Deep Learning Approach to Link Weight Prediction

Yuchen Hou
School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99164
yuchen.hou@wsu.edu

Lawrence B. Holder
School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99164
holder@wsu.edu

***Abstract*—Deep learning has been successful in various domains including image recognition, speech recognition and natural language processing. However, the research on its application in graph mining is still in an early stage. Here we present Model R, a neural network model created to provide a deep learning approach to link weight prediction problem. This model extracts knowledge of nodes from known links' weights and uses this knowledge to predict unknown links' weights. We demonstrate the power of Model R through experiments and compare it with stochastic block model and its derivatives. Model R shows that deep learning can be successfully applied to link weight prediction and it outperforms stochastic block model and its derivatives by up to 73% in terms of prediction accuracy. We anticipate this new approach to provide effective solutions to more graph mining tasks.**

I. Introduction

Both academia and industry have seen pervasive adoption of deep learning techniques powered by neural network models since early 2010s, when they began to outperform other machine learning techniques in various application domains, e.g., speech recognition [1], image recognition [2], natural language processing [3], recommendation systems [4], and graph mining [5]. These neural net models can not only achieve higher prediction accuracy than traditional models, but also require much less domain knowledge and engineering.

Among those domains, graph mining is a new and active application area for deep learning. An important task in graph mining is link prediction [6] [7], i.e., link existence prediction: to predict the existence of a link. A less well-known problem is the link weight prediction: to predict the weight of a link. Link weight prediction is more informative in many scenarios. For example, when describing the connection of two users in a social network, a description "Alice texts Bob 128 times per day" is more informative than "Alice likes Bob".

We want to create a technique to predict link weights in a graph using a neural net model. The estimator should learn to represent the graph in a meaningful way and to learn to predict the target link weights using the representation it learns. The contribution of this paper is a deep learning approach to link weight prediction problem.

II. Problem

We consider the problem of link weight prediction in a weighted directed graph. We first take a look at an example of the problem, and then give the problem a definition. An undirected graph can be reduced to a directed graph by converting each weighted undirected link to two directed links with the same weight and opposite directions, so the prediction for a weighted undirected graph is a special case of the problem we consider.

A. Problem example

Let us look at an example of link weight prediction - message volume prediction in a social network, shown in Figure 1 and Table I. In this example, there are 3 users in a social network: A, B and C. Each user can send any amount of text messages to every other user. We know the message size transmitted between A and C, B and C, but not A and B. We want to predict the message size transmitted between A and B. This is a simplified network similar to many real social networks, where every user interacts with other users by posting, sharing, following or liking them. There can not be any logical approach to derive the unknown message volumes, as they have randomness. But there can be statistical approaches to build models to predict them. The ability to predict these interactions potentially allows us to recommend

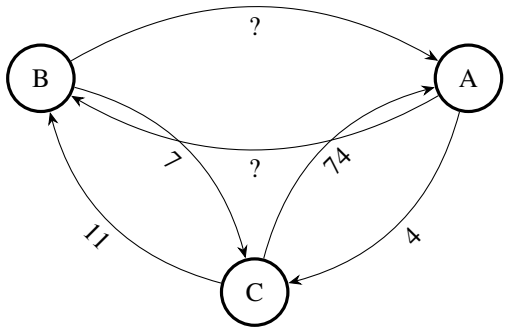

Fig. 1. An example of link weight prediction in a weighted directed graph - message volume prediction in a social network.

TABLE I
The edge list representation of the example in Figure 1.

Source node	Destination node	Link weight
A	B	?
A	C	74
B	A	?
B	C	7
C	A	4
C	B	11

new connections to users: if A is predicted/expected to send a large amount of messages to B by some model, and A is not connected to B yet, we can recommend B as a new connection to A.

B. Problem definition

Now we define the link weight prediction problem in a weighted directed graph.

- Given a weighted directed graph with the node set V and link subset E
- Build a model w = f(x, y) where x and y are nodes and w is the weight of link (x, y) that can predict the weight of any link

For every possible link (1 out of n^2, where n is the number of nodes), if we know its weight, we know it exists; if we do not know its weight, we do not know it exists. This is a very practical point when we handle streaming graphs: for any possible link, we either know it exists and know its weight (if it has been streamed in), or we do not know if the link will ever exist, nor know its weight.

III. Existing approaches

In our literature study on previous research in the link weight prediction problem, we have found few existing approaches and no deep learning ones. In this section, we review these existing approaches.

