sciendo

JAISCR, 2019, Vol. 9, No. 1, pp. 21 – 40
10.2478/jaiscr-2018-0022

ON GRAPH MINING WITH DEEP LEARNING: INTRODUCING MODEL R FOR LINK WEIGHT PREDICTION

Yuchen Hou, Lawrence B. Holder

School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99164 USA

E-mail: yuchen.hou@wsu.edu, holder@wsu.edu

Submitted: 20th October 2017; Accepted: 13th November 2017

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 the link weight prediction problem. This model uses a node embedding technique that extracts node embeddings (knowledge of nodes) from the known links' weights (relations between nodes) and uses this knowledge to predict the unknown links' weights. We demonstrate the power of Model R through experiments and compare it with the 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 analyze the node embeddings to confirm that closeness in embedding space correlates with stronger relationships as measured by the link weight. We anticipate this new approach will provide effective solutions to more graph mining tasks.

Keywords: Deep learning, Neural networks, Machine learning, Graph mining, Link weight prediction, Predictive models, Node embeddings.

1 Introduction

Both science and industry have seen pervasive adoption of deep learning techniques powered by neural network models since the 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 cannot 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 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 texts 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 learn to predict the target link weights using the representation it learns.

The contribution of this paper is a first deep learning approach to the link weight prediction problem. We introduce Model R - the first deep neural network model specifically designed to solve the link weight prediction problem. We systematically study Model R's node embedding technique and illustrate its uniqueness compared to other embedding techniques. We also show that Model R significantly outperforms the state of the art non-deep learning approach to the link weight prediction problem - the stochastic block model.

The rest of the paper includes the following Sections:

- Problem: a description of the link weight prediction problem, including a social network message volume prediction example and a formal definition.
- Existing approaches: a review of the state of the art approaches to the link weight prediction problem, including Node Similarity Model, Stochastic Block Model and models derived from Stochastic Block Models.
- Deep learning and embeddings: a review of latest deep learning embedding techniques including content based techniques and relation based techniques.
- Approach: an introduction to Model R, including its neural network architecture, node embedding technique, deep learning techniques, design parameters and choices.
- Experiments: an experimental evaluation of the performance of Model R, with the comparison to 4 baseline approaches on 4 datasets.
- Node embedding analysis: an analysis of the node embeddings Model R produces to confirm that closeness of nodes in embedding space correlates with a strength of node relations.
- Conclusion: Model R outperforming Stochastic Block Model shows deep learning can be applied to the link weight prediction problem and achieve better performance than the state of the art non-deep learning approaches.
- Future work: a brief discussion of possible future directions of this work.

2 Problem

We consider the problem of link weight prediction in a weighted directed graph. We first show an example of the problem, and then give the problem 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.

2.1 Problem example

Let us look at an example of link weight prediction, message volume prediction in a social network, shown in Figure 1. 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 number of messages transmitted between A and C, B and C, but not A and B. We want to predict the number of messages transmitted between A and B.

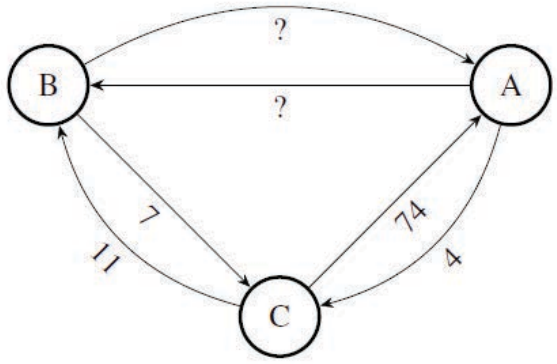

Figure 1. An example of link weight prediction in a weighted directed graph - message volume prediction in a social network.

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 new connections to users: if A is predicted/expected to send a large number 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.

2.2 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 if 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.

3 Existing approaches

In our literature study on previous research in the link weight prediction problem, we have found some existing approaches, but none use deep learning. In this Section, we review these existing approaches.

3.1 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 1.

In Table 1, 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:

- Common Neighbors: this form is based on paths and ignores 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.

3.2 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 $A_{ij} \in \{0, 1\}$

Table 1. 9 different forms of index factor F.

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

– 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.

3.3 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 a 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

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

$\eta(\theta)$ 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

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

3.4 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

$$\log(P(A|z,\theta)) =$$

$$\alpha \sum_{ij \in E} (T_e(A_{ij}) \eta_e(\theta_{z_i z_j})) + (1-\alpha) \sum_{ij \in W} (T_w(A_{ij}) \eta_w(\theta_{z_i z_j})),$$

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].

3.5 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

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

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

4 Deep Learning and Embeddings

As deep learning techniques become more powerful and standardized, a key process of a domain-specific deep learning application is converting entities to points in an embedding space, or equivalently, mapping entities to vectors in a vector space, because a neural net needs vectors as inputs. These vectors are called embeddings and this process is called embedding. Embeddings are ubiquitous in deep learning, appearing in natural language processing (embeddings for words), recommender systems (embeddings for users and items), graph mining (embeddings for nodes) and other applications. In this Section, we review a few classical embedding techniques and models for images, audio, words, documents, items, and nodes. A common goal of these techniques is to ensure that similar entities are close to each other in the embedding space. Observations about this process lead to the deep learning approach to link weight prediction.

