao et al., 2016) to learn a effective low dimensional representation of the Wiki dataset, and used different evaluation criterion to assess the classification performances and using the t-distributed stochastic neighbor embedding (t-SNE) method to gain an visualization of it. Semi-supervised learning is worth to be explored later.

1. Introduction

Networks are a general language for describing complex systems and indeed exist in our daily lives. For example, social media such as Facebook and Twitter making up social networks among people; thousands pages of the Internet making up a web-linked network; roads and highways between national cities making up a traffic network; Even our cells and brains making up a network in our body. Networks are everywhere! The analysis of the networks plays a vital role in many emerging applications of many disciplines. For example, in social networks, users are classified as meaningful social groups, which is a useful method for many important tasks such as user search, targeted advertising and recommendation; in communication networks, community structure detection is helpful to understand the rumor spreading process better; in biological networks, protein interaction inference can promote new treatment to the disease.

It is well recognized that network data is often sophisticated so it is challenging to deal with. To process network data effectively, the first critical challenge is to find effective network data representation, that is, how to represent networks concisely- advanced analytic tasks can be conducted efficiently in both time and space. So how to represent these networks? In general, a discrete adjacency matrix is used to represent a network that captures only the adjacent representations relationships between vertices. However, this simple representation can not reflect more complex, higher-order structural relationships, such as paths, frequent substructures and so on.

Traditionally, we usually represent a network as a graph G = (V, E), where V is a vertex set representing the nodes in a network, and E is an edge set representing the relationships among the nodes. For large networks, such as those with billions of nodes, the traditional network representation poses several challenges when processing and analysis the network. Recently, network representation learning (NRL) has aroused many researcher's attention, NRL is to learn latent and representate low-dimensional of network vertices, preserving network topology structure, vertex content, and other side information(Yang et al., 2015). Network representation learning is generally referred to the network embedding method. Briefly speaking, it turns a structure (node, edge or subgraph) in a network into a multidimensional vector, which makes the complex network relationships become structured multi-dimensional features. Thus, we can use machine learning methods to achieve more convenient algorithms.

Network representation learning is capable of supporting subsequent network processing and analysis tasks such as node classification, node clustering, network visualization and link prediction(Cao et al., 2016). Recently, many methods have been proposed in network representation learning, such as DeepWalk (Perozzi et al., 2014) and LINE(Tang et al., 2015). DeepWalk is the method introduced the deep learning technology to network representation learning firstly and it is actually an improved method comparing to previous ones. That is to say, the method of deep learning has a great advantage over traditional methods. Therefore, there are many questions deserve further study, for example, "How to deal with network data in deep neural networks?" and "How deep is the neural network?".

The purpose of the project is to explore and modify the state-of-the-art network embedding methods to do some specific tasks (in this project, I will do the node classification task). The method I used is DNGR (deep neural networks for graph representations), which uses the unsupervised stacked auto-encoder neural network model to extract complex features of high dimensional input matrix(Cao et al., 2016). The first thing I want to do is to explore the effect of the number of the layers based on the auto-encoder neural network. And then I would like to implement the classification task and compare the performances in different number of layers. Finally, I will make a plan for future work and make some conclusions for this project.

2. Feature Extraction

Feature extraction is a data processing method aims to reduce the amount of resources when describe a large set of data and build derived values (features) intended to be informative and non-redundant in machine learning, pattern recognition or image processing(Guyon & Elisseeff, 2006). Feature extraction is also related to dimensionality reduction. When performing analysis of complex data, one of the significant problems maybe the varies variables and high-dimension calculates, which requires a large amount of memory and computation power, in addition, it also may cause over-fitting when training a classification algorithm and can not generalize new samples well. Feature extraction is a general method to solve these problem, describing the data accuracy while reducing the unnecessary complexity(dimension).

The machine learning algorithm can not recognize raw data. At that point, feature extraction need to be used to change unrecognizable data into recognizable data, especially in text and image. The text must be transformed into a quantifiable eigenvector which can be read and processed by machine learning, one of the most common text representation method is Bag-of-words model which regarded the text as a set of words and each word exist individually. And Hashing Trick can determine the position of the word block at the index of the feature vector(Guyon et al., 2008). In addition, some dimensionality reduction methods(like PCA, SVD)are important since the more dimension used, the more training model required. The feature extraction of image use pixel values to extract features and majority image feature is not sparse, which is not similar to text, according to Guyon et al. (2008).