A. Node similarity model

This approach is designed for undirected graphs. It assumes the weight of a link between two nodes is proportional to the similarity of those two nodes. It employs a linear regression model [8]:

$$w_{xy} = k \cdot s_{xy}$$

where k is the regression coefficient, w_{xy} is the weight of the link between node x and node y, and s_{xy} is the similarity of x and y, calculated based on their common neighbors:

$$s_{xy} = \sum_{z \in N(x) \cap N(y)} F$$

where N(x) is the set of neighbors of node x, z is any common neighbor of x and y, and F is an index factor which has nine different forms, shown in Table II. In Table II, d_z is the degree of node z and s_z is the strength of node z:

$$s_z = \sum_{u \in N(z)} w_{zu}$$

These nine forms represent three groups of measures of 2-hop paths connecting those two nodes:

- Unweighted group [9]: this group is based on path existence and ignore path weights.
- Weighted group [10]: this group is based on path length, i.e., the sum of path weights.
- Reliable route weighted group [11]: this group is based on path reliability, i.e., the product of path weights.

And each group contains three forms:

TABLE II
9 different forms of index factor F.

	Common Neighbors	Adamic-Adar	Resource Allocation
Unweighted	1	$\frac{1}{\log(d_z)}$	$\frac{1}{d_z}$
Weighted	$w_{xz} + w_{zy}$	$\frac{w_{xz} + w_{zy}}{\log(1 + s_z)}$	$\frac{w_{xz} + w_{zy}}{s_z}$
Reliable-route Weighted	$w_{xz} \cdot w_{zy}$	$\frac{w_{xz} \cdot w_{zy}}{\log(1 + s_z)}$	$\frac{w_{xz} \cdot w_{zy}}{s_z}$

- Common Neighbors: this form is based on path and ignore node degrees.
- Adamic-Adar: this form is similar to Common Neighbors, but depresses the contribution of nodes with high degrees or high strengths.
- Resource Allocation: this form is similar to Adamic-Adar, but depresses more than Adamic-Adar does.

B. SBM (Stochastic Block Model)

This approach is designed for unweighted graphs and uses only link existence information [12]. The main idea is to partition nodes into L groups and connect groups with bundles. In this way, the graph has a 2-level structure:

- Lower level: each group consists of nodes which were topologically similar in the original graph
- Upper level: groups are connected by bundles to represent the original graph

Given a graph with adjacency matrix A, the SBM has the following parameters:

- A: link existence matrix, where $A_{ij} \in \{0, 1\}$
- z: the group vector, where $z_i \in \{1...L\}$ is the group label of node i
- θ: the bundle existence probability matrix, where $\theta_{z_i z_j}$ is the existence probability of bundle (z_i, z_j)

So the existence of link (i, j) A_{ij} is a binary random variable following the Bernoulli distribution:

$$A_{ij} \sim B(1, \theta_{z_i z_j})$$

The SBM fits parameters z and θ to maximize the probability of observation A:

$$P(A|z, \theta) = \prod_{ij} \theta_{z_i z_j}^{A_{ij}} (1 - \theta_{z_i z_j})^{1 - A_{ij}}$$

We rewrite the log likelihood of observation A as an exponential family:

$$\log(P(A|z, \theta)) = \sum_{ij} (T(A_{ij}) \eta(\theta_{z_i z_j}))$$

where

$$T(A_{ij}) = (A_{ij}, 1)$$

is the vector-valued function of sufficient statistics of the Bernoulli random variable and

$$\eta(\theta) = (\log(\frac{\theta}{1-\theta}), \log(1-\theta))$$

is the vector-valued function of natural parameters of the Bernoulli random variable.

C. pWSBM (pure Weighted Stochastic Block Model)

The pWSBM is designed for weighted graphs and uses only link weight information [13]. So it differs from SBM in a few ways described below. Here we choose model link weight with normal distribution. Adjacency matrix A becomes the link weight matrix where the weight of link (i, j) A_{ij} is a real random variable following the normal distribution:

$$A_{ij} \sim N(\mu_{z_i z_j}, \sigma^2_{z_i z_j})$$

$\theta_{z_i z_j}$ becomes the weight distribution parameter of bundle (z_i, z_j):

$$\theta_{z_i z_j} = (\mu_{z_i z_j}, \sigma^2_{z_i z_j})$$

$T(A_{ij})$ becomes the vector-valued function of sufficient statistics of the normal random variable:

$$T(A_{ij}) = (A_{ij}, A^2_{ij}, 1)$$

$\eta(\theta)$ becomes the vector-valued function of natural parameters of the normal random variable:

$$\eta(\theta) = (\frac{\mu}{\sigma^2}, -\frac{1}{2\sigma^2}, -\frac{\mu^2}{2\sigma^2})$$

