Link Prediction Based on Graph Neural Networks

Muhan Zhang
Washington University in St. Louis
muhan@wustl.edu

Yixin Chen
Washington University in St. Louis
chen@cse.wustl.edu

ABSTRACT

Traditional methods for link prediction can be categorized into three main types: graph structure feature-based, latent feature-based, and explicit feature-based. Graph structure feature methods leverage some handcrafted node proximity scores, e.g., common neighbors, to estimate the likelihood of links. Latent feature methods rely on factorizing networks' matrix representations to learn an embedding for each node. Explicit feature methods train a machine learning model on two nodes' explicit attributes. Each of the three types of methods has its unique merits. In this paper, we propose SEAL (learning from Subgraphs, Embeddings, and Attributes for Link prediction), a new framework for link prediction which combines the power of all the three types into a single graph neural network (GNN). GNN is a new type of neural network which directly accepts graphs as input and outputs their labels. In SEAL, the input to the GNN is a local subgraph around each target link. We prove theoretically that our local subgraphs also reserve a great deal of high-order graph structure features related to link existence. Another key feature is that our GNN can naturally incorporate latent features and explicit features. It is achieved by concatenating node embeddings (latent features) and node attributes (explicit features) in the node information matrix for each subgraph, thus combining the three types of features to enhance GNN learning. Through extensive experiments, SEAL shows unprecedentedly strong performance against a wide range of baseline methods, including various link prediction heuristics and network embedding methods.

1 INTRODUCTION

Link prediction is to predict whether two nodes in a network are likely to have a link [1]. Given the ubiquitous existence of networks, it has many applications such as friend recommendation [2], movie recommendation [3], knowledge graph completion [4], and metabolic network gap filling [5].

In early works, heuristic scores are studied for link prediction. These scores measure the proximity or connectivity of two nodes [1, 6]. Popular heuristics include common neighbors, Adamic-Adar [2], Katz index [7], rooted PageRank [8], etc. Later, latent feature techniques are introduced, which learn nodes' latent features by factorizing networks' matrix representations. Matrix factorization [3] and stochastic block model [9] are two representative methods. Moreover, link prediction is also studied as a supervised learning problem, where nodes' explicit attributes are combined with some heuristics to train a supervised learning model [10].

These three types of features are largely orthogonal to each other, and are known to work well only for some networks and poorly on others. How to combine them in an effective and principled way remains a challenge. In this paper, we propose a novel framework, SEAL, which uses a *graph neural network* (GNN) to jointly learn from graph structure features, latent features, and explicit features.

Graph structure features are those features located inside the observed node and edge structures of the network. The above heuristic scores belong to graph structure features, as they are calculated based on observed local or global graph patterns. Although effective, these heuristics are handcrafted – they only capture a small set of graph features, lacking the ability to express general graph features which may decide the true link formation mechanisms. In contrast, SEAL does not presume particular heuristics, but learn graph structure features through a GNN.

Latent features are latent properties or representations of nodes, often obtained by factorizing a specific matrix derived from a network, such as the adjacency matrix or the Laplacian matrix. Through factorization, a low-dimensional embedding is learned for each node. Latent features focus more on global properties and long range effects, because the network's matrix is treated as a whole during factorization. Latent features cannot capture structural similarities between nodes [11], and usually need an extremely large dimension to express some simple heuristics [12]. Latent features are also transductive – the changing of network structure will require a complete retraining to get the latent features again.

The third type of features, **explicit features**, are often available in the form of node attributes (continuous or discrete), describing all kinds of side information about individual nodes, and can help improve the link prediction performance [13].

Our framework **SEAL** (learning from Subgraphs, Embeddings, and Attributes for Link prediction) incorporates latent features and explicit features into a node information matrix, and feeds both the subgraphs and the node information matrices to a GNN. Thus, SEAL is able to unify all three types of features into a single framework and learn from them jointly.

As our first contribution, we propose the SEAL framework to simultaneously learn from graph structure features, latent features, and explicit features. SEAL transforms a link prediction problem into a subgraph classification problem. For each pair of target nodes x and y, SEAL extracts an enclosing subgraph around them. The *enclosing subgraph* is a subgraph induced from the network by the union of all of x and y's neighboring nodes within h hops. These enclosing subgraphs contain rich information about link existence, and can be used to learn general graph structure features for link prediction.

A key issue is how large h should be. A large h enables learning more global graph structures which is desirable since global heuristics such as PageRank and Katz are shown to have better performance than local ones such as common neighbors [6]. However, using a large h often results in unaffordable time and memory consumption, making it infeasible even for small networks.

As our second contribution, we show that to learn global graph features, we do not necessarily need a very large h. We show that two most popular heuristics, Katz index and rooted PageRank, can

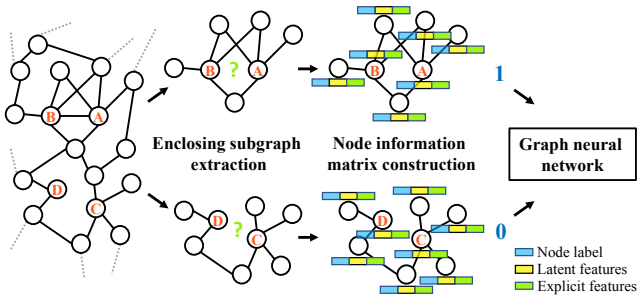

Figure 1: The SEAL training framework.

be approximated from an h-hop enclosing subgraph and the approximation error decreases exponentially with h. Our results build the foundation of subgraph-based link prediction. Our empirical evaluations verify that using an $h = 1$ or 2 can already outperform all global link prediction heuristics.

