GraphNet: Recommendation system based on language and network structure

Rex Ying
Stanford University
rexying@stanford.edu

Yuanfang Li
Stanford University
yli03@stanford.edu

Xin Li
Stanford University
xinli16@stanford.edu

Abstract

Recommendation systems are widely used in many popular online services, however, present algorithms use either network structure or language features only. In this paper, we present a scalable and efficient recommendation system that combines both language content and complex social network structure. Given a dataset consisting of objects created and commented on by users, the system predicts other content that the user may be interested in. We demonstrate the efficacy of our system through the task of recommending posts to reddit users based on their previous posts and comments. We extract the language feature using GloVe vectors and sequential model, and use attention mechanism, multi-layer perceptron and max pooling to learn hidden representations for users and posts. We show in experiments that our method is able to achieve the state-of-the-art performance.

1 Introduction

In many popular social networks today, users interact and connect with each other via written language. On Amazon, users write and read reviews for products; on reddit.com, users write posts and comment on them; on Twitter, users view and post tweets and follow those users whose tweets interest them. In such settings, it is often beneficial to build recommendation systems which, given a history of the content that a user has previously shown interest in (e.g. through re-tweeting or commenting), can recommend other content that may interest them. In this task, both language content and social network structure are useful in making recommendations.

However, most previous algorithms use only one of the above in recommendation tasks. Methods that consider both, such as Yang *et. al.* [20], assume that features representing network structure can be computed using multi-layer perceptron but such an assumption does not hold given the complex network structure in a large social network (see the experiments section). Here we present a machine learning algorithm that combines information from both language and social network structures to build a scalable, efficient and state-of-the-art recommendation system. In general, given a dataset consisting of objects created and commented on by different users, we aim to be able to provide users with recommendations of other objects they may be interested in.

We use a concrete example of reddit.com recommendations to illustrate our task: given a bipartite network $G = (\{U, P\}, E)$, and language content $F(v)$ associated with each node $v \in U \bigcup P$, where U denotes the set of users and P denotes the set of posts, we want to predict posts that a user might be interested in based on the history of the user's comments on other posts. It is important to note that the set of users for which we want to recommend posts at test time is *not* necessarily a subset of the users available at training time, since new users can join a social network at any time. We show in experiments that our method achieves the state-of-the-art performance.

An important feature of our algorithm is that it allows inductive inference. We define inductive inference on networks as inference of labels for nodes or edges that are not present in the network at training time. In practice social networks evolve over time and thus inductive inference is crucial for making recommendations to users and for posts that were not available when training the model.

Another advantage of inductive inference is that when building recommendation systems on large social networks with billions of users, existing algorithms will require tremendous computational resources and time to train on the entire network. Thanks to inductive inference, however, our algorithm can be run on a much smaller part of the network and make recommendations for users in the entire network. Our experimental results show that our model is able to transfer weights learned from a small part of the network and make recommendations for the entire social network, without losing much performance.

1.1 Related work

Learning representations for documents have been extensively studied. While documents can have variable length, most machine learning algorithms require a fixed length representation as input. Thus we need an appropriate method of extracting feature representations from documents for use in downstream tasks. This is analogous to word vectors but for entire documents. The Paragraph Vector method proposed by Le et al [10] uses an approach similar to Word2Vec [3] to predict the next word based on context words sampled from the whole paragraph. Recently the sequence to sequence learning approach [16] has become prevalent in various NLP tasks, including translation, captioning and question answering systems. The encoder of such a system uses a recurrent neural network (RNN) to compute a hidden representation of a sentence or document. The vanishing gradient problem is addressed via a gating mechanism such as Long Short Term Memory (LSTM). In this work, we use ideas in learning document representations while incorporating network structure in the representations.

Network embedding is a common technique that learns a representation for nodes in graphs based on network structure. Recent methods such as LINE [17], DeepWalk [14] and Node2Vec [4] make use of this idea and, similar to Word2Vec, optimize for prediction of nodes in the neighbourhood of a given node based on its representation. These approaches, however, do not make use of the rich features provided by the language content associated with nodes.

The difficulty faced by many network-based recommendation algorithm is dealing with new users and content that were not available at training time. For example, methods based on neighbourhood prediction such as DeepWalk assign a unique embedding for every node in the network. Such an algorithm is not inductive in nature since addition of new users/products in the network would require running the training algorithm again.