The pWSBM fits parameter z and θ to maximize the log likelihood of observation A:

$$\log(P(A|z,\theta)) = \sum_{ij}(A_{ij}\frac{\mu_{z_i z_j}}{\sigma^2_{z_i z_j}} - A^2_{ij}\frac{1}{2\sigma^2_{z_i z_j}} - \frac{\mu^2_{z_i z_j}}{\sigma^2_{z_i z_j}})$$

D. bWSBM (balanced Weighted Stochastic Block Model)

The bWSBM is a hybrid of SBM and pWSBM and uses both link existence information and link weight information [13]. The hybrid log likelihood becomes:

$$\begin{aligned}&\log(P(A|z,\theta))\\ =&\alpha \sum_{ij\in E}(T_e(A_{ij})\eta_e(\theta_{z_i z_j}))\\ &+ (1-\alpha)\sum_{ij\in W}(T_w(A_{ij})\eta_w(\theta_{z_i z_j}))\end{aligned}$$

where pair (T_e, η_e) denotes the family of link existence distributions in SBM and pair (T_w, η_w) denotes the family of the link weight distributions in pWSBM. and $\alpha \in [0, 1]$ is a tuning parameter that determines their relative importance, E is the set of observed interactions, and W is the set of weighted edges. In the following, we use $\alpha = 0.5$ following the practice in [13].

E. DCWBM (Degree Corrected Weighted Stochastic Block Model)

The DCWBM is designed to incorporate node degree by replacing pair (T_e, η_e) in the bWSBM with:

$$\begin{aligned}T_e(A_{ij}) &= (A_{ij}, -d_i d_j)\\ \eta_e(\theta) &= (\log\theta, \theta)\end{aligned}$$

where d_i is the degree of node i [13].

F. Deep learning approaches to other related problems

There are a few deep learning approaches to the node classification problem (i.e., to predict whether a node has a label) and link existence prediction problem (i.e., to predict whether a link exists). Two earlier approaches are graph neural nets and relational neural nets [14], where neural nets are incorporated into a traditional iterative graph information propagation approach. A newer approach is deep walk [15], where a graph is reduced to a natural language corpus so that existing neural net model designed for natural language processing - skip-gram model - can handle the graph. A very recent approach is node2vec [5], which uses the same skip-gram model, but a different way to reduce the graph to a natural language corpus.

IV. Observations

As deep learning techniques become more powerful and standardized, a key process of a domain-specific deep learning application is to represent entities as vectors, because a neural net needs vectors as inputs. In this section, we make some observations about this process, which lead to the deep learning approach to link weight prediction.

A. Entities and representations

First of all, we summarize how a neural net represents various types of entities in different domains with different relations, as shown in Table III. An image in image recognition is represented as a 2D light amplitude array with dimensions height and width; an audio/spectrogram in speech recognition is represented as a 2D sound amplitude array with dimensions time and frequency. The relation between two images or two audio is not commonly used. Words in natural languages, items in recommendation systems, and nodes in graphs can be represented by vectors (1D numeric arrays). The relations between two words, two items and two nodes are commonly used to learn these vectors. It is clear that representations for all the entities are numeric arrays, because neural nets rely on neurons' activations and communications, which are both numeric.

B. Mapping entities to vectors

The word2vec technique is famous for using a neural net to learn to map every entity (word in this case) in a vocabulary to a vector without any domain knowledge [16]. The subsequent techniques item2vec [4] and node2vec [5] use the same skip-gram model used in word2vec to map items and nodes to vectors. In a corpus, every word is described/defined only by

TABLE III
A SUMMARY OF VARIOUS TYPES OF ENTITIES, THEIR NUMERIC REPRESENTATIONS AND INTER-ENTITY RELATIONS IN DIFFERENT DOMAINS.