Feature extraction is also used in the network, the most popular method used in graph/network feature extraction is network embedding(or called graph representation and node2vector). In this method, the network nodes are regarded as embedding vectors and their relationship are edges. The detail description will be introduced in the following content.

3. Methods

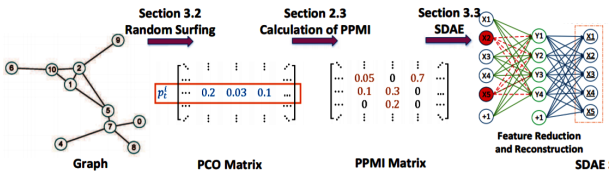

Figure 1. Network Nodes Represent Learning Flowcharts

The figure 1 describes the steps of network learning model I used. The raw dataset is a graph and I use several kinds of matrix to process the dataset,and then use auto-encoder to reduce the dimension and extract features. I will explain these steps in detail in the following text.

3.1. Graph Adjacency Matrix

I can consider the network what I am interested in as a graph G (V, E), that consists of many vertices(V), and edges(E) which connect each two of the vertices. Each vertex in the graph represents a node in the network, and the edge in the graph represents the relationship of the corresponding two nodes in the network. Due to the particularity of the network, this traditional representation of the network poses challenges to further understanding and processing the network, regarding the high computational complexity, and lower parallelism, as well as the unavailability to the ML methods (ML methods usually require independent vector representation of each examples, while here the nodes in the network are dependent to each other to some degree). Those problem becomes severe as the network becomes further larger. Therefore, effectively representation of the network is extremely important for many network related applications, i.e., node classification, node clustering, link prediction, and network visualization.

Graph Adjacency Matrices is an intuitive representation of network, which is quite direct and non-novel, but worth to have a look at. This is usually the rawest information we get from the network, and the subsequent feature extraction processes can be applied based on it. Assume we have a graph $G(V, E)$, which contain n nodes, then its adjacency matrix A is a n by n square matrix, defined by:

$$A[i][j] = \begin{cases} 1 & \text{if } (i, j) \epsilon E \\ 0 & \text{if } else. \end{cases} \tag{1}$$

For this experiment, I have simplified the problem. I only consider whether there is an edge between each two nodes or not, and do not consider the degree of association between them, which means there are no weights on each edge.

3.2. Random Surfing

The process of finding a more effective representation of the network also called the network embedding. According to the type of information preserved in the embedding process, the network embedding methods can be categorized into Structure and Property preserving Network Embedding, Network Embedding with Side Information, Advanced Information Preserving Network Embedding. This classification method was proposed by Cui et al. (2017).

As for the model, the Random Walk (Perozzi et al., 2014), which is commonly used to capture unweighted graph structural information by the sampled linear sequences, is used to transforming a graph structure into linear sequence. It was inspired by the skip-gram model(Mikolov et al., 2013) in the word2Vec. The RandomWalk is a sampling process of the graph, and the results are chunks of ordered nodes sequences, which can be reckoned as the sentence (word sequence) in the corpora, and then we can apply the skip-

gram model in these sampled node sequences. Experiments show that the network has a high efficiency when the network is large. But this method has some limitations caused by the setting of the hyper parameters (walk length and total walks). The Random surfing methods proposed by Cao et al. (2016) addresses this problem, which do not need to worry about the setting of the hyper-parameters in the previous methods and can directly get a probabilistic co-occurrence(PCO) matrix without the need of a lengthy sampling process. Deep neural network and its variations also used to learn different levels of feature representations.