4.1 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 2. 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.

4.2 Mapping entities to vectors

The word2vec technique in natural language processing 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 [14]. In a corpus, every word is described/defined only by 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. It provides strong evidence that word embedding with the Skip-gram model can extract knowledge about words from the relations between words and represent this knowledge in the word embedding space [15]. In fact, most subsequent embedding techniques in other domains use the same Skip-gram model, such as doc2vec [16], item2vec [4], node2vec [5] and deep walk [17]. All these techniques have achieved high prediction accuracies in their various applications including language modeling, document classification, item rating prediction, and node classification.

4.3 Content-based embedding techniques

Techniques in this group extract knowledge about an entity from its content, i.e., the input of the neural network is a vector produced from the item's content. For example, the content can be the pixel values of an image, or the spectrogram of an utterance. The similarity of these techniques is that the content (the raw input to the neural network) is already a vector. Therefore, the embedding process is practically a dimensionality reduction process that converts a high dimensional raw input vector to a low dimensional vector containing more ab-

Table 2. 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)

stract knowledge about the input entity.

4.3.1 Image embedding with auto-encoders

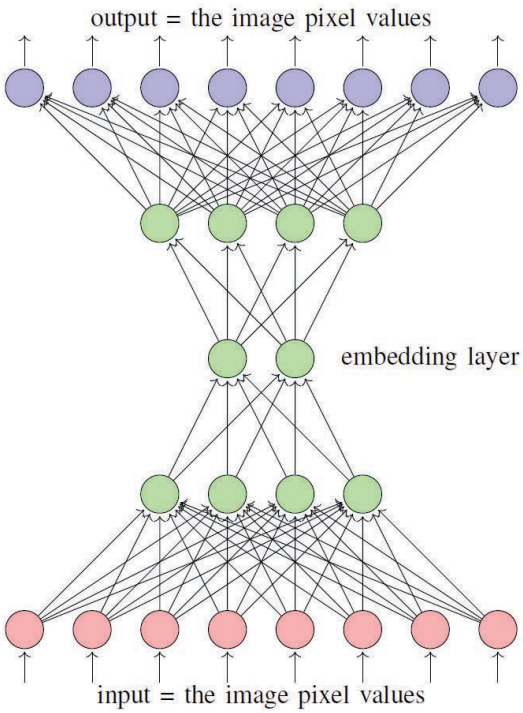

Figure 2. A small auto-encoder neural network model with 1 input layer (red), 3 hidden layers (green), and 1 output layer (blue), input size 8, embedding size 2. Notice that the embedding layer is the innermost hidden layer. The hidden layers use rectified linear units. The output layer uses linear units.

This is an unsupervised embedding technique commonly used in image recognition [18]. A small auto-encoder neural network model is shown in Figure 2. The model is a feed-forward neural network. The output layer and the input layer have the same size. The hidden layers closer to the input or output layers have larger sizes. This technique is unique because during training, the input activation and the expected output activation are always the same vector of pixel values of the image. From the input layer to the embedding layer, the layer size decreases, compressing the information. It can effectively reduce a high dimensional vector (the activation of a large number of input units with raw pixel values) to a low dimensional vector (the activations of a small number of hidden units with abstracted meanings) [19]. This technique applies to not only images for image recognition, but also audio spectrogram for speech recognition [20] and words for natural language processing [21].

4.3.2 Audio embedding with convolutional neural network

This is a supervised deep learning technique commonly used in speech recognition [22]. A small convolutional neural network model is shown in Figure 3.

The model is a feed-forward neural network. The input activation is the vector of pixel values of the audio spectrogram. The output layer uses softmax units to predict the target label, such as the genre of a song [23] or the word of an utterance [22]. From the input layer upward, each convolutional and pooling layer combo extracts more abstract information than the previous layer. Eventually, the neural network converts the raw input data to an embedding at the fully connected layer and uses it to predict the target label. Most of the stud-

ies on convolutional neural networks focus on accurately predicting the target attributes, and the concept of entity embedding is under-explored. This technique applies to not only audio spectrogram for speech recognition, but also images for image recognition [24], letter trigram for natural language processing [25] and items for recommender systems [26].

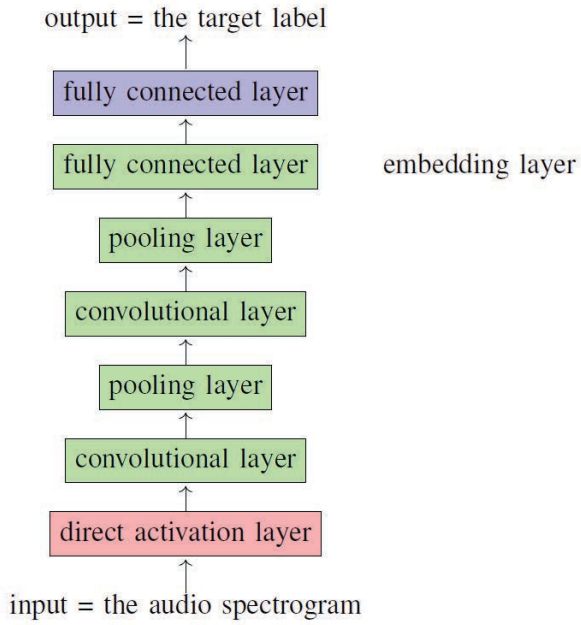

Figure 3. A small convolutional neural network model with 1 input layer (red), 5 hidden layers (green), and 1 output layer (blue). Notice that the embedding layer is the last hidden layer. The hidden layers use rectified linear units. The output layer uses softmax units. Only layers and the connections between layers are shown, while the units in each layer and the connections between units are not shown.