Recently, Zhang and Chen [14] proposed Weisfeiler-Lehman Neural Machine (WLNM), which also learns general graph structure features for link prediction by encoding enclosing subgraphs into fixed-size adjacency matrices and training a fully-connected neural network on the adjacency matrices. However, since fully-connected neural networks only accept fixed-size tensors, WLNM requires deleting nodes or adding dummy nodes to enclosing subgraphs to unify their sizes, which loses the strength to learn from full h-hop neighborhood. In addition, WLNM does not support learning from latent features and explicit features due to the limitation of adjacency matrix representations.

As our third contribution, we for the first time introduce graph neural network (GNN) for subgraph-based link prediction. GNN is a new type of neural network with specially designed layers to deal with graphs. Graph neural networks have the advantages of 1) supporting graphs of different sizes and 2) supporting learning from continuous node information matrices other than pure graph structures. SEAL leverages a GNN for subgraph classification, and can learn from full h-hop enclosing subgraphs as well as latent and explicit node features.

We illustrate the SEAL framework in Figure 1. We summarize our contributions as follows: 1) We propose SEAL, a novel link prediction framework which simultaneously learns from graph structure features, latent features, and explicit features. 2) We theoretically show that we do not need a huge hop number h to learn global graph structure features, which justifies subgraph-based link prediction. 3) We for the first time introduce GNN for subgraph-based link prediction. 4) We evaluate the performance of SEAL on 13 networks against 16 link prediction methods. SEAL achieves universally better performance, even outperforming the previous state-of-the-art method, WLNM, on many networks by a large margin.

2 PRELIMINARIES

2.1 Heuristic scores for link prediction

Existing graph structure features can be categorized based on the maximum hop of neighbors needed to calculate the score. For example, common neighbors (CN) and preferential attachment (PA) [16] are **first-order** features, since they only involve the one-hop neighbors of two target nodes. Adamic-Adar (AA) and resource allocation (RA) [17] are **second-order** features, as they are calculated from up to two-hop neighborhood of the target nodes.

Table 1: Popular Heuristics for Link Prediction

Name	Formula	Order
common neighbors	$\lvert\Gamma(x) \cap \Gamma(y)\rvert$	first
Jaccard	$\frac{\lvert\Gamma(x)\cap\Gamma(y)\rvert}{\lvert\Gamma(x)\cup\Gamma(y)\rvert}$	first
preferential attachment	$\lvert\Gamma(x)\rvert \cdot \lvert\Gamma(y)\rvert$	first
Adamic-Adar	$\sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\log\lvert\Gamma(z)\rvert}$	second
resource allocation	$\sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\lvert\Gamma(z)\rvert}$	second
Katz	$\sum_{l=1}^{\infty} \beta^l \lvert\text{walks}^{\langle l\rangle}(x, y)\rvert$	high
PageRank	$[\pi_x]_y + [\pi_y]_x$	high
SimRank	$\gamma \frac{\sum_{a\in\Gamma(x)}\sum_{b\in\Gamma(y)}\text{score}(a,b)}{\lvert\Gamma(x)\rvert\cdot\lvert\Gamma(y)\rvert}$	high
resistance distance	$\frac{1}{l^+_{xx}+l^+_{yy}-2l^+_{xy}}$	high

Notes: $\Gamma(x)$ denotes the neighbor set of vertex x. $\beta < 1$ is a damping factor. $\lvert\text{walks}^{\langle l\rangle}(x, y)\rvert$ counts the number of length-l walks between x and y. $[\pi_x]_y$ is the stationary distribution probability of y under the random walk from x with restart, see [15]. SimRank score is a recursive definition. l^+_{xy} is the (x, y) entry of the pseudoinverse of the graph's Laplacian matrix.

There are also some **high-order** heuristics which require knowing the entire network. Examples include Katz, PageRank (PR) [15], and resistance distance (RD) [18]. Table 1 summarizes nine popular heuristic scores.

A significant limitation of link prediction heuristics is that they are all handcrafted features, which have limited expressibility. Thus, they may fail to express some complex nonlinear patterns in the graph which actually determine the link formations.

2.2 Weisfeiler-Lehman Neural Machine

Weisfeiler-Lehman Neural Machine (WLNM) [14] is a recent algorithm which automatically learns general graph structure features from links' local enclosing subgraphs. WLNM samples a set of observed links as *positive links* and a set of random node pairs without connecting edges as *negative links*. For each link, WLNM extracts a local neighborhood subgraph enclosing it, and encodes the subgraph into an adjacency matrix using the vertex order defined by the Weisfeiler-Lehman algorithm [19]. Then, it trains a fully-connected neural network on these adjacency matrices to classify positive and negative links. WLNM achieves state-of-the-art performance on various networks, outperforming all handcrafted heuristics.

WLNM has three steps: enclosing subgraph extraction, subgraph pattern encoding, and neural network training. For enclosing subgraph extraction, for each target node pair x and y, it extracts x and y's one-hop neighbors, two-hop neighbors, and so on, until the enclosing subgraph has **more than** K vertices, where K is a user-defined integer. Note that, the extracted enclosing subgraphs may not have exactly K vertices right now.

In subgraph pattern encoding, WLNM uses the Weisfeiler-Lehman algorithm to define an order for nodes within each enclosing subgraph, so that the neural network can read different subgraphs' nodes in a consistent order and learn meaningful patterns. To unify

the sizes of the enclosing subgraphs, after getting the vertex order, the last few vertices are deleted so that all the truncated enclosing subgraphs have the same size K. These truncated enclosing subgraphs are reordered and their fixed-size adjacency matrices are fed into the fully-connected neural network to train a link prediction model. Due to the above truncation, WLNM cannot consistently learn from each link's full h-hop neighborhood. Thus, given a moderate K, WLNM may not even fully learn all first-order and second-order graph structure features.