Recently Kipf *et. al.* [7] proposed a graph convolution network (GCN) that combines input features and network structure in learning node representations by performing convolutions. This approach, however, trains on the entire network data instead of minibatches, and is thus not scalable. In this work, we demonstrate that the GCN is a special case of our general model, and propose modules in the architecture that improves the performance compared to the GCN model. Our implementation allows minibatch training and is efficient and scalable.

2 General framework to learn language from networks structured data

Our general framework consists of the following steps: (1) extract language features from contents of users; (2) for each user and post, sample intelligently a set of similar users and posts; (3) for each user and post, use a deep architecture to aggregate information from the features of its sampled similar users and posts and output a representation for each user and post, which captures both its language features and the network structure; and (4) use a loss function specific to the task to train the model. In this section, we explain each stage in more detail.

2.1 Language feature extraction

We first compute the GloVe word vectors [13] for each word that appears in a post/comment. The feature vector for each post, p, is the average of the GloVe vectors of all words present in the post. Similarly, the feature vector for each user, u, is the average of the GloVe vectors of all words written by the user. Our network takes these feature vectors for users and posts as inputs, and outputs a representation for each user and post in the minibatch. At training time, gradient information is also back-propagated into the word vectors to improve its accuracy. It has been shown that using a sequence model such as LSTM instead of averaging word vectors can increase the performance of various downstream tasks ([16]). Hence we also explored using a sequence model to encode posts and comments to be used as feature inputs to our network and compared the performance.

In addition, we concatenate metadata such as the number of posts a user writes and the number of comments a post receives to the language features for users and posts respectively.

2.2 Neighbour sampling

When determining which posts to recommend to a given user, we consider the posts commented on by similar users, or neighbours, in social network graphs. However this method can introduce high variance and bias into our model as follows: an extremely active user who comments on many posts will be in the neighbourhood of many users even if they are not similar and likewise a very popular post commented on by many users will cause all these users to become neighbours. In order to mitigate the effects of these two cases, we utilize only a sample of a user's neighbours. To this end, we implement an attention mechanism using multi-layer perceptron to learn which subset of neighbours are most helpful in providing a user with reasonable recommendations.

Another advantage of sampling is that it reduces the memory requirement of the network and makes minibatch training possible.

2.3 Feature aggregation

In a typical convolutional neural network as applied to images, we can use a fixed kernel size to operate on the neighbours of a sample. This is because each pixel of an image has a fixed location with respect to neighbouring pixels - that is we can think of a pixel being to the right or left of another pixel. However in a graph we do not have a fixed number of neighbours for each user/post and there is no sense of ordinality. Consequently, we need a method to aggregate the neighbours' features into a single representation that can accept a variable number of inputs as well as remain invariant to the order of the samples.

In practice, there are various ways to aggregate features that meet the requirement. Common approaches include: (1) use an RNN to encode the neighbouring features one-by-one as a sequence, and augment with different permutations of the features so that ideally the RNN learns an order-invariant model for feature aggregation; (2) define a consistent canonical ordering of the features based on graph structure, and aggregate features using this ordering; and (3) use a symmetric function $f(x_1, x_2, \ldots, x_n)$, whose value does not change when the input $\{x_i | 1 \leq i \leq n\}$ is permuted. We now discuss these approaches in detail.

By augmenting the inputs with random permutations of the features and using the same downstream supervision signal, we hope that the RNN can learn an order invariant function. This approach, however, relies on the assumption that the number of neighbouring features to be aggregated is relatively small, i.e. in the order of 10's. This is because the effective memory of the RNN (with LSTM) is still limited. Once the number of neighbouring features exceeds 100, the RNN with LSTM will experience vanishing gradients and fail to learn a good aggregation function. Another limitation to this approach is that augmentation of the training data uses many random permutations, leading to large computation overhead.

The approach of aggregation based on a canonical ordering has been explored in [12], which proposes computing a canonical labelling according to graph automorphism [11] which respects the graph colouring defined by the colour refinement algorithm [6].