4.4 Relation-based embedding techniques

Techniques in this group extract knowledge about an entity from its relations with other entities, such as words, users, items, and nodes. The input of the neural network is the one-hot encoding vector of an entity. An example of this encoding is shown in Table 3.

The similarity of these techniques is that each entity does not contain any information about itself and therefore its one-hot encoding vector is also a meaningless vector. In other words, each entity is only defined by its relations with other entities. Therefore, in the embedding process, the neural network gradually forms an understanding of the meaning of all entities by observing the relations between all entities.

Table 3. One hot encoding example for a dictionary of words.

Word	One-hot encoding
w_1	[1, 0, 0, 0, ... 0]
w_2	[0, 1, 0, 0, ... 0]
w_3	[0, 0, 1, 0, ... 0]
w_4	[0, 0, 0, 1, ... 0]
...	...

4.4.1 Word embedding with skip-gram model

This is an unsupervised embedding technique commonly used in natural language processing [15]. A small skip-gram neural network model is shown in Figure 4.

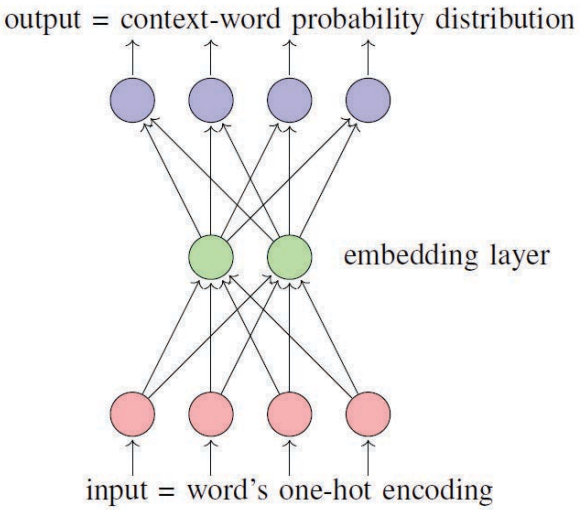

Figure 4. A small skip-gram neural network model with 1 input layer (red), 1 hidden layer (green), and 1 output layer (blue), vocabulary size 4 and embedding size 2. Notice that the embedding layer is the hidden layer. The hidden layer uses linear units. The output layer uses softmax units.

The model is a feed-forward neural network. The definition of context is the set of words close to the given word. For example, given the natural language vocabulary {the, quick, brown, fox, jumps, over, lazy, dog}, the sentence "the quick brown fox jumps over the lazy dog", a context radius of 2, and the word "fox", we have the context of fox {quick,

brown, jumps, over}. A natural language corpus has many sentences, therefore, from these sentences we can produce a dataset where each example is a (word, context-word) pair, as shown in Table 4.

Table 4. The words dataset for a natural language corpus.

Input = word	Output = context-word
...	...
brown	fox
brown	jumps
fox	quick
fox	brown
fox	jumps
fox	over
jumps	brown
jumps	fox
...	...

Given a word fox, its context word is a random variable with a probability distribution

$$P(context\_word = x | given\_word = fox),$$

where

$$x \in vocabulary$$

and the context-word probability distribution sums to 1 over the vocabulary

$$\sum_{x \in vocabulary} P(context\_word = x | given\_word = fox) = 1. \quad (1)$$

During each training step, one training example - a (word, context-word) pair - is used. The input layer is activated by the one-hot encoding of the given word. The output layer is expected to predict the one-hot encoding of the context-word. However, as each word can have many possible context-words, there is always a difference between the expected output and the actual output. After substantial training, a skip-gram neural network model will eventually output the context-word probability distribution. The embeddings for all words are technically the weights in the embedding layer. For any natural language corpus and any two words X and Y in this corpus, we have the following equivalent statements:

– X and Y have the similar meanings

– X and Y have the similar context-word probability distributions

– X and Y have the similar embeddings

The final outcome is similar words have similar embeddings. Acquiring these embeddings is often the first step in many natural language processing tasks such as paraphrasing detection [27], constituency parsing [28], sentiment analysis [29], and information retrieval [30].

4.4.2 Item embedding with skip-gram model

This is an embedding technique similar to word embedding, commonly used in recommender systems [4]. This technique reduces the item embedding problem to the word embedding problem and then applies the word embedding technique. For example, given a purchase order {monitor, keyboard, mouse, printer, scanner}, we have the context of mouse {monitor, keyboard, printer, scanner}. An e-commerce platform has many purchase orders, which can produce a dataset where each example is an (item, context-item) pair as shown in Table 5.

Table 5. The items dataset for a collection of orders.

Input = item	Output = context-item
...	...
keyboard	mouse
keyboard	printer
mouse	monitor
mouse	keyboard
mouse	printer
mouse	scanner
printer	keyboard
printer	mouse
...	...