WLNM is not expected to learn high-order graph features well as K is often small and the subgraphs only cover limited neighborhood. However, a surprising phenomenon is that WLNM outperforms all the high-order heuristics in the experiments [14], where K is only set to 10 or 20. This poses a question unanswered by [14]: why using a small K can learn high-order graph features? We will answer this question by showing that many high-order features can be well approximated by small enclosing subgraphs, and the approximation error decreases exponentially with the neighborhood size. Our result can explain WLNM and SEAL's better performance than high-order heuristics. Its practical significance is that we do not need a large K or h for enclosing subgraph extraction.

2.3 Network embedding

Network embedding has gained great popularity recently since the pioneering work of DeepWalk [20] proposes to use random walks as node sentences and apply skip-gram models [21] to learn node embeddings. LINE [22] and node2vec [23] are proposed to improve DeepWalk. The low-dimensional node embeddings are useful for visualization, node classification, and link prediction. In particular, node2vec has shown very good results in link prediction [23].

Recently, it is shown that network embedding methods (including DeepWalk, LINE, and node2vec) implicitly factorize some matrix representation of a network [24]. For example, DeepWalk approximately factorizes $\log(\mathrm{vol}(G)(\frac{1}{T}\sum_{r=1}^{T}(D^{-1}A)^r)D^{-1}) - \log(b)$, where A is the adjacency matrix of the network G, D is the diagonal degree matrix, T is skip-gram's window size, and b is the number of negative samples. For LINE and node2vec, there also exist such matrices. Since network embedding methods also factorize matrix representations of networks, we may regard them as learning more expressive latent features through factorizing some more informative matrices.

Due to the enhanced scalability and expressibility of node2vec compared to traditional latent features, we choose the node2vec embeddings as the default latent features for our SEAL framework. We note that, in principle, any kinds of latent features can be used.

3 THE SEAL FRAMEWORK

In this section, we introduce the SEAL framework. SEAL also samples a set of positive and negative training links for model training. It has three stages: enclosing subgraph extraction, node information matrix construction, and GNN learning.

Let $G = (V, E)$ be an undirected graph, where $V = \{v_1, ..., v_n\}$ is the set of vertices and $E \subseteq V \times V$ is the set of observed edges. Its adjacency matrix is A, where $A_{i,j} = 1$ if $(i, j) \in E$ and $A_{i,j} = 0$ otherwise. For any nodes $x, y \in V$, let $\Gamma(x)$ be the 1-hop neighbors of x, and $d(x, y)$ be the shortest path distance between x and y.

Algorithm 1 Enclosing Subgraph Extraction

input: target link (x, y), network G, integer h
output: h-hop enclosing subgraph $G^h_{x,y}$
$V^h_{x,y} = \{x, y\}$
$fringe = \{x, y\}$
for $i = 1, 2, ..., h$ **do**
  if $|fringe| == 0$ **then** break **end if**
  $fringe = (\bigcup_{v \in fringe} \Gamma(v)) \setminus V^h_{x,y}$
  $V^h_{x,y} = V^h_{x,y} \cup fringe$
end for
return $G^h_{x,y} = G(V^h_{x,y})$

3.1 Enclosing subgraph extraction

To learn graph structure features, SEAL extracts a local subgraph enclosing each training link.

Definition 3.1. **(Enclosing subgraph)** For a graph $G = (V, E)$, given two nodes $x, y \in V$, the h-hop enclosing subgraph for x and y is the subgraph $G^h_{x,y}$ induced from G by the set of nodes $\{ i \mid d(i, x) \leq h \text{ or } d(i, y) \leq h \}$.

The enclosing subgraph extraction algorithm is given by Algorithm 1. Note that instead of using K to control subgraph sizes like WLNM, SEAL directly uses the hop number h, and does not require all the subgraphs to have the same size.

The enclosing subgraph describes the "h-hop surrounding environment" of (x, y). Since $G^h_{x,y}$ contains all nodes within h hops to x or y and their edges, we naturally have the following theorem.

Theorem 3.2. *Any h-order graph structure features for (x, y) can be accurately calculated from $G^h_{x,y}$.*

For example, when $h \geq 2$, the extracted enclosing subgraphs will contain all the information needed to calculate any first-order and second-order link prediction heuristics, such as common neighbors, preferential attachment, and Adamic-Adar heuristics. Due to the unparallel feature learning ability of neural networks, learning from such enclosing subgraphs is expected to achieve performance at least as good as a wide range of link prediction heuristics.

It is shown that high-order heuristics often perform better than first-order and second-order heuristics [6]. It seems that, to learn high-order features, a larger h will be needed, which is often infeasible on real-world networks due to time and space complexities.

Surprisingly, our following analysis shows that SEAL is actually **capable of learning high-order graph features**, even with a small h. We support this by studying two popular high-order heuristics: Katz index and PageRank. We prove that these two heuristics can be approximated well by the h-hop enclosing subgraph and the approximation error decreases exponentially with h.

3.1.1 Katz index. The Katz index [7] for (x, y) is defined as

$$\mathrm{Katz}_{x,y} = \sum_{l=1}^{\infty} \beta^l |\mathrm{walks}^{\langle l \rangle}(x, y)|, \tag{1}$$

where $\mathrm{walks}^{\langle l \rangle}(x, y)$ is the set of length-l walks between x and y. Katz index sums over the collection of all walks between x and y where a walk of length l is damped by β^l $(0 < \beta < 1)$, giving more weights to shorter walks.

Writing (1) into matrix expressions, we have:

$$\text{Katz}_{x,y} = \sum_{l=1}^{\infty} \beta^l [A^l]_{x,y}, \tag{2}$$

where A^l is l^{th} power of the adjacency matrix of the network.