In practice, it is also common to compute an average or weighted average of the features. In the YouTube recommendation system [1], a user's history of videos watched is aggregated by taking the average of individual video features. The Graph Convolutional network (GCN) also applies averaging when aggregating over neighbouring features. Averaging is one possible symmetric function, but it might not be suitable for the downstream tasks. We thus explore the following more general approach that has shown good performance in computer vision [15].

Our model aggregates the features via MLP and max-pooling to estimate the optimal aggregation function. Suppose that the best symmetric aggregation function for our task is $f(x_1, x_2, \ldots, x_n)$, where $\{x_i | 1 \leq i \leq n\}$ are the neighbouring features. We approximate it using Equation (1), where g is an MLP learned by the network, and MAX is the max pooling operation. It can be shown that the composition of max pooling and MLP is able to approximate any such symmetric aggregation function arbitrarily closely, given a sufficient number of neurons [15]. We will show in experiments that this model learns a meaningful aggregation function and compare the performance with other alternatives.

$$f(x_1, x_2, \ldots, x_n) \approx \mathrm{MAX}(g(x_1), g(x_2), \ldots, g(x_n)) \tag{1}$$

3 Deep learning on network structured language data

3.1 Recommendation algorithm

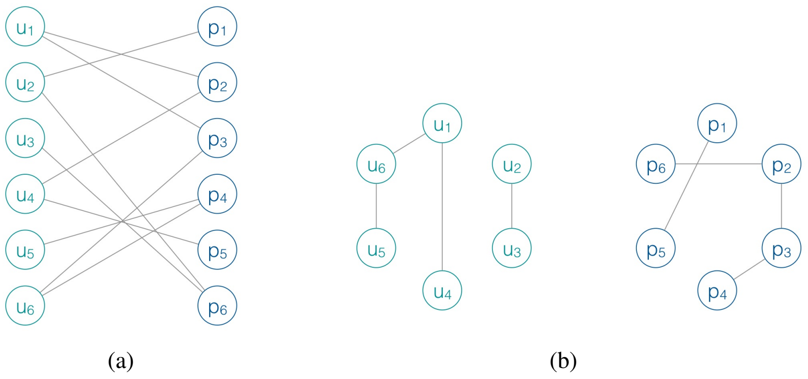


Figure 1: (a) Bipartite graph of users and posts. (b) Induced networks of users and posts.

We introduce our GraphNet architecture applied to our problem of language and network-based recommendation system. We model the training data using a bipartite graph, as presented in Figure 1 (a). User u_i is connected to post p_j if and only if u_i comments on p_j.

As explained in Section 2.1, users and posts have features of different dimensions. We therefore use two different networks for users and posts respectively. The two networks have different weights, but use the same architecture. To this end, we decompose the bipartite graph into two induced graphs for users and posts. The induced graphs are shown in Figure 1 (b). In the induced graph for users, two users are connected if and only if they both commented on the same post. The idea is that users who have similar interests tend to comment on the same posts. Also, posts commented on by the same user are connected in the induced network for posts.

3.2 GraphNet architecture

An example of the GraphNet architecture that we developed is illustrated in Figure 2, which has 2 layers.

We use n to denote the batch size. The input features are the language features for each node in the bipartite graph (see Figure 1). These are passed into a sampler, which samples from the neighbourhood of nodes a subset that is useful for the downstream task. A naive implementation is a uniform random sampler. However, we could use an attention-based sampler that uses MLP and softmax layers to do sampling. Specifically we use the Gumbel Softmax [5] such that the sampling stage is differentiable and allows gradients to be back-propagated to weights for samplers as well as weights for computing the input language features. The example architecture has two layers, so we apply the sampler twice to generate a set of sample nodes that are within the 2-hop neighbourhood of each input node in the minibatch. We use n_2 to denote the total number of samples in the minibatch. Thanks to the sampling stage, we can describe the neighbourhood of each node using a constant