Domain	Entity	Relations	Representation
image recognition	image	N/A	2D light amplitude array[width, height]
speech recognition	audio/spectrogram	N/A	2D sound amplitude array[time, frequency]
natural language processing	word	co-occurrences of words in a context	1D array (i.e., word vector)
recommendation systems	item	co-purchases of items in a order	1D array (i.e., item vector)
graph mining	node	connections of nods (i.e., links)	1D array (i.e., node vector)

related words in its contexts, by implicit relations between words in word co-occurrences. Nonetheless, the neural net can learn from word co-occurrences and map words to vectors accordingly, such that the relations between words are preserved in the word vector space [17]. The same arguments apply to relations between items and between nodes.

C. Mapping nodes to vectors

The relation between nodes is quite different from that between words:

- The weight of a link from one node to another specifically tells us how strong one node connects to the other; this type of relation has regular and explicit form: entity - relation - entity.
- On the other hand, the co-occurrences of words (e.g., in the context [The quick brown fox jumps over the lazy dog]) implicitly tell us these words are related but do not tell us any specific relations (e.g., Are [quick] and [brown] related? What is the relation between [fox] and [jumps]? Is [over] a relation or entity?).

Natural languages do not have the notion that all information can be described as entities and their relations, as in graphs. For a neural net, words can simply show up in sequences from day-to-day conversations, in many flexible and unpredictable ways, with little structure or regularity. Therefore, the skip-gram model, designed to handle natural languages, can not take advantage of highly structured data in graphs. This suggests that a neural net, if correctly designed to handle graphs, should be able to learn a node-to-vector mapping supervised by the link weight, in a more specific, direct and simply way than it learns word-to-vector mappings supervised by word co-occurrences.

V. APPROACH

Following the above observations, we build an estimator with a neural net model using a node pair as its input and the weight of the link connecting the nodes as its output.

A. Model R

We design the model in the estimator as a fully connected neural net which we call Model R (R as in relation), shown in Figure 2. We have considered a convolutional neural net as an alternative, but we decided it would not be a good fit for this application. These node vectors do not have any spacial property for a convolutional neural network to take advantage of, compared to the 2D array of an image where the spacial location of each pixel has significant meaning (e.g., relative distances of pixels). These node vectors do not have any of the invariance properties of an image either, such as translation invariance, rotation invariance, size invariance and illumination invariance.

The model contains the following layers:

- A mapping layer that maps node IDs to node vectors. In the node dictionary, the key is the node ID, the value is the node vector, which is learned through back propagation and stochastic gradient descent.
- An input layer directly activated by the node dictionaries. Its activations are the two node vectors.
- Multiple fully connected hidden layers of rectified linear units (only two layers are shown in the figure). These units employ the rectifier ($f(x) = \max(0, x)$) as their activation function. These layers learn to extract more and more abstract weight-relevant information.
- An output layer with a linear regression unit. This unit employs linear regression ($f(x) = kx$) as its activation function. It learns to predict the link weight as a real number using abstracted weight-relevant information.

Link weights provide the information about nodes. We fully take this property into account and design this model to learn complex and unobservable node information (i.e., node vectors) supervised by a simple and observable relations between nodes (i.e., link weight).

B. Learning techniques

The estimator uses the above model and a number of popular deep learning techniques:

- Backpropagation: propagation of the error gradients from output layer back to each earlier layer [18]
- Stochastic gradient descent: the optimization that minimizes the error (descending against the error gradient in weight space) for a random sample in each gradient descent step [19]
- Mini-batch: the modification to stochastic gradient descent to accelerate and smooth the descent by minimizing the error for a small random batch of samples in each gradient descent step [20]
- Early stopping: the regularization used to reduce overfitting during the iterative learning process by stopping the learning when validation error stops decreasing [21]

C. Model R design parameters

Given the above design and learning techniques, there are still a number of design parameters we can adjust, e.g., the number of layers, the number of units in each layer,