Lemma 3.3. *Any walk between x and y with length $l \leq 2h+1$ is included in the h-hop enclosing subgraph $G^h_{x,y}$.*

Proof. Consider any walk $w = \langle x, v_1, v_2, \cdots, v_{l-1}, y\rangle$ between x and y with length l. We need to show that the nodes $v_1, \cdots, v_{l-1}$ are all included in $G^h_{x,y}$. Consider any node v_i, let its shortest path distance to x and y be $d(i,x)$ and $d(i,y)$, respectively.

Assume $d(i,x) \geq h+1$ and $d(i,y) \geq h+1$. Then, we have $2h+1 \geq l = |\{x, v_1, \cdots, v_{i-1}\}| + |\{v_{i+1}, \cdots, v_{l-1}, y\}| \geq d(i,x) + d(i,y) \geq 2h+2$, a contradiction. Thus, $d(i,x) \leq h$ or $d(i,y) \leq h$. By the definition of $G^h_{x,y}$, v_i must be included in $G^h_{x,y}$. Hence, all the nodes $v_1, \cdots, v_{l-1}$ in the walk w are included in $G^h_{x,y}$. □

Lemma 3.3 indicates that we can find all the walks between x and y with length up to $2h+1$ from $G^h_{x,y}$. Thus, we can approximate Katz by summing over these walks, as follows:

$$\widetilde{\text{Katz}}_{x,y} = \sum_{l=1}^{2h+1} \beta^l [A^l]_{x,y}. \tag{3}$$

Next, we show that the approximation error of (3) decreases exponentially with h. We first give a bound on the number of length-l walks between any pair of nodes in the network.

Lemma 3.4. *The number of length-l walks $[A^l]_{i,j}$ between any pair of nodes i and j is bounded by d^{l-1}, where d is the maximum node degree of the network.*

Proof. We prove the lemma by induction. When $l = 1$, $A_{i,j} \leq 1$ for any (i,j). Thus the base case is correct. Now, assume by induction that $[A^l]_{i,j} \leq d^{l-1}$ for any (i,j), we have

$$[A^{l+1}]_{i,j} = \sum_{k=1}^{|V|} [A^l]_{i,k} A_{k,j} \leq d^{l-1} \sum_{k=1}^{|V|} A_{k,j} \leq d^{l-1} d = d^l. \tag{4}$$

□

With Lemma 3.4, we bound the approximation error of (3) as follows.

Theorem 3.5. *Assume the damping factor $\beta < \frac{1}{d}$, where d is the maximum node degree. The error between the approximated Katz in (3) and the real Katz in (2) is bounded as*

$$|Katz_{x,y} - \widetilde{Katz}_{x,y}| \leq \beta^{2h+2}(1-\beta d)^{-1} d^{2h+1}. \tag{5}$$

Proof. We have,

$$\begin{aligned} |\text{Katz}_{x,y} - \widetilde{\text{Katz}}_{x,y}| &= |\sum_{l=1}^{\infty} \beta^l [A^l]_{x,y} - \sum_{l=1}^{2h+1} \beta^l [A^l]_{x,y}| \\ &= |\sum_{l=2h+2}^{\infty} \beta^l [A^l]_{x,y}| \leq \sum_{l=2h+2}^{\infty} \beta^l d^{l-1} \qquad (6) \\ &= \beta^{2h+2} d^{2h+1} \sum_{l=0}^{\infty} (\beta d)^l = \beta^{2h+2} d^{2h+1} (1-\beta d)^{-1}. \end{aligned}$$

□

In practice, the damping factor β is often set to very small values like 5E-4 [1]. Thus, Theorem 3.5 implies that the Katz index can be very well approximated from the h-hop enclosing subgraph, and the approximation error decreases exponentially with h.

3.1.2 PageRank. The rooted PageRank for node x calculates the stationary distribution of a random walker starting from x, who iteratively moves to a random neighbor of its current position with probability α or returns to x with probability $1-\alpha$. Let π_x denote the stationary distribution vector with $[\pi_x]_i$ denoting the probability that the random walker is at node i under the stationary distribution. Note that $\sum_{i \in V} [\pi_x]_i = 1$.

Let P be the transition matrix with $P_{i,j} = \frac{1}{|\Gamma(j)|}$ if $(i,j) \in E$ and $P_{i,j} = 0$ otherwise. Let $\mathbf{e}_x$ be a vector with the x^{th} element being 1 and others 0. The stationary distribution satisfies

$$\pi_x = \alpha P \pi_x + (1-\alpha)\mathbf{e}_x = [\alpha P + (1-\alpha)\mathbf{e}_x \mathbf{1}^\top]\pi_x, \tag{7}$$

where $\mathbf{1}$ is an all-one column vector. Define $S := \alpha P + (1-\alpha)\mathbf{e}_x \mathbf{1}^\top$. Note that S is a left stochastic matrix with each column summing to 1. Then we can derive the eigenvector expression of π_x, $\pi_x = S\pi_x$.

π_x can be computed by the power iteration method:

- Pick an initial guess $\pi_x^{(0)}$.
- Repeat
$$\pi_x^{(k)} = S\pi_x^{(k-1)}, \tag{8}$$
until termination criterion is satisfied.

Given a local enclosing subgraph, we can approximate PageRank using the following power method:

- Set $\pi_x^{(0)} = \mathbf{e}_x$.
- For $k = 1$ to h, calculate
$$\pi_x^{(k)} = S\pi_x^{(k-1)}. \tag{9}$$
- Return the approximated PageRank $\pi_x^{(h)}$.