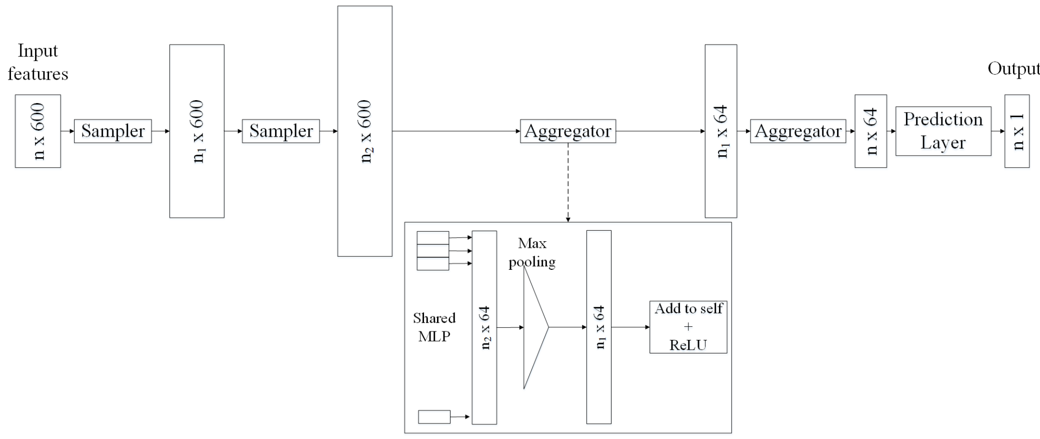


Figure 2: Architecture overview. The link prediction network takes a batch of n user/post vectors, D, as input, applies 2 sampling layer to collect all document features we need to compute the hidden representations for D, and aggregates features via MLP and max pooling. The prediction layer takes the final hidden representation as input and outputs a score for each edge.

number of samples, i.e. $n_2 = O(n)$. In practice, we sample 25 nodes in the 2-hop neighbourhood of each node.

The aggregator then aggregates the features of neighbours of each node and compute a representation of the node. In Figure 2, we illustrate the computation of the MLP and max pooling aggregator of the second layer introduced in Section 2.3. We apply an MLP to each input feature produced by the sampler, and use max pooling across all neighbouring feature of each node in the 1-hop neighbourhood of nodes in the minibatch. A mathematical expression of this aggregation is also shown in Equation 2. Here h_i^t is the hidden representation of node $v_i \in V$ at layer t. W_1 and W_2 are weights for self feature and neighbour feature respectively. f is the aggregation function, which takes in all sampled neighbour features and outputs an order-invariant representation of the neighbourhood.

$$h_i^t = W_1 h_i^{t-1} + W_2 f(\{h_j^{t-1} | v_j \in S(v_i)\}) \tag{2}$$

Since the network has 2 layers, we again apply the aggregator twice to obtain the final representation of all nodes in the minibatch. In this process, only the first layer representations are computed for node samples that are within the 1-hop neighbourhood of nodes in the minibatch, since they are only used to compute the second layer representations of nodes in the minibatch. Note that the architecture has a recursive structure: the second layer sampler and aggregator are next to each other.

Once the hidden representation of nodes in the minibatch is obtained, we can then use a prediction layer to compute the output. The prediction layer will depend on the actual task at hand. In our recommendation system setting, we introduce the loss function in Section 3.3.

3.3 Readout layer and loss function

The GraphNet architecture outputs a hidden vector representation for each user and post. An affinity score between a post and a user is defined as: $\mathbf{u}^T A \mathbf{p}$, where A denotes the bilinear weights to be learned and u and p are the representations of a user and a post output by the aggregator. We use the bilinear weights A since $\mathbf{u}$ and $\mathbf{p}$ could have different dimensions, and it allows interaction between different dimensions of u and p. Since the number of posts could be extremely large, we compute a negative sampling loss [3] instead of cross entropy and softmax. We optimize the loss function in Equation (3), where each p_k denotes a negative sample, which is a post that no user in the minibatch has commented on.

$$J = -\log(\sigma(\mathbf{u}^T A\mathbf{p})) - \sum_k \log(\sigma(-\mathbf{u}^T A\mathbf{p}_k)) \tag{3}$$

We regularize the weights to prevent overfitting. Hidden representations of each user and post are regularized so that the affinity score is small enough that applying a sigmoid function does not cause a vanishing gradient.

3.4 Evaluation method

Our primary evaluation method is the mean reciprocal rank (MRR) score [2]. A true label consists of a user and a post that the user comments on. We sample a random set of posts, and rank the affinity score between the user and all the posts including the true label post, and compute the rank of the affinity score of the true label post among all post. We then take the average of the reciprocal of the ranks for all the true labels in the validation/test set.