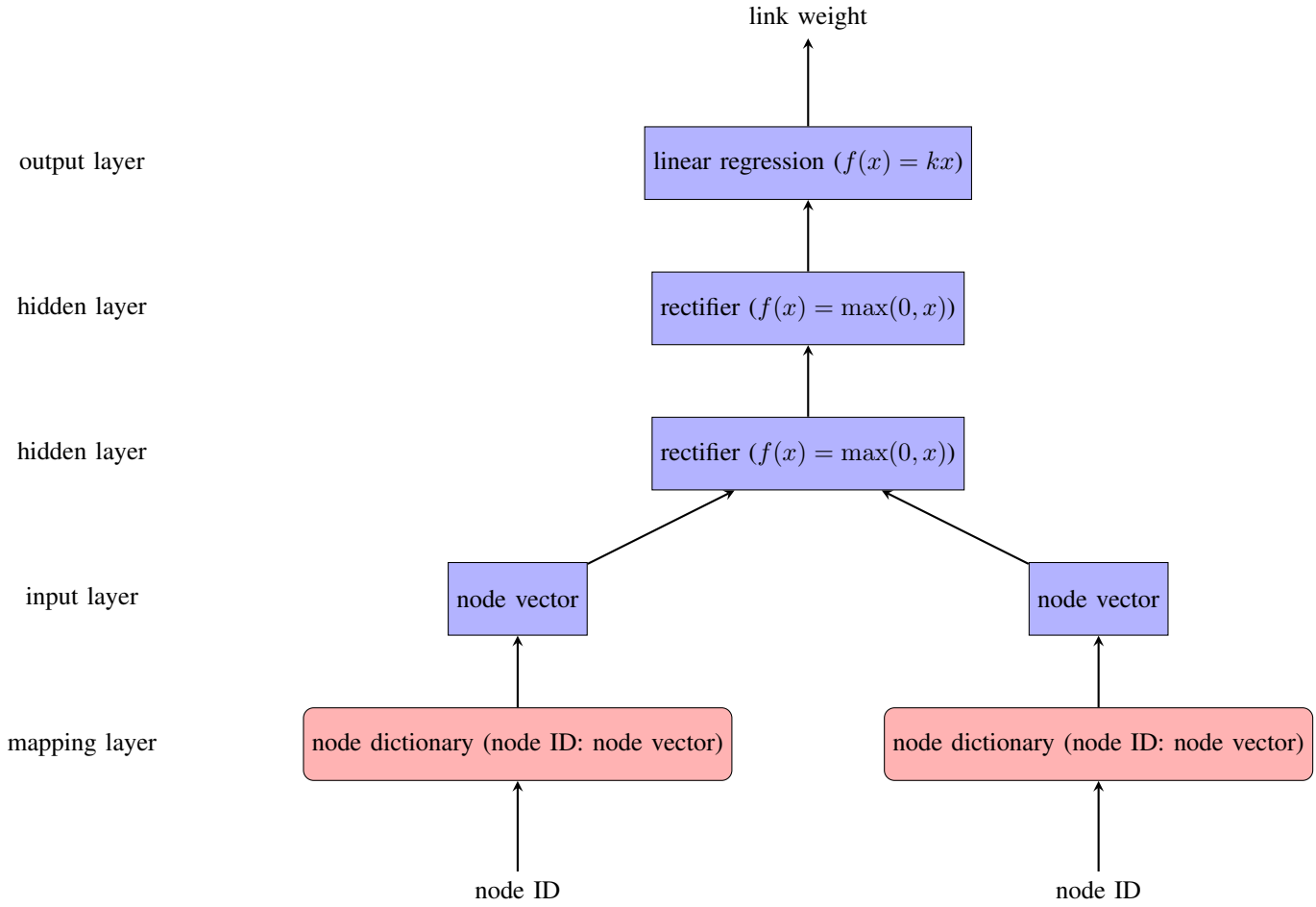

Fig. 2. Model R for a weighted graph. Only layers and their connections are shown, while the units in each layer and their connections are not shown.

and the activation function of each unit. Now we briefly discuss different options for these parameters, and also some justifications for our choices.

- The choice of rectifier as the activation function is a relatively easy one. Compared to earlier popular activation functions like sigmoid function ($f(x) = (1 + \exp(-x))^{-1}$), rectifier not only simplifies and accelerates computation, but also eliminates vanishing gradient problems, and has become the most popular activation function for deep neural networks [22].
- The choice of layer size is related to the number of examples in the dataset. Naturally, the larger the dataset is, the more discriminative the model should be, and consequently higher degrees of freedom, higher dimensions of vectors and larger layer sizes. Empirically, we usually set the layer size as a logarithm function of the dataset size:

  $$d = log_2(n)$$

  where d (as in dimension) is the layer size and n is the dataset size.
- The choice of number of layers is related to the complexity of the relation between the input and the output of the model. As a trivial example, if the input and the output have linear relation, no hidden layer is necessary and the model is simply a linear model. If the input and the out put have non-linear relation, the more complex the relation is, the more number of layers are necessary. Empirically, we usually set the layer size to 4, as a good compromise of learning speed and prediction accuracy.