By reducing purchase orders to natural language sentences and items to words, this technique reduces the item embedding problem to the word embedding problem. Applying the word embedding technique will produce the desired item embeddings. The final outcome is similar items have similar embeddings.

4.4.3 Node embedding with skip-gram model

This is an embedding technique similar to item embedding, commonly used in graph mining [17, 5]. This technique reduces the node embedding problem to the word embedding problem and then

applies the word embedding technique. For example, given a walk in a social network of users {John, Mary, James, Alice, Bob}, we have the context of James {John, Mary, Alice, Bob}. A graph has many walks, which can produce a dataset where each example is a (node, context-node) pair. By reducing walks to natural language sentences and nodes to words, this technique reduces the node embedding problem to the word embedding problem. The final outcome is similar nodes have similar embeddings.

4.5 The weakness of skip-gram model in node embedding

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.

5 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. Given a weighted graph, from its adjacency list we can produce a dataset where each example is a (source node, destination node, link weight) triplet. For example, given a social network where nodes are users and link weights, are numbers of messages users send to other users, we have its adjacency list dataset as shown in Table 6.

Table 6. The adjacency list dataset for a social network.

Input = (source, destination)	Output = weight
...	...
(Mary, John)	8645
(John, Mary)	9346
(John, Alice)	2357
(John, Bob)	9753
(Alic, Bob)	1238
...	...

5.1 Model R

We design the model in the estimator as a fully connected neural network model which we call Model R (R as in relation), shown in Figure 5. We have considered a convolutional neural net as an alternative, but we decided it would not be a good fit for this application. The reason is that 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. In this Section, we describe the architecture of Model R and the node embedding technique based on this model.

The model contains the following layers:

- An input layer directly activated by the one-hot encodings of a (source node, destination node) pair.
- A hidden embedding layer of linear units. This layer maps each node from its one-hot encoding to the corresponding node vector.
- Multiple fully connected hidden layers of rectified linear units (only two layers are shown in the figure). These units employ the rectifier

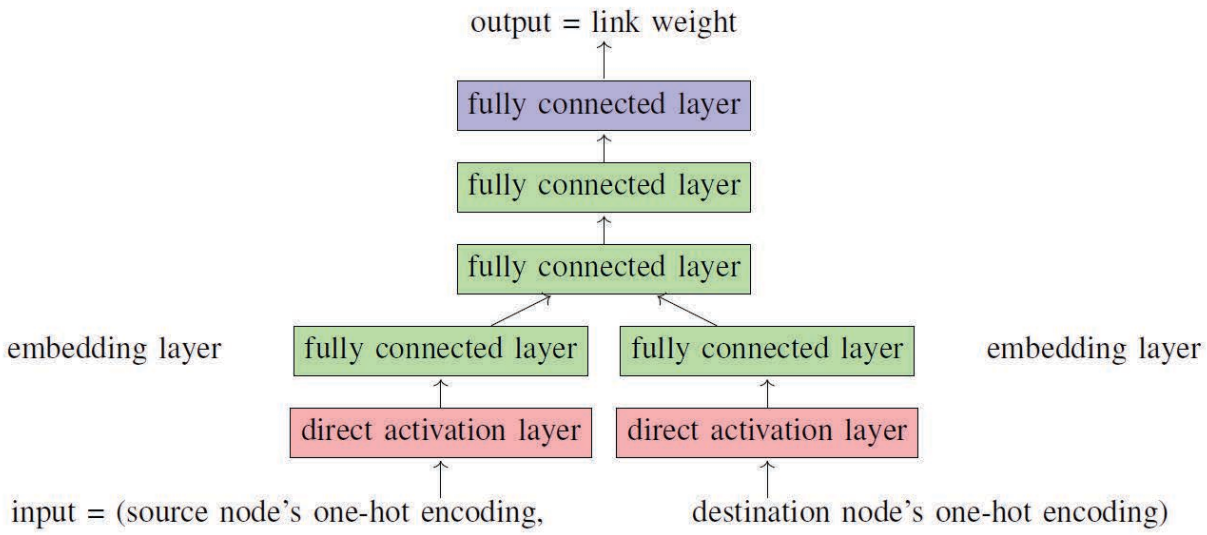

Figure 5. A simplified version of Model R with 1 input layer (red), 3 hidden layers (green), and 1 output layer (blue). Notice that the embedding layer is the first hidden layer. The embedding layer and the input layer each has two channels: one channel for the source node and one channel for the destination node. The embedding layer uses linear units while other hidden layers use rectified linear units. The output layer uses linear units. Only layers and their connections are shown, while the units in each layer and their connections are not shown.

($f(x) = \max(0,x)$) 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.

5.2 Model R node embedding technique

The Model R based node embedding technique is different from the skip-gram based techniques. One advantage of the Model R based node embedding technique is that it takes advantage of the highly organized, regular and repeated structure in the relational dataset representing a graph, i.e., a source node connects to a destination node through one and only one weighted link. The skip-gram model does not exploit this structure in natural language processing because this structure does not exist. 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 relation between nodes (i.e., link weight).

5.3 Model R 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 [31]

– 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 [32]

– 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 [33]

– Early stopping: the regularization used to reduce over-fitting during the iterative learning process by stopping the learning when validation error stops decreasing [34]

5.4 Model R design parameters and choices

Now we briefly discuss different options for our design parameters and choices, 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 [35].