Although the global matrix S still appears in the second step, we do not need to know the full S to compute (9). Consider a random walker starting from x, within k steps, it will never reach a node more than k hops away from x. Thus, $[\pi_x^{(k-1)}]_j = 0$ if $d(j,x) \geq k$.

Now, let's look at (9), we have

$$[\pi_x^{(k)}]_i = \sum_{j=1}^{|V|} S_{i,j} [\pi_x^{(k-1)}]_j = \sum_{j:d(j,x) \leq k-1} S_{i,j} [\pi_x^{(k-1)}]_j. \tag{10}$$

By the definition of h-hop enclosing subgraphs, the transition probability $S_{i,j} = \frac{1}{|\Gamma(j)|}$ is known if $d(j,x) \leq h-1$. This means that we can always calculate (10) without unknown variables.

Now, we can bound the error between the approximated PageRank $\pi_x^{(h)}$ and the real PageRank π_x as follows.

Theorem 3.6. $\|\pi_x - \pi_x^{(h)}\|_1 \leq 2\alpha^h$.

Proof.

$$\begin{aligned} \|\pi_x - \pi_x^{(h)}\|_1 &= \|S(\pi_x - \pi_x^{(h-1)})\|_1 \\ &= \|\alpha P(\pi_x - \pi_x^{(h-1)}) + (1-\alpha)\mathbf{e}_x(\mathbf{1}^\top \pi_x - \mathbf{1}^\top \pi_x^{(h-1)})\|_1 \\ &= \|\alpha P(\pi_x - \pi_x^{(h-1)})\|_1 \leq \alpha \|P\|_1 \|\pi_x - \pi_x^{(h-1)}\|_1 \\ &= \alpha \|\pi_x - \pi_x^{(h-1)}\|_1 \leq \alpha^h \|\pi_x - \pi_x^{(0)}\|_1, \end{aligned} \tag{11}$$

where the second to last step is true because P is a column stochastic matrix, whose matrix 1-norm (maximum absolute column sum) is 1. Since π_x and $\pi_x^{(0)} = \mathbf{e}_x$ are both probability distributions, we know $\|\pi_x - \pi_x^{(0)}\|_1 \leq \|\pi_x\|_1 + \|\pi_x^{(0)}\|_1 = 2$. Therefore, we have $\|\pi_x - \pi_x^{(h)}\|_1 \leq 2\alpha^h$. □

When used for link prediction, the score for (x, y) is given by $[\pi_x]_y + [\pi_y]_x$. We have $|[\pi_x]_y - [\pi_x^{(h)}]_y| \leq \|\pi_x - \pi_x^{(h)}\|_1 \leq 2\alpha^h$. The same bound can be obtained for $|[\pi_y]_x - [\pi_y^{(h)}]_x|$. Finally, we have that $|([\pi_x]_y + [\pi_y]_x) - ([\pi_x^{(h)}]_y + [\pi_y^{(h)}]_x)| \leq |[\pi_x]_y - [\pi_x^{(h)}]_y| + |[\pi_y]_x - [\pi_y^{(h)}]_x| \leq 4\alpha^h$.

The above results show that the rooted PageRank heuristic can be approximated from the h-hop enclosing subgraph where the approximation error decreases exponentially with h. It has also been shown empirically that local methods can often estimate PageRank very well in practice [25].

3.1.3 Other high-order features. There are several other high-order heuristics, e.g., SimRank [26] and resistance distance [18]. Proving that they can also be approximated from local enclosing subgraphs is beyond the scope of this paper. However, some studies have shown that SimRank and resistance distance can be approximated locally [27, 28]. It intuitively makes sense as distant structures are expected to have little influence on link formation.

Our theoretical results can explain why WLNM achieves better performance than high-order heuristics as shown in [14]. The results also build the foundation of subgraph-based link prediction, as they imply that we do not need a very large h to learn good graph features for link prediction. To summarize, from the extracted small enclosing subgraphs around links, we are able to accurately calculate first and second-order features, and approximate a wide range of high-order features with small errors. Therefore, leveraging the expressing power of neural networks, we are expected to learn good graph structure features through subgraph learning.

3.2 Node information matrix construction

In WLNM, enclosing subgraphs are encoded into fixed-size adjacency matrices. This has two limitations: 1) To fix the matrix size, some nodes must be removed from the h-hop enclosing subgraph. As a result, the h-order graph features may not be completely learned. 2) Training on adjacency matrices using fully-connected NN is only capable of learning from graph structures, but cannot simultaneously learn from latent features and explicit features.

Different from WLNM, our SEAL does not need to truncate enclosing subgraphs, since SEAL uses a GNN which can accept graphs of arbitrary sizes. The input to a GNN consists of the adjacency matrix A (with slight abuse of notation) of the enclosing subgraph and a *node information matrix* X where each row of X contains additional information about a node. This provides a way to incorporate latent and explicit features into the GNN learning too.

In SEAL, the construction of the node information matrix for an enclosing subgraph is done by the following steps.

1) Assign integer labels to nodes according to their topological positions within the subgraph and encode them into vectors $\mathcal{L} = \{\mathbf{x}_1^l, \mathbf{x}_2^l, \cdots, \mathbf{x}_n^l\}$ using one-hot encoding, where we assume there are n nodes in the enclosing subgraph and we use $\mathbf{x}_i^l$ to denote the one-hot encoding vector of node i's label. Note that the labeling process is done separately for different enclosing subgraphs, i.e., the labels only reflect nodes' positions **within** the subgraph.

2) Get the node embedding vectors $\mathcal{E} = \{\mathbf{x}_1^e, \mathbf{x}_2^e, \cdots, \mathbf{x}_n^e\}$ for nodes in the enclosing subgraph, where $\mathbf{x}_i^e$ is the node embedding vector of node i generated in a preprocessing step. Note that a network embedding algorithm is only run once for the entire network in order to generate latent node features which are reused in different enclosing subgraphs.