The closer the MRR score is to 1, the more accurate is the recommendation.

4 Experiments

4.1 Dataset

Data is collected from the Stanford Reddit Data [9], which contains 132,308 reddit.com submissions. We extract all active users and posts in a subreddit community (eg. politics) in a few months for training and validation. At test time, we extract users who comment on posts in the next month.

We split the data by randomly picking edges in the bipartite graph and assigning them to training, validation and test sets. To demonstrate inductive inference, we also show our results for training on one month of reddit network data (Feb 2014), and test on the reddit network data of another month (May 2014).

4.2 Performance comparison

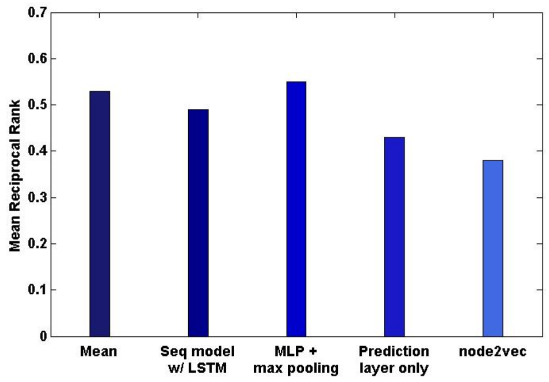


Figure 3

We use the following two baselines for performance comparison.

1. Node2Vec (n2v) is a state-of-the-art network embedding algorithm, which only uses network structure when computing an embedding for each node in a graph.
2. Another baseline uses the prediction layer explained in Section 3.3 without using the deep network architecture. This method simply takes the language features, and optimizes for the negative sampling loss by learning the bilinear weights A in Equation 3. This method uses the language feature information, but is oblivious to the network structure.

Figure 3 shows the performance of different aggregation methods. The approach of using MLP combined with max pooling achieves the best performance. LSTM does not work as well since there is no sense of sequential order between a user's neighbours. Here we only augment each aggregation instance with 3 random permutations of sampled neighbouring nodes. Increasing the number of random permutations may improve the performance, but incurs a significant runtime overhead.

We see significant performance increase by back-propagating gradient to GloVe vectors of words, but do not see significant performance different when using RNN with LSTM versus averaging the GloVe vectors.

4.3 Inductive inference

We demonstrate that our model performs well in the inductive inference scenario. Since users from the training set may not appear in the test set, network-based methods such as Node2Vec need to be trained on the new network. Language feature-based methods, however, can be transfered since weights only apply to the language features of each user and post. Our method, however, is able to transfer weights learned that captures both language feature and network structure information. In Table 4, we compare our method with a feature-based method that uses the prediction layer of GraphNet only, and the network-based state-of-the-art algorithm Node2Vec. Note that since Node2Vec is not inductive, we need to train and test on the same set of nodes.

Our result shows that our approach performs better than both baselines, demonstrating that it uses both language features and network structure information in the inductive inference scenario. Therefore in practice, we could train on a small sample of subnetworks in a social network, and be able to build high performance recommendation systems for the entire social network, which may contain billions of users and posts.

Method	MRR	MRR for testing on a different network
GraphNet	0.55	0.54
Prediction Layer only	0.50	0.46
Node2Vec		0.41

Figure 4: Performance of our method in inductive scenario.

4.4 Runtime analysis

We now analyze the runtime of our algorithm. In practice, the number of nodes sampled in a neighbourhood is constrained to be a small constant, e.g. 25. For every node in the minibatch, we sample a constant number of nodes in its minibatch. Every edge in the neighbourhood is visited a constant number of times. Therefore, the runtime of our algorithm is proportional to the number of edges in the given graph. The runtime of our algorithm is comparable to that of other popular graph embedding algorithms (see Figure 5).

We have shown that our algorithm is inductive, so it is possible to train on a sub-network and test on the entire network. Consequently, our algorithm is able to perform inference over billions of nodes in a social network very efficiently.

4.5 Visualization

We design an experiment to show what has been learned by our GraphNet in producing the representations for users and posts. We pick a user who has commented on many posts, say 100, and compute the representations for posts commented on by the user, and a random set of posts that the user does not comment on. In Figure 3, we pick a user from the test set who comments on many posts, and use t-SNE [18] to embed the hidden representations for posts that the user comments on (in red), and a random selection of posts that the user does not comment on (in blue) in 2D. As we can see, our network with sampling and aggregation captures the non-linearity in the data very well such that it makes the job of the prediction layer very easy: the positive and negative posts are almost linearly separable. In contrast, the raw language features in Figure 6 (a) are entangled.