– 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 hidden 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 a linear relation, no hidden layer is necessary and the model is simply a linear model. If the input and the output have a non-linear relation, the more complex the relation is, the more layers are necessary. Empirically, we usually set the number of hidden layers 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.

6 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.

6.1 Datasets

The experiments use four datasets:

– Airport [36]. Nodes represent the busiest airports in the United States, and each of the directed edges is weighted by the number of passengers traveling from one airport to another.

– Collaboration [37]. Nodes represent nations on Earth, and each of the edges is weighted by a zed count of academic papers whose author lists include that pair of nations.

– Congress [38]. Nodes represent the committees in the 102nd United States Congress, and each of the edges is weighted by the number of shared members.

– Forum [39]. Nodes represent users of a student social network at UC Irvine, and each of the directed edges is weighted by the number of messages sent between users.

The statistics of these datasets are summarized in Table 7.

Table 7. The statistics of the graph datasets used in experiments.

Dataset	Node #	Link #	Degree
Airport	500	5960	11.92
Collaboration	226	20616	91.22
Congress	163	26569	163
Forum	1899	20291	10.68

6.2 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 the training set
- 10% into the validation set
- 20% into the testing set

We use mean squared error as the prediction accuracy metric. For each trial we learn using the training set until error on the validation set increases. We then evaluate the error of the learned model on the testing set. For each experiment, we report the mean and standard deviation of the errors from 25 trials.

6.3 Experiment results

In our experiments, Model R's error is lower than all other models on all datasets, as shown in Figure 6 and Table 8.

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 the 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

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

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

$$\overline{ModelRError} \neq \overline{BaselineError} \tag{3}$$

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

$$Reduction = \frac{BaselineError - ModelRError}{BaselineError}. \tag{4}$$

The p-value is almost 0 for all datasets and error reduction is significant, shown in Table 8. Model R has the 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.

6.4 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.

6.5 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 [40]
- Operating system: Ubuntu 16.10 64-bit
- Memory: 16 GB
- Processor: Intel Core i7-4770 CPU @ 3.40GHz

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 everyone can use it without any restriction.

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

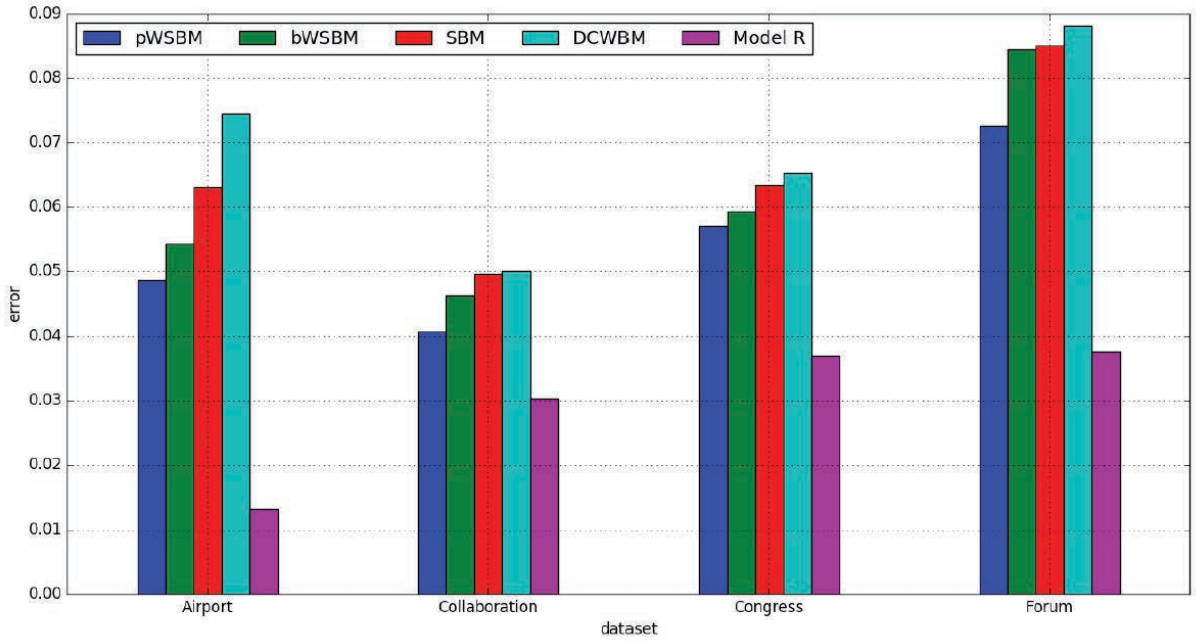

Figure 6. The mean squared errors of 5 models on 4 datasets: Model R has the lower error than every other model on every dataset. Every error value shown here is the mean error for the 25 trials in the experiment.

7 Node embedding analysis

The purpose of this Section is to find out what knowledge Model R learns during training. Our hypothesis is that the knowledge it learns consists of meaningful node embeddings where similar nodes (according to our own (human) semantics associated with the domain's entities) are close to each other in the node embedding space. We make this hypothesis based on the following observations that Model R and the skip-gram model have similar architecture in their one-hot encoding input layer and linear embedding layer, and that the skip-gram model produces word embeddings where similar words are close to each other in the word embedding space. Our goal is to verify this hypothesis by analyzing and visualizing the node embeddings produced by Model R in real-world datasets from well-understood domains to make the results obvious to most readers.