I use the random surfing method in my experiment. First, I need to randomly order the nodes in the graph, and assuming the current vertex is the i^{th} vertex. I use the graphs adjacency matrix A to generate the transition matrix T to indicate the transition probabilities of all the two nodes, which only need to divide by the total number of edges that related to a certain node. Row vector p_k indicates the probability of reaching each vertex after k steps of transitions given starting from the i^{th} vertex. Supposed that we have a random surfing model with a probability of $(1-\alpha)$ that the model will restart the procedure from the original node i^{th}, and then we have:

$$p_k = \alpha p_{k-1} A + (1-\alpha) p_0 \tag{2}$$

Where the p_0 is the initial one 1-hot vector with the value of the i^{th} entry being 1 and all other entries being 0. Without the restart mechanisms, the random surfing model is as follows:

$$p_k^* = p_{k-1}^* A = p_0 A^k \tag{3}$$

Then I have now obtained the probabilistic co-occurrence matrix.

3.3. PPMI

I use the probabilistic matrix obtained from the previous step to generate the Positive Pointwise Mutual Information (PPMI) matrix(Bullinaria & Levy, 2007), which was an amended version of PMI. The PMI matrix is a kind of association measurement, which was defined as:

$$PMI_{w_1,w_2} = log(\frac{\#(w_1,w_2)|D|}{\#(w_1)\#(w_2)}) \tag{4}$$

where $|D| = \sum_{w_1}\sum_{w_2} \#(w_1,w_2)$.

and the PPMI is just forcing the negative entry in the PMI to be zero, which is defined as:

$$X_{w_1,w_2} = max(PMI_{w_1,w_2}, 0) \tag{5}$$

As I can see, the PPMI matrix that obtained from the above procedure is still a very high dimensional matrix, which has the size of n by n (n is the number of vertex in the graph). Singular Value Decomposition is a straightforward procedure to get a linear transformation from the original vector space of the PPMI matrix to a low-dimensional vector representation. However, as stated, SVD can only map linear relationship, while the deep neural network has provided a non-linear alternative to mapping, that is, the auto-encoder methods can be effectively used to learning a low-dimensional vector representation from the high dimensional vertex representation in PPMI matrix.

3.4. Stacked Denoising Auto-encoder

The stacked Auto-encoder can be regarded as an effective way to learning high-level abstractions from low-level features (generally in high dimension),which consists two parts, one is encoding, the other is decoding. The encoding procedure relates to a function f_{θ_1}, which mapping the input to a new feature space, and the activation function in hear can model the non-linearity between these two spaces. The decoding procedure relates to a function g_{θ_2}, which can reconstruct the original input vectors from the learned latent representation. Specifically:

$$f_{\theta_1}(x) = \sigma_1(W_1 x + b_1) g_{\theta_2}(y) = \sigma_2(W_2 y + b_2) \tag{6}$$

where $\theta_1 = \{W_1, b_1\}$, $\theta_2 = \{W_2, b_2\}$

For Auto-encoder, we need to minimize the following reconstruction loss function:

$$\min_{\theta_1,\theta_2} \sum_{i=1}^{n} L(x^{(i)}, g_{\theta_2}(f_{\theta_1}(\tilde{x}^{(i)})) \tag{7}$$

Where the $\tilde{x}^i$ is the corrupted input entry of x^i, which was introduced to this model to increase the robustness. This method for short is SDAE.

3.5. Logistic Regression

Finally, I use the learned low-dimensional representation from the SDAE procedure as the inputs to my classifier. Here I used multiple Logistic Regression as the classifiers, and applied various evaluation criterion, such as micro, macro and weighted et.al., to assess the classification results.

This graph(network) representation model described above is called Deep Neural networks for Graph Representations(DNGR for short), and was first proposed in 2016 by Cao et al. (2016).

4. Experiments

In this section, I do the classification task on the network nodes to evaluate the performance of our model. I try to explore and improve the best benchmark method of network representation method called DNGR (deep neural networks for graph representations) until 2016, which was proposed by Cao et al. (2016).