3) Get the node attribute vectors (if available) $\mathcal{A} = \{\mathbf{x}_1^a, \mathbf{x}_2^a, \cdots, \mathbf{x}_n^a\}$.

4) Concatenate the one-hot encodings of node labels, embeddings, and attributes to form the node information matrix X, where the i^{th} row of X is the concatenation of $\mathbf{x}_i^l$, $\mathbf{x}_i^e$, and $\mathbf{x}_i^a$.

We explain the first and second steps in detail in the following.

3.2.1 Node labeling. The first step, node labeling, uses a function $g : V \to \mathbb{N}$ which assigns an integer label $g(i)$ to every node i in the enclosing subgraph. The purpose is to use different labels to mark nodes' different roles in an enclosing subgraph: 1) The center nodes x and y are the target nodes between which the link is located. 2) Other nodes maintain an intrinsic directionality – they are iteratively added outwards from center based on their distance to the center nodes. 3) The h-hop nodes mark the boundary of the enclosing subgraph. A proper node labeling should mark such differences. If we do not mark such differences, GNNs will not be able to tell where are the target nodes between which a link existence should be predicted, and lose structural information.

Our node labeling method is derived from the following criteria: 1) The two target nodes x and y always have the distinctive label "1". 2) Nodes i and j have the same label if $d(i, x) = d(j, x)$ and $d(i, y) = d(j, y)$. The second criterion is because, intuitively, a node i's topological position within an enclosing subgraph can be described by its *coordinates* with respect to the target nodes x and y, namely $(d(i, x), d(i, y))$. Thus, we let nodes with the same coordinates have the same label, so that the integer labels can reflect nodes' relative positions within subgraphs.

Note that when calculating $d(i, x)$, we temporally remove y from the subgraph, and vice versa. This is because we aim to use the pure distance between i and x without the influence of y. If we do not remove y, the distance $d(i, x)$ will be upper bounded by $d(i, y) + d(x, y)$, obscuring the true distance between i and x.

Now, we give a simple node labeling algorithm based on the above criteria. First, assign label 1 to x and y. Then, for all nodes with $(d(i, x), d(i, y)) = (1, 1)$, assign label $g(i) = 2$. Nodes with coordinates $(1, 2)$ or $(2, 1)$ get label 3. Nodes with coordinates $(1, 3)$ or $(3, 1)$ get 4. Nodes with $(2, 2)$ get 5. Nodes with $(1, 4)$ or $(4, 1)$ get 6. Nodes with $(2, 3)$ or $(3, 2)$ get 7. So on and so forth. For nodes with $d(i, x) = \infty$ or $d(i, y) = \infty$, we give them a null label 0. Figure 2 illustrates this labeling process.

Our labeling algorithm not only satisfies the above label-by-coordinates criteria, but also attains the additional benefits that for nodes i and j:

1) if $d(i, x) + d(i, y) \neq d(j, x) + d(j, y)$, then
$d(i, x) + d(i, y) < d(j, x) + d(j, y) \Leftrightarrow g(i) < g(j)$; and
2) if $d(i, x) + d(i, y) = d(j, x) + d(j, y)$, then
$d(i, x)d(i, y) < d(j, x)d(j, y) \Leftrightarrow g(i) < g(j)$.

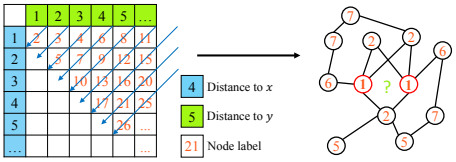

Figure 2: Node labeling in SEAL.

That is, the magnitude of node labels also reflects their distance to center. Nodes with smaller arithmetic mean distance to the target nodes get smaller labels. If two nodes have the same arithmetic mean distance, the node with a smaller geometric mean distance to the target nodes gets a smaller label. Note that these additional benefits will not be available under one-hot encoding of node labels, since the magnitude information will be lost after one-hot encoding. However, such a labeling is potentially useful when node labels are directly used for training, or used to rank the nodes.

Our node labeling algorithm is different from the Weisfeiler-Lehman algorithm used in WLNM. In WLNM, node labeling is for defining a node order in adjacency matrices – the labels are not really input to machine learning models. To rank nodes with least ties, the node labels should be as fine as possible in WLNM. In comparison, the node labels in SEAL need not be very fine, as their purpose is for indicating nodes' different roles within the enclosing subgraph, not for ranking nodes.

Finally, we give a closed-form formula for our node labeling algorithm above.

Proposition 3.7. *The above node labeling algorithm has a perfect hashing function*

$$g(i) = 1 + min(d_x, d_y) + (d/2)[(d/2) + (d\%2) - 1], \qquad (12)$$

where $d_x := d(i, x)$, $d_y := d(i, y)$, $d := d_x + d_y$, $(d/2)$ *and* $(d\%2)$ *are the integer quotient and remainder of* d *divided by* 2*, respectively.*

3.2.2 Node embedding by negative injection. Generating the node embeddings for SEAL is not trivial. Suppose we are given the observed network $G = (V, E)$, a set of sampled positive training links $E_p \subseteq E$, and a set of sampled negative training links E_n with $E_n \cap E = \emptyset$. If we directly generate embeddings on G. the node embeddings will record the link existence information of the training links (since E_p is included in the observed E). Due to its great expressing power and capacity, GNN can quickly find out such link existence information and optimize by only fitting this part of information. This results in bad generalization performance in our experiments.