7.1 Motivation

Seeing the good performance of Model R, we are interested in this question: what exactly does Model R learn? Even though Model R outperforms some of the latest link weight prediction techniques by a large margin, the knowledge it learns is not apparent. We claim that Model R learns knowledge of nodes in the form of node embeddings from the known link weights and uses that knowledge to predict unknown link weights. This claim is plausible and similar to the claim in the original word2vec paper [15]. There have been several studies on word2vec focusing on the analysis and visualization of the word embeddings [14, 15]. These studies have provided strong evidences that the word embeddings learned by the skip-gram model represent meaningful knowledge about words. However, we need to provide evidence that the node embeddings learned by Model R represent knowledge consistent with our understanding of the domain. If we can have more in-depth study on those node embeddings similar to the studies on word embeddings, we can have a much better understanding of what Model R learns and why it performs well.

7.2 Methods

In order to perform node embedding analysis, the experiment needs additional embedding logging and visualization methods. The model trains on the datasets and produces node embeddings. The visualizer reduces the dimension of the embeddings, attaches domain metadata to the embeddings and produces the final visualization. The visualization is then analyzed to confirm closeness of semantically-similar nodes.

7.2.1 Datasets

In order to verify the knowledge of nodes learned by the model is meaningful (agrees with our domain knowledge), the experiments use real-world datasets from two well-known domains:

- Collaboration [37]. Nodes represent 226 nations on Earth, and each of the 20616 edges is weighted by the number of academic papers whose author lists include that pair of nations. This is the same dataset used in Section 6 to evaluate the prediction error of Model R.

- MovieLens100K [41]. This dataset is a recommendation dataset, and also a bipartite graph dataset. Nodes represent 1000 users and 1700 movies, and each of the 100000 edges is weighted by the rating score a user has given to a movie.

A snippet of MovieLens100K dataset is shown in Table 9 as an example.

Table 9. A snippet of MovieLens100K dataset.

User ID	Item ID	rating
196	272	3
186	302	3
22	377	1
244	51	2
166	346	1
...	...	...

7.2.2 Embeddings

The embeddings are the vectors the model maps the nodes to, and the vectors these experiments produce for us to visualize and analyze. Technically, these embeddings are the weights of the embedding layer of Model R shown in Figure 5.

7.2.3 Metadata

The metadata provides domain-specific information about the datasets necessary to verify the embeddings match our understanding about the specific domain. A snippet of MovieLens100K dataset metadata is shown in Table 10 as an example.

7.2.4 Model training

The model training process is the same as the previous experiment, with an extra step to log embedding layer weights.

Table 10. A snippet of MovieLens100K dataset metadata.

Item ID	Title	Release date
1	Toy Story	01-Jan-1995
2	GoldenEye	01-Jan-1995
3	Four Rooms	01-Jan-1995
4	Shanghai Triad	01-Jan-1995
5	Twelve Monkeys	01-Jan-1995
...	...	...

7.2.5 Embedding visualization

The embedding visualization process plays an important role in the final knowledge representation. This embedding visualization process has the following steps:

1 Join the metadata and the embeddings on the node ID to attach the related information provided by the metadata to each node.

2 Calculate the Euclidean distances between pairs of nodes.

3 Dimensionality reduction through PCA (principal component analysis) on the embeddings to project these points from the high dimensional embedding space to a 2-dimensional space so that we can visualize them.

4 Display all embeddings in an image, e.g., Figure 7 and Figure 8.

7.3 Data analysis and visualization

The experiment results meet our expectation - nodes more similar to each other (based on their semantics associated with the specific domain) have their corresponding points closer to each other in the embedding space. The experiment process runs once for each of the two datasets: MovieLens100K and Collaboration. We present the data analyses on a few well-known cases and visualizations on the entire datasets as well. The analysis for each dataset has the following steps:

Table 8. The mean squared errors with standard deviations of 5 models on 4 datasets. Welch's t-test defines its p-value as the Student's t-distribution cumulative density function $p = 2\int_{-\infty}^{-|t|} f(x)dx$.

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

– Select a well-known reference node in the domain with two similar nodes that are easy to identify.

– Sort all nodes with respect to their distances to the reference.

– Verify that the distances from the two similar nodes to the reference node are much shorter than that of the median point.

In this Section, we perform data analysis on the experiment results for both datasets.

7.3.1 MovieLens100K

For this dataset, we select the movie "Star Wars: The Empire Strikes Back" as the reference movie and the other two movies in the original Star Wars trilogy as the two similar movies, i.e., "Star Wars: A New Hope" and "Star Wars: Return of the Jedi". Notice that we do not need to assume which attributes of a movie have the most influence in users' preferences for movies, because these two movies are similar to the reference movie in many attributes such as genre, actors, screenwriter, distributor and storyline. The distances of a number of closest movies to the reference movie are shown in Table 11.

The data indicate that the distances from similar movies to the reference movie are much shorter than that from the median point. The embeddings of all movies are shown in Figure 7.