We naturally assume the most optimum design parameters are dataset dependent. However, we do not know any theoretical way to calculate the most optimum parameters based on the statistic signatures of a specific dataset. Therefore, in this work, we evaluate different parameter choices through a few experiments. We will work on design parameter optimization in our future research.

VI. Experiments

We evaluate Model R experimentally with SBM, pWSBM, bWSBM and DCWBM as baselines, and compare their prediction errors on several datasets. We use the same datasets and experiment process used in a recent study of these baselines [13]. The results show that Model R can achieve much lower prediction error than the baseline models.

A. Datasets

The experiments use four datasets summarized in Table IV.

B. Experiment process

We do the same experiment for each dataset. All the link weights are normalized to the range [-1, 1] after applying a logarithm function. Each experiment consists of 25 independent trials. In each trial, we split the dataset randomly into 3 subsets:

- 70% into training set
- 10% into validation set
- 20% into testing set

We use mean squared error as the prediction accuracy metric. For each experiment, we report the mean and standard deviation of the errors from 25 trials. The pseudo code of the experiment process is shown in Figure 3.

C. Experiment results

In our experiments, Model R's error is lower than every other model on every dataset, shown in Figure 4 and Table V. In this section we compare Model R with the baseline models on every dataset. Given the dataset, we regard ModelRError (as well as BaselineError) as a random variable so each trial generates an example of it. We can do a t-test to justify the significance of difference between the means of variables ModelRError and BaselineError. The mean of a variable is not the same as the mean of a sample of the variable. More specifically, a variable can generate two samples with different sample means, therefore two samples with different means do not imply the two variables generating them have different means. For each dataset, we do a t-test for the two variables where the null hypothesis is that the two variables have the same mean:

$$\overline{X_1} == \overline{X_2}$$

where X_1 and X_2 are ModelRError and BaselineError and where $\overline{X}$ is the mean of variable X. Welch's t-test defines its p value as the Student's t-distribution cumulative density function:

$$p = 2 \int_{-\infty}^{-|t|} f(x)dx$$

The smaller p is, the more confidently we can reject the null hypothesis, i.e., accept that:

$$\overline{ModelRError} \neq \overline{BaselineError}$$

Typically there is a domain specific threshold for p, e.g., 0.1 or 0.01. If p is smaller than the threshold we reject the null hypothesis. We calculate the p value and also error reduction from baseline to Model R as:

$$Reduction = \frac{BaselineError - ModelRError}{BaselineError}$$

The p value is almost 0 for all datasets and error reduction is significant, shown in Table V. Model R has lower error than every other model on every dataset, reducing error by 25% to 73% from the best baseline model - pWSBM. The number in every parenthesis is the standard deviation of the errors in 25 trials in the last digit. The very low p values strongly indicate the error reduction is significant. These results show that Model R outperforms pWSBM on all these datasets.

D. Model robustness

In our experiments, we have not observed any significant (more than 5%) prediction error increase or decrease when we change parameters around the values we typically choose. Overall, Model R has demonstrated a very high level of model robustness.

E. Reproducibility

In order to ensure the reproducibility of the experiment, we specify the implementation details in this section:

- Programming language: Python 3
- Python implementation: CPython 3.5
- Deep learning package: TensorFlow [27]
- Operating system: Ubuntu 16.10 64-bit
- Computer make and model: Lenovo ThinkCentre M83
- Memory: 16 GB
- Processor: Intel Core i7-4770 CPU @ 3.40GHz $\times$ 8

The program uses all 8 threads of the processor. Each experiment takes about one hour to finish, depending on the dataset and parameters in the learning algorithm. The program is open-source under MIT license hosted on Github [1] so that anyone can use it without any restriction.

VII. Conclusion

Model R shows that deep learning can be successfully applied to the link weight prediction problem. It effectively learns complex and unobservable node information (i.e., node vectors) from simple and observable relations between nodes (i.e., link weights), and uses that information to predict unknown link weights. We anticipate this new approach will provide effective solutions to more graph mining tasks.

Compared to SBM based approaches, Model R is much more accurate. A few possible reasons are:

- Higher level of discrimination for nodes: SBM based approaches do not differentiate nodes within the same group, and assume the weights of all links connecting two nodes from two groups follow the same distribution. Model R does not assume that, but gives every node a unique description - the node vector - so that it can have a more accurate description for every single node.
- Higher level of model flexibility: SBM based approaches assume the weight of every link follows a normal distribution. Model R does not assume that, but takes advantage of high flexibility of layers of non-linear neural network units, so that it can model very complex weight distributions.