4.1. Dataset and details

I choose the Wiki(Sen et al., 2008) dataset provided by openNE toolkit (https://github.com/thunlp/OpenNE) for node classification task, which contains 2405 web pages from 19 categories and 17981 links among them. The Wiki

dataset was composed by two files: "Wiki_edgelist.txt" and "Wiki_category.txt". The "Wiki_edgelist.txt" file stores the relationship graph between different nodes, which is a 2045 by 2045 adjacent matrix. The "Wiki_category.txt" file stores the actual classification of these nodes, which is used to evaluate the performance of my model in node classification task. I firstly use the PPMI method which is similar to a normalized method to preprocess the original adjacent matrix as the input of the auto-encoder neural network. Then I use the auto-encoder neural network for training to get the best feature representation of the reduced-dimension nodes (In this experiment, I set the reduced nodes to 128, i.e. After training in the auto-encoder neural network, the number of the nodes decreased from 2045 to 128). In the process of training, I try to minimize the standard squared loss of the input and output, which is used to evaluate the performance of training. After training, I use half of the 128 trained nodes to do the classification task. Meanwhile, I get the score of the 'micro', 'macro', 'samples' and 'weighted', which is widely used as the evaluation of the classification performances(Tang & Liu, 2009a;b).

4.2. Baseline

I have considered several baselines to demonstrate the effectiveness and robustness of my system, for example DeepWalk(Perozzi et al., 2014), LINE(Tang et al., 2015) and so on. And finally I choose the state-of-the-art method DNGR (deep neural networks for graph representations), which use the stacked auto-encoder neural network model to extract complex features(Cao et al., 2016). Their activation function is Sigmoid and they only use one layer for this neural network. In order to explore a better method, I have done some changes based on the DNGR method. On the one hand, I tried different layers (from one to five) of the auto-encoder neural network to train nodes. On the other hand, I changed the activation function to Relu(Nair & Hinton, 2010), which is seen as a promoted function of Sigmoid. After extracting the features of the nodes, I used 50% of the trained nodes as the DNGR do to carried out the classification task. I use a one-vs-rest logistic regression as the classifier to implement the multi-label classification(Fan et al., 2008). The aim of my baseline experiment is to explore the effect of the numbers of the layer of the auto-encoder neural network and find a relatively better number of layer for my dataset.

4.3. Results

4.3.1. Comparison

I trained the data on the auto-encoder neural network with different hidden layers (from 1 layer to 5 layers) and each run I trained 400 epochs. The results are shown in figure 2. In addition, for the 5 layers, when at epoch 0, its loss value is nearly 352, which is very high and can affect the representation of other number of layers, so I removed the value at epoch 0 of the 5 layers. It is obvious to see that after two layers, as the number of hidden layers increases, the oscillation of the curve becomes more intense and the value of the loss increases. After comparing the different layers, I decided to abandon the 4 and 5 layers that have relatively bad performances than the other layers. In order to do the further comparison, I focused on the 250 epochs to 400 epochs of the one, two and three layers and the results are shown in figure 3. From figure 3, I can see that when the number of hidden layer is 2, the final value of the loss is the lowest. So I may assume that two hidden layers is more suitable than other number of layers for this dataset.

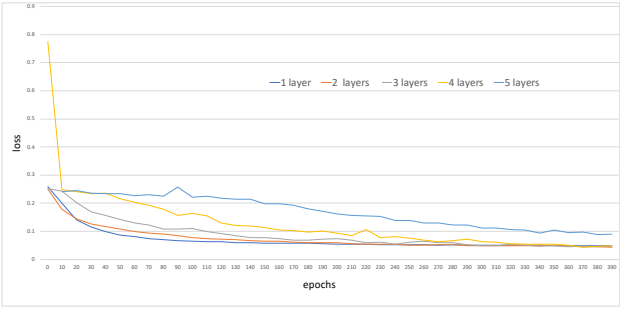

Figure 2. Comparing the Value of Loss of Different Number of Layers (1-5 layers)