Our trick is to temporally add E_n into E, and generate the embeddings on $G' = (V, E \cup E_n)$. This way, the positive and negative training links will have the same link existence information recorded in embeddings, so that GNN cannot only fit this part of information to classify links. Furthermore, the added E_n increases the connectivity of the network, which potentially makes node embeddings contain more long-range node connection information. We empirically verified the much improved performance of this trick to SEAL. We name this trick "negative injection".

3.3 Graph neural network (GNN) learning

Traditional neural networks can only deal with tensor data, where all tensors have the same size and the tensor elements are arranged in a consistent order. For example, in image classification, images have the same size and their pixels are arranged with a spatial order. If image pixels are shuffled randomly, convolutional neural networks (CNNs) will fail to train a working model.

However, graphs usually lack such tensor representations like images. Thus, traditional neural networks cannot directly deal with graphs. *Graph neural network* (GNN) [29–33] is a new type of neural networks which are capable of reading and learning directly from graphs. As an emerging field, GNN has attracted a lot of research in the past two years. It has achieved state-of-the-art results in many tasks such as graph classification [32, 33], semi-supervised learning [31], and chemical property regression [30, 34].

A GNN usually consists of 1) *graph convolution layers* which extract local substructure features for individual nodes, and 2) a *graph aggregation layer* which aggregates node-level features into a graph-level feature vector. Many graph convolution layers can be incorporated into a message passing framework [35]. In particular, a graph convolution operation on a node propagates its neighboring nodes' states to it, and then applies a nonlinear transformation to get the new state of this node. Graph convolution summarizes the local feature and structure patterns around each node and is translation-invariant (meaning all nodes have the same graph convolution parameters), effectively mimicking the convolution filters of traditional CNNs.

Compared to other graph classification methods such as graph kernels, GNNs have the following advantages. 1) The ability to deal with big data – GNNs do not have the quadratic complexity of graph kernels to store the kernel matrix, and can leverage the power of GPU computing. This ability is very useful for subgraph-based link prediction when the network size is large. 2) The ability to learn from continuous node features – GNNs naturally support learning from continuous node features in the node information matrix, while most graph kernels can only learn from discrete structures and integer node labels. These two advantages make GNNs particularly suitable for SEAL.

Recently, Zhang et. al. proposed a novel GNN architecture, named Deep Graph Convolutional Neural Network (DGCNN) [33]. DGCNN is equipped with propagation-based graph convolution layers and a novel graph aggregation layer, called SortPooling. DGCNN has achieved state-of-the-art graph classification performance on various benchmark datasets. We use DGCNN as the default GNN engine in SEAL. We illustrate the overall architecture of DGCNN in Figure 3. Given the adjacency matrix $A \in \{1, 0\}^{n \times n}$ and the node information matrix $X \in \mathbb{R}^{n \times c}$ of an enclosing subgraph, DGCNN uses the following graph convolution layer:

$$Z = f(\tilde{D}^{-1}\tilde{A}XW), \qquad (13)$$

where $\tilde{A} = A + I$, $\tilde{D}$ is a diagonal degree matrix with $\tilde{D}_{i,i} = \sum_j \tilde{A}_{i,j}$, $W \in \mathbb{R}^{c \times c'}$ is a matrix of trainable graph convolution parameters, f is an element-wise nonlinear activation function, and $Z \in \mathbb{R}^{n \times c'}$ are the new node states. The mechanism behind (13) is that the initial node states X are first applied a linear transformation by multiplying W, and then propagated to neighboring nodes through

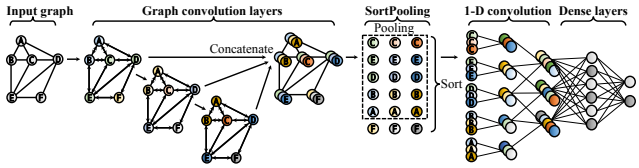

Figure 3: The DGCNN architecture.

the propagation matrix $\tilde{D}^{-1}\tilde{A}$. After graph convolution, the i^{th} row of Z becomes:

$$Z_i = f\big(\frac{1}{|\Gamma(i)|+1}[X_i W + \sum_{j\in\Gamma(i)} X_j W]\big), \tag{14}$$

which summarizes the node information as well as the first-order structure pattern from i's neighbors. DGCNN stacks multiple graph convolution layers (13) and concatenates each layer's node states as the final node states, in order to extract multi-hop node features.

The graph aggregation layer constructs a graph-level feature vector from individual nodes' final states, which is used for graph classification. The most widely used aggregation operation is simple summing, i.e., nodes' final states after graph convolutions are summed up as the graph's representation. However, the averaging effect of summing will lose much individual nodes' information as well as the topological information of the graph. DGCNN uses a novel *SortPooling layer*, which sorts the final node states according to the last graph convolution layer's output to achieve an isomorphism invariant node ordering [33]. A max-k pooling operation is then used to unify the sizes of the sorted representations of different graphs, which enables training a traditional 1-D CNN on the node sequence. It is shown that the SortPooling layer achieves much improved performance than summing [33].

Discussion. It is interesting that many latent feature models can be seen as special cases of SEAL with $h = 0$. When $h = 0$, there are only the two isolated target nodes in the enclosing subgraph – $\tilde{D}^{-1}\tilde{A}$ becomes an 2×2 identity matrix. Thus, the graph convolution layers (13) will reduce to two standard neural networks on the target nodes' embeddings. If we use a summing or Hadamard product based aggregation layer, we can recover node2vec's link prediction model. If we use an inner product-based aggregation layer, we can recover the matrix factorization model.