[1] https://github.com/yuchenhou/elephant

TABLE IV
THE DATASETS USED IN EXPERIMENTS.

Dataset name	Node count	Node type	Link count	Link weight type
Airport[23]	500	busiest airports in US	5960	number of passengers traveling from one airport to the other
Collaboration[24]	226	nations on Earth	20616	number of papers written by authors from the two nations
Congress[25]	163	102nd US Congress committees	26569	interlock value of shared members from the two committees
Forum[26]	1899	users of a student social network	20291	number of messages sent from one student to the other

```
def main():
    for dataset in [Airport, Collaboration, Congress, Forum]:
        (error_mean, error_standard_deviation) = do_experiment(dataset)
def do_experiment(dataset):
    errors = list()
    for trial in range(25):
        testing_error = append(evaluate_model_on(dataset))
        errors.append(testing_error)
    return (errors.mean(), errors.standard_deviation())
def evaluate_model_on(dataset):
    (training_set, validation_set, testing_set) = split(dataset)
    while validation_error decreases:
        training_error = estimator.learn(training_set)
        validation_error = estimator.predict(validation_set)
    testing_error = estimator.predict(testing_set)
    return testing_error
```

Fig. 3. The pseudo code of the experiment process. The estimator.learn() method learns for one epoch through the training set, updating the weights after each example batch.

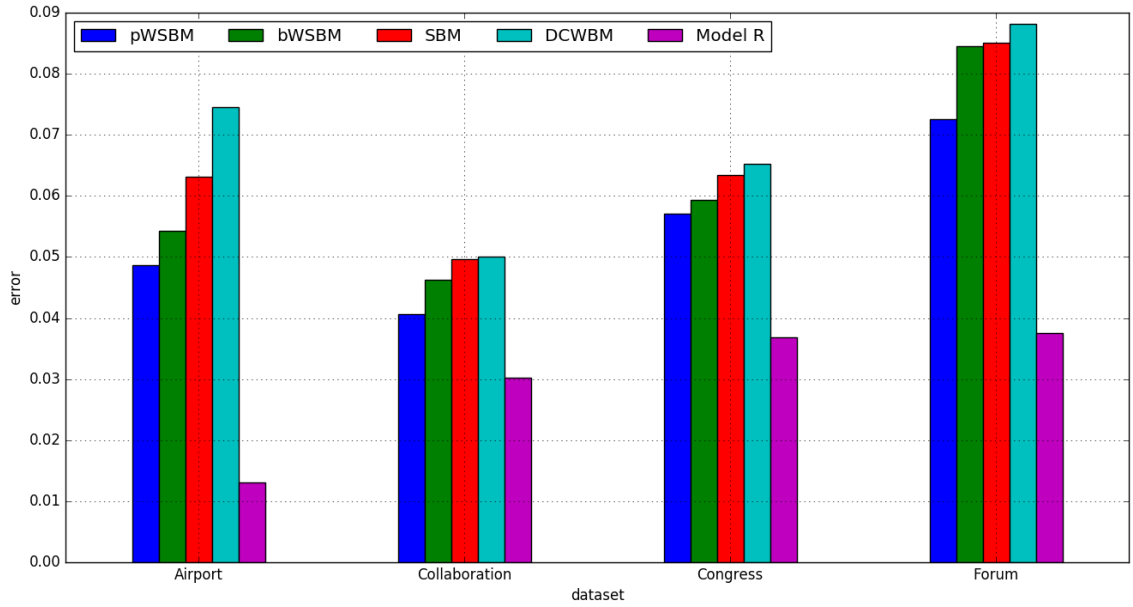

Fig. 4. The mean squared errors of 5 models on 4 datasets: Model R has lower error than every other model on every dataset. Every error value shown here is the mean errors for the 25 trials in the experiment.

VIII. FUTURE WORK

There are three aspects we would like to study in our future work on this model.

The first aspect is the mapping layer of Model R. In the current model, the mapping layer contains two independent

TABLE V
THE MEAN SQUARED ERRORS WITH STANDARD DEVIATIONS OF 5 MODELS ON 4 DATASETS.