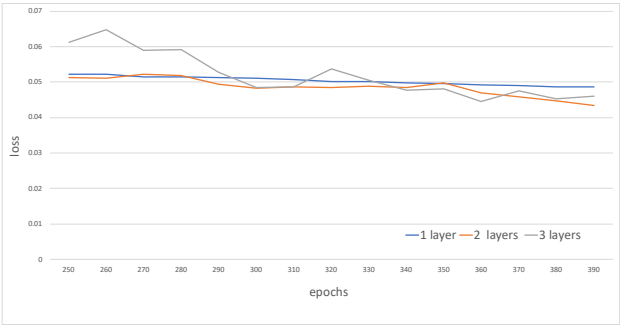

Figure 3. Comparing the Value of Loss of 1-3 Layers Between 250-400 Epoch

However, I cannot jump to the conclusion that I still need to do the classification task to verify that two-layers is the best choice. Then I compared the scores of the 'micro', 'macro', 'samples' and 'weighted' for the five different numbers of layers and the results are showed in table 1. However, in table 1, the four evaluation criteria of the five different number of layers are almost the same, which cannot be as a reference for me to choose the appropriate number of layers.

No. of Layers	Micro	Macro	Samples	Weighted
1 layers	0.6592	0.5450	0.6592	0.6560
2 layers	0.6542	0.5237	0.6542	0.6503
3 layers	0.6617	0.5461	0.6617	0.6602
4 layers	0.6584	0.5373	0.6584	0.6575
5 layers	0.6459	0.5347	0.6459	0.6413

Table 1. Comparing Four Variables of Different Number of Layers

4.3.2. Visualization

In order to show the performances of different numbers of layers in the classification task more intuitively, I use t-SNE to visualize the results (Maaten & Hinton, 2008). t-SNE is a non-linear dimensionality reduction algorithm that is well suited for dimensionality reduction of high-dimensional data to two or three dimensions for visualization(Maaten & Hinton, 2008). The results are showed from figure 4 to figure 8. In order to identify the performance of the classification task in the figures clearly, I randomly selected three categories in the original 19 categories. Through these intuitive figures, I can roughly see that the classification performance of the 2 layers is better than others.

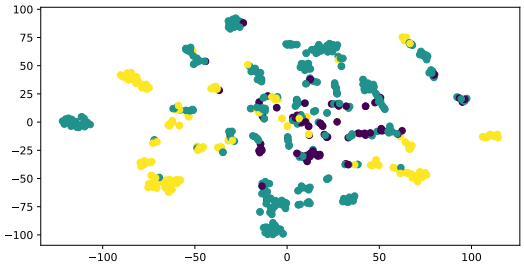

Figure 4. Classification Results in 1 Layer

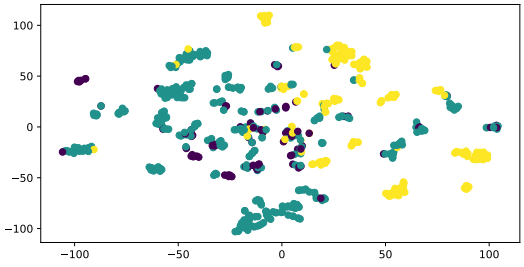

Figure 5. Classification Results in 2 Layers

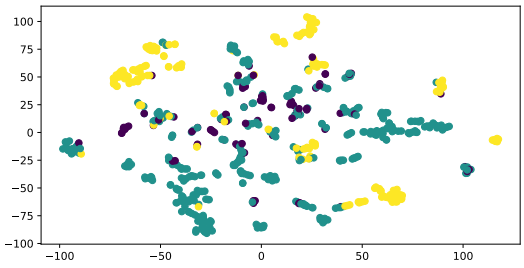

Figure 6. Classification Results in 3 Layers

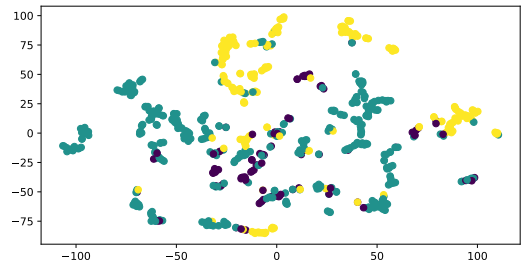

Figure 7. Classification Results in 4 Layers

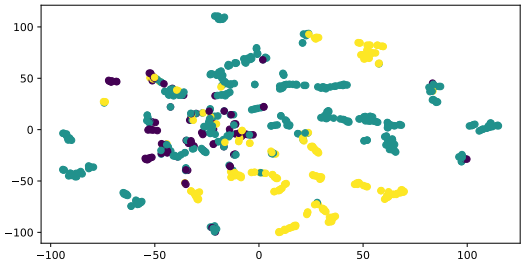

Figure 8. Classification Results in 5 Layers

5. Future Work

I have run the baseline and what I want to do in the following work mainly includes three parts:

Firstly, optimizing baseline and adding labels(supervised learning). As I mentioned before, auto-encoder is a kind of unsupervised learning method, but supervised learning always perform better than unsupervised learning by using same method in general, so I want to implement experiments to test it in two ways: 1.Add label information; 2.Add network nodes' information. In addition, the hyper-parameters I used also can be changed.For example: I used 50% data as training data before so I can increase the rate; change the steps of random surfing(4 steps initial); param-

eter α in random surfing, the number of hidden-dim; the activate functions(Relu initial).

Secondly, Generative Adversarial Network(GAN) is a famous unsupervised method in machine learning and introduced by Goodfellow et al. (2014) in 2014. I want to apply GANs in the experiment since GAN has robustness, so it can reduce the effect of environment or noise. GAN includes two models: a generative model G that captures the data distribution and a discriminative model D that estimates the probability that a sample came from the training data rather than G. This method can generate image which look at least superficially authentic to human observers, having many realistic characteristics(Salimans et al., 2016).

Thirdly, Expanding the range of applications and implement experiments on more datasets. More applications like node clustering, link prediction, and network visualization can be done besides node classification. What's more, I can implement the baseline or optimized method on other datasets to test the universality of my method and find general rules.

6. Conclusion

In this report, I set up the baseline based on DNGR(Cao et al., 2016) and explored the depth of hidden layers of auto-encoder neural network as well as the multiple label classification task. In order to show the performances of different numbers of layers in the classification task more intuitively, I use t-SNE(Maaten & Hinton, 2008) to visualize the trained nodes on two dimension space. Through the experiments, I find that two hidden layers rather than one hidden layer used in DNGR performs best on the Wiki dataset. I can also conclude that the complexity or deeper network may not better than the simple ones, efficiency and accuracy are the most important factors I need to consider. In the future work, I will improve our baseline, try GAN algorithm, and do more experiments on another datasets for other applications.

References

Bullinaria, John A and Levy, Joseph P. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior research methods*, 39(3): 510–526, 2007.

Cao, Shaosheng, Lu, Wei, and Xu, Qiongkai. Deep neural networks for learning graph representations. In *AAAI*, pp. 1145–1152, 2016.

Cui, Peng, Wang, Xiao, Pei, Jian, and Zhu, Wenwu. A survey on network embedding. *arXiv preprint arXiv:1711.08752*, 2017.

Fan, Rong-En, Chang, Kai-Wei, Hsieh, Cho-Jui, Wang, Xiang-Rui, and Lin, Chih-Jen. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Guyon, Isabelle and Elisseeff, André. An introduction to feature extraction. In *Feature extraction*, pp. 1–25. Springer, 2006.

Guyon, Isabelle, Gunn, Steve, Nikravesh, Masoud, and Zadeh, Lofti A. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.

Maaten, Laurens van der and Hinton, Geoffrey. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Perozzi, Bryan, Al-Rfou, Rami, and Skiena, Steven. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

Salimans, Tim, Goodfellow, Ian, Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, and Chen, Xi. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pp. 2234–2242, 2016.

Sen, Prithviraj, Namata, Galileo, Bilgic, Mustafa, Getoor, Lise, Galligher, Brian, and Eliassi-Rad, Tina. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

Tang, Jian, Qu, Meng, Wang, Mingzhe, Zhang, Ming, Yan, Jun, and Mei, Qiaozhu. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

Tang, Lei and Liu, Huan. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 817–826. ACM, 2009a.

Tang, Lei and Liu, Huan. Scalable learning of collective behavior based on sparse social dimensions. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pp. 1107–1116. ACM, 2009b.

Yang, Cheng, Liu, Zhiyuan, Zhao, Deli, Sun, Maosong, and Chang, Edward Y. Network representation learning with rich text information. In *IJCAI*, pp. 2111–2117, 2015.