7.3.2 Collaboration

For this dataset, we select the country United States as the reference country and two other countries with similar economy, education and culture backgrounds as the two similar countries: United Kingdom and Germany. Notice that we assume economy, education and culture background have the most influence in international collaboration patterns. The distances of a number of all countries to the reference country are shown in Table 7.3.2.

Table 12. The distances of countries to the reference country for Collaboration dataset.

Country	Distance	Similarity
United States	0	self (reference)
China	0.216	most similar
...	...	...
United Kingdom	0.411	more similar
Germany	0.483	more similar
...	...	...
Jamaica	29.531	median point
...	...	...
Senegal	31.018	less similar
Peru	31.259	less similar
...	...	...
Zimbabwe	32.283	least similar

The data indicate that the distances from similar countries to the reference country are much shorter than that from the median point. The embeddings of all countries are shown in Figure 8.

8 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

Table 11. The distances of movies to the reference movie for MovieLens100K dataset.

Movie	Distance	Similarity
The Empire Strikes Back (1980)	0	self (reference)
Raiders of the Lost Ark (1981)	0.012	most similar
...	...	...
Star Wars (1977)	0.047	more similar
Return of the Jedi (1983)	0.063	more similar
...	...	...
Children of the Revolution (1996)	0.256	median point
...	...	...
Tomorrow Never Dies (1997)	0.295	less similar
Ayn Rand: A Sense of Life(1997)	0.296	less similar
...	...	...
101 Dalmatians (1996)	0.335	least similar

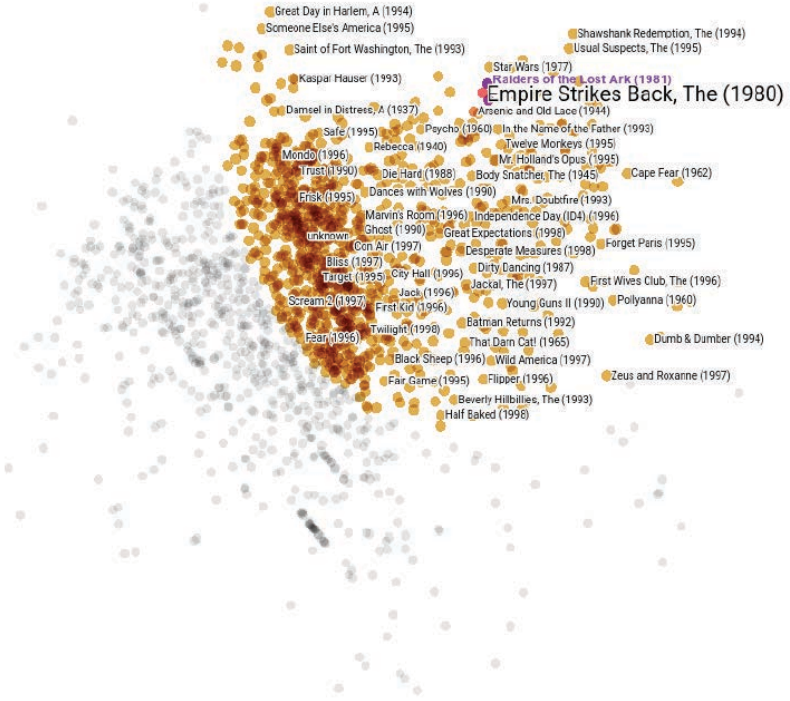

Figure 7. The embeddings of all movies in MovieLens100K dataset: "The Empire Strikes Back" is the reference movie, shown as a red node with bold font name. "Raiders of the Lost Ark" is the closest movie to the reference movie, shown as a purple node with purple name. A few other close movies are also shown as purple nodes. The image does not display names for many movies to avoid overlapping of the text. This embedding shows the overall distribution of movies where similar movies are closer to the reference movie.

Figure 8. The embeddings of countries in Collaboration dataset: The United States is the reference country, shown as a red node with bold font name. China is the closest country to the reference country, shown as a purple node with purple name. A few other close countries are also shown as purple nodes. The image does not display names for many countries to avoid overlapping of the text. This embedding shows the overall distribution of countries where similar countries are closer to the reference country.

and observable relations between nodes (i.e., link weights), and uses that information to predict unknown link weights. 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.

Model R learns meaningful node embeddings where similar nodes (based on their semantics associated with the specific domain) are close to each other in the node embedding space. This work provides direct evidences that deep learning based embedding techniques are effective in two application domains beside natural language processing: recommender systems and graph mining. We anticipate this new approach will provide effective solutions to more graph mining tasks.

9 Future work

There are a few directions we would like to study in our future work on this model.

9.1 Node embedding metrics

An important direction for this work is to identify metrics for evaluating the learned node embeddings. As embeddings are ubiquitous and valuable in deep learning and the popularity of deep learning is on the rise, we believe an important question is: what are good embeddings? A direct answer can be: embeddings that match humans' perceptions of the nodes are good embeddings. But humans' perceptions are, by nature, very complicated and subjective.