Dataset	pWSBM	bWSBM	SBM	DCWBM	Model R	Reduction	p
Airport	0.0486 ± 0.0006	0.0543 ± 0.0005	0.0632 ± 0.0008	0.0746 ± 0.0009	0.013 ± 0.001	73%	4.2e-66
Collaboration	0.0407 ± 0.0001	0.0462 ± 0.0001	0.0497 ± 0.0003	0.0500 ± 0.0002	0.030 ± 0.001	25%	9.1e-44
Congress	0.0571 ± 0.0004	0.0594 ± 0.0004	0.0634 ± 0.0006	0.0653 ± 0.0004	0.036 ± 0.003	35%	7.1e-35
Forum	0.0726 ± 0.0003	0.0845 ± 0.0003	0.0851 ± 0.0004	0.0882 ± 0.0004	0.037 ± 0.001	48%	4.2e-68

node dictionaries to learn the node vectors. During the learning stage, these two dictionaries learn to map the nodes to two spaces:

- The source node dictionary maps nodes to source node space.
- The destination node dictionary maps nodes to destination node space.

A more intuitive approach is to have every node mapped to one and only one node vector in both dictionaries, as in the word2vec technique. However, will this more intuitive approach actually perform better? Or does it depend on whether the graph is undirected or directed? These are interesting questions we can study in the future.

The second aspect is to handle more complex graphs, especially for social network applications. In particular, we want to handle graphs with node attributes. Users can have labels like "nationality", and real attributes like "age" . One feasible approach is to append one unit for each of these attributes to the node vector.

The third aspect is to handle large dynamic graphs. A social network can have a large volume of links collected continuously by a distributed system. A potential approach is to deploy an estimator to each computing node of the distributed system and analyze a link stream there, and let these estimators exchange their knowledge periodically.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1646640.

REFERENCES

[1] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] K. Yao, G. Zweig, M.-Y. Hwang, Y. Shi, and D. Yu, "Recurrent neural networks for language understanding." in *INTERSPEECH*, 2013, pp. 2524–2528.

[4] O. Barkan and N. Koenigstein, "Item2vec: Neural item embedding for collaborative filtering," *arXiv preprint arXiv:1603.04259*, 2016.

[5] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016.

[6] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[7] M. Al Hasan, V. Chaoji, S. Salem, and M. Zaki, "Link prediction using supervised learning," in *SDM06: workshop on link analysis, counter-terrorism and security*, 2006.

[8] J. Zhao, L. Miao, J. Yang, H. Fang, Q.-M. Zhang, M. Nie, P. Holme, and T. Zhou, "Prediction of links and weights in networks by reliable routes," *Scientific reports*, vol. 5, 2015.

[9] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social networks*, vol. 25, no. 3, pp. 211–230, 2003.

[10] T. Murata and S. Moriyasu, "Link prediction of social networks based on weighted proximity measures," in *Web Intelligence, IEEE/WIC/ACM international conference on*. IEEE, 2007, pp. 85–88.

[11] H. A. Taha, *Operations Research: An Introduction (For VTU)*. Pearson Education India, 1982.

[12] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social networks*, vol. 5, no. 2, pp. 109–137, 1983.

[13] C. Aicher, A. Z. Jacobs, and A. Clauset, "Learning latent block structure in weighted networks," *Journal of Complex Networks*, p. cnu026, 2014.

[14] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[15] 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*. ACM, 2014, pp. 701–710.

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[17] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[19] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[20] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *The Journal of Machine Learning Research*, vol. 11, pp. 19–60, 2010.

[21] S. Smale and D.-X. Zhou, "Learning theory estimates via integral operators and their approximations," *Constructive approximation*, vol. 26, no. 2, pp. 153–172, 2007.

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[23] V. Colizza, R. Pastor-Satorras, and A. Vespignani, "Reaction–diffusion processes and metapopulation models in heterogeneous networks," *Nature Physics*, vol. 3, no. 4, pp. 276–282, 2007.

[24] R. K. Pan, K. Kaski, and S. Fortunato, "World citation and collaboration networks: uncovering the role of geography in science," *arXiv preprint arXiv:1209.0781*, 2012.

[25] M. A. Porter, P. J. Mucha, M. E. Newman, and C. M. Warmbrand, "A network analysis of committees in the us house of representatives," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 20, pp. 7057–7062, 2005.

[26] T. Opsahl and P. Panzarasa, "Clustering in weighted networks," *Social networks*, vol. 31, no. 2, pp. 155–163, 2009.

[27] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.