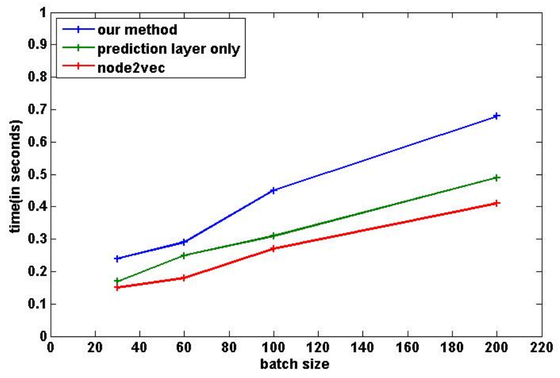
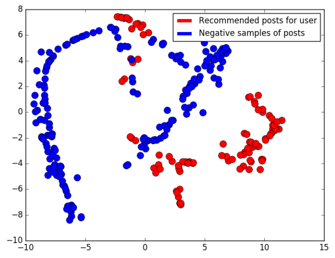
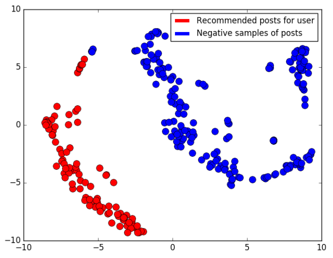


Figure 5: Runtime comparison.



(a)



(b)

Figure 6: Visualization for posts that are commented by the user (in red), and posts that are not (in blue). (a) t-SNE embedding of raw language features. (b) t-SNE embedding of representation learned by GraphNet.

5 Conclusion

In this work we develop a general framework for simultaneous learning of language features and network structures in the context of social networks. We show that various state-of-the-art methods are special cases of our general framework consisting of input feature extraction, neighbour sampling, feature aggregation and prediction. We combine both NLP and network data mining to develop a deep architecture based on this framework, that out-performs state-of-the-art methods that are based on either language features or network structure only. Our algorithm is efficient in practice for both training and inference. We also show that it performs well in the inductive inference scenario, which allows us to train the model on a small sub-network and apply it to very large social networks with billions of users and posts.

6 Contribution

All members of our group contributed to the code for this project.

In addition, Rex has worked on experimenting and analyzing different aggregation architecture, and designing the input language feature for the dataset. Xin has worked on collecting and preprocessing the network data, analyzing the architecture and designing evaluation metrics for validation and test. Yuanfang has worked on designing experiments to evaluate our results, visualizing learned representations, and running experiments for inductive inference.

Acknowledgments

Part of this project is a joint work with Will Hamilton and Jure Leskovec. We would also like to thank Danqi Chen for her advice on our project.

References

[1] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 191–198. ACM, 2016.

[2] N. Craswell. Mean reciprocal rank. In *Encyclopedia of Database Systems*, pages 1703–1703. Springer, 2009.

[3] Y. Goldberg and O. Levy. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[4] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

[5] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[6] K. Kersting, M. Mladenov, R. Garnett, and M. Grohe. Power iterated color refinement. In *AAAI*, pages 1904–1910, 2014.

[7] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[8] M. J. Kusner, Y. Sun, N. I. Kolkin, K. Q. Weinberger, et al. From word embeddings to document distances. In *ICML*, volume 15, pages 957–966, 2015.

[9] H. Lakkaraju, J. J. McAuley, and J. Leskovec. What's in a name? understanding the interplay between titles, content, and communities in social media. *ICWSM*, 1(2):3, 2013.

[10] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. In *ICML*, volume 14, pages 1188–1196, 2014.

[11] B. D. McKay and A. Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

[12] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd annual international conference on machine learning. ACM*, 2016.

[13] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[14] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[15] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[16] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[17] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. ACM, 2015.

[18] L. van der Maaten. Learning a parametric embedding by preserving local structure. *RBM*, 500(500):26, 2009.

[19] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234. ACM, 2016.

[20] Z. Yang, W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.