As similar nodes should have their embeddings close to each other, a possible metric is the distances of the embeddings of similar nodes. This is an ob-

vious metric and also the one we tried to use intuitively for this work. The distance measurement for embeddings is relatively easy but the similarity measurement for nodes is relatively hard. One possible way to measure similarity is some type of statistical measurement of the behaviors of nodes. For example, in recommender systems, it is natural to assume two users are very similar if they always give the same ratings to each one of the movies. The challenge is this metric is not easy to measure if there are few movies rated by both users. Another possible way to measure similarity is some type of measurement of the distance of their attribute vectors. For example, the attribute vector of a user can be [age, gender, occupation, location] as exposed by the metadata. In order for this measure to be useful, we need to know what attributes are the most relevant to users' movie preferences.

Good embeddings should produce good prediction accuracy, despite the type of node targeted for prediction. This one is very obvious because eventually some other machine learning system should use these embeddings to do valuable predictions. Therefore, good metrics should have positive correlation with prediction accuracy of the later stages of the deep learning system whose inputs are the embeddings. This does not require any work with respect to the actual nodes but it is still hard to measure: some embeddings might work well on some models but not on other models, so it is hard to decide which model should be used as the evaluation reference.

9.2 Complex graphs

This direction is especially appealing 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.

9.3 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. 164640.

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, in Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on. IEEE, 2016, pp. 1–6.

[5] 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. ACM, 2016, pp. 855–864.

[6] D. Liben-Nowell and J. Kleinberg, The link-prediction problem for social networks, Journal of the Association 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, Counterterrorism 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 IEEE/WIC/ACM International Conference on Web Intelligence (WI). 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] 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.

[15] T. Mikolov, W.-t. Yih, and G. Zweig, Linguistic regularities in continuous space word representations. in NAACL HLT, vol. 13, 2013, pp. 746–751.

[16] Q. Le and T. Mikolov, Distributed representations of sentences and documents, in Proceedings of the 31st International Conference on Machine Learning (ICML-14), 2014, pp. 1188–1196.

[17] 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.

[18] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in Proceedings of the 25th International Conference on Machine Learning. ACM, 2008, pp. 1096–1103.

[19] G. E. Hinton and R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science, vol. 313, no. 5786, pp. 504–507, 2006.

[20] X. Feng, Y. Zhang, and J. Glass, Speech feature denoising and dereverberation via deep autoencoders for noisy reverberant speech recognition, in Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on. IEEE, 2014, pp. 1759–1763.

[21] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou, Autoencoder for words, Neurocomputing, vol. 139, pp. 84–96, 2014.

[22] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, Convolutional neural networks for speech recognition, IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 22, no. 10, pp. 1533–1545, 2014.

[23] A. Van den Oord, S. Dieleman, and B. Schrauwen, Deep content-based music recommendation, in Advances in Neural Information Processing Systems, 2013, pp. 2643–2651.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[25] R. Collobert and J. Weston, A unified architecture for natural language processing: Deep neural networks with multitask learning, in Proceedings of the 25th International Conference on Machine Learning. ACM, 2008, pp. 160–167.

[26] A. M. Elkahky, Y. Song, and X. He, A multi-view deep learning approach for cross domain user modeling in recommendation systems, in Proceedings of the 24th International Conference on World Wide Web. ACM, 2015, pp. 278–288.

[27] R. Socher, Y. Bengio, and C. D. Manning, Deep learning for nlp (without magic), in Tutorial Abstracts of ACL 2012. Association for Computational Linguistics, 2012, pp. 5–5.

[28] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng, Parsing with compositional vector grammars. in ACL (1), 2013, pp. 455–465.

[29] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, C. Potts et al., Recursive deep models for semantic compositionality over a sentiment treebank, in Proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP), vol. 1631, 2013, p. 1642.

[30] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, A latent semantic model with convolutional-pooling structure for information retrieval, in Proceedings of the 23rd ACM International Conference on Information and Knowledge Management. ACM, 2014, pp. 101–110.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

[37] R. K. Pan, K. Kaski, and S. Fortunato, World citation and collaboration networks: uncovering the role of geography in science, Scientific Reports, vol. 2, 2012.

[38] 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.

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

[40] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., Tensorflow: A system for large-scale machine learning. in Operating Systems Design and Implementation (OSDI), vol. 16, 2016, pp. 265–283.

[41] F. M. Harper and J. A. Konstan, The movielens datasets: History and context, ACM Transactions on Interactive Intelligent Systems (TiiS), vol. 5, no. 4, p. 19, 2015.

Yuchen Hou is a Ph.D. student in Computer Science at Washington State University. Yuchen's research interests include artificial intelligence, machine learning, data mining, deep learning. He enjoys learning latest technologies in data visualization, web application development, serverless computing. His career plan is to work in the Internet industry as a data scientist after graduation.

Lawrence Holder is currently a professor in the School of Electrical Engineering and Computer Science at Washington State University. Before joining WSU in 2006 he was a professor in the Department of Computer Science and Engineering at the University of Texas at Arlington from 1991-2006. He received his Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 1991. Dr. Holder's research interests include artificial intelligence, machine learning, data mining, graph theory, and algorithms with applications ranging from security to biology. Dr. Holder is director of the WSU Smart Environments Research Center and co-director of the WSU Artificial Intelligence Laboratory. He is an associate editor for the journals IEEE Transactions on Knowledge and Data Engineering, Computing, Intelligent Data Analysis, and Social Network Mining. He is a senior member of ACM and IEEE.