We summarize SEAL's training and testing process as follows:

- Given a network $G = (V, E)$, randomly select N edges from E as positive training links, and randomly select another N node pairs (without edges) as negative training links. Positive and negative training links have labels $y = 1$ and $y = 0$, respectively.
- For each training link, extract the h-hop enclosing subgraph A from the observed network, and build the node information matrix X. Then, feed the (A, X, y) tuples to DGCNN to train a link prediction model.
- Given any test link, also extract its h-hop enclosing subgraph, build its node information matrix, and then feed the (A, X) to the trained DGCNN to get the prediction y.

Note that, in any positive training link's enclosing subgraph, we should always remove the edge between the two target nodes. This is because this edge will contain the link existence information, which is not available in any testing link's enclosing subgraph.

4 EXPERIMENTAL RESULTS

We conduct extensive experiments to evaluate the performance of SEAL. Our results show that SEAL is a superb and robust framework for link prediction, achieving unprecedentedly strong performance on various networks. All the experiments are ran 5 times on a 20-core Linux server with 128GB RAM. The average AUC results are reported. We guarantee the reproducibility of all results. The code and datasets are in https://github.com/muhanzhang/SEAL.

4.1 Experiments on medium-sized networks

We first do experiments under the setting of WLNM [14].

Datasets. The eight datasets used are: USAir, NS, PB, Yeast, C.ele, Power, Router, and E.coli. USAir [36] is a network of US Air lines with 332 nodes and 2,126 edges. NS [37] is a collaboration network of researchers in network science with 1,589 nodes and 2,742 edges. PB [38] is a network of US political blogs with 1,222 nodes and 16,714 edges. Yeast [39] is a protein-protein interaction network in yeast with 2,375 nodes and 11,693 edges. C.ele [40] is a neural network of C. elegans with 297 nodes and 2,148 edges. Power [40] is an electrical grid of western US with 4,941 nodes and 6,594 edges. Router [41] is a router-level Internet with 5,022 nodes and 6,258 edges. E.coli [42] is a pairwise reaction network of metabolites in E. coli with 1,805 nodes and 14,660 edges. In each dataset, all existing links are randomly split into a training set (90%) and a testing set (10%). For the testing set, we sample an equal number of node pairs with no connecting edges as the negative testing links to evaluate the link prediction performance. For the training set, we additionally sample an equal number of node pairs without edges to construct the negative training data. Area under the ROC curve (AUC) is adopted to measure the performance.

Baselines and experimental setting. A total of 14 methods are used as baselines. They are 1) nine popular heuristic methods listed in Table 1: common neighbors (CN), Jaccard (Jac.), preferential attachment (PA), Adamic-Adar (AA), resource allocation (RA), Katz, resistance distance (RD), PageRank (PR), and SimRank (SR); and 2) five state-of-the-art latent feature models: matrix factorization (MF), stochastic block model (SBM) [9], node2vec (N2V) [23], LINE [22], and spectral clustering (SPC). For Katz, we set the damping factor β to 0.001. For PageRank, we set the damping factor α to 0.85. For SBM, we use the implementation of [43] using a latent group number 12. For MF, we use the libFM [44] software with the default parameters. For node2vec, LINE, and spectral clustering, we first generate 128-dimensional embeddings from the observed networks with default parameters of each software. Then, we use the Hadamard product of two nodes' embeddings as a link's embedding as suggested in [23], and train a logistic regression model with LIBLINEAR [45] using automatic hyperparameter selection.

For our SEAL, we select h only from $\{1, 2\}$ in order to demonstrate its ability to learn high-order features from local subgraphs. The selection principle is very simple: if the second-order heuristic AA outperforms the first-order heuristic CN on 10% validation data, then we choose $h = 2$, otherwise we choose $h = 1$. To construct the node information matrix, we use the hashing function in (12) to generate node labels. We use the 128-dimensional node2vec embeddings as the embedding part. The used datasets do not have node attributes. The DGCNN architecture in SEAL contains four

Table 2: AUC results. For WLK, WLNM and SEAL, bold results mean they outperform all 14 baselines. Best result for each dataset is in blue.

Data	CN	Jac.	PA	AA	RA	Katz	RD	PR	SR	MF	SBM	N2V	LINE	SPC	WLK	WLNM	SEAL
USAir	0.9368	0.9027	0.8876	0.9507	0.9591	0.9273	0.8813	0.9486	0.7963	0.9117	0.9431	0.9122	0.8003	0.7482	**0.9598**	0.9571	**0.9729**
NS	0.9495	0.9495	0.6893	0.9498	0.9498	0.9524	0.5784	0.9529	0.9522	0.7195	0.9307	0.9198	0.8048	0.8829	**0.9864**	**0.9886**	**0.9761**
PB	0.9218	0.8760	0.9022	0.9250	0.9263	0.9306	0.8841	0.9374	0.7740	0.9452	0.9418	0.8621	0.7779	0.8261	OOM	0.9363	**0.9540**
Yeast	0.8966	0.8963	0.8279	0.8973	0.8973	0.9264	0.8835	0.9314	0.9190	0.9010	0.9149	0.9407	0.8527	0.9346	**0.9550**	**0.9582**	**0.9693**
C.ele	0.8471	0.7958	0.7460	0.8659	0.8716	0.8606	0.7315	0.9046	0.7650	0.8633	0.8828	0.8387	0.6958	0.5007	0.8965	0.8603	**0.9114**
Power	0.5912	0.5912	0.4402	0.5912	0.5912	0.6570	0.8515	0.6632	0.7648	0.5002	0.6714	0.7681	0.5779	**0.9147**	0.8404	0.8488	0.8502
Router	0.5640	0.5639	0.4788	0.5641	0.5641	0.3871	0.9356	0.3883	0.3751	0.7822	0.8529	0.6518	0.6723	0.7067	0.8792	**0.9463**	**0.9630**
E.coli	0.9353	0.8125	0.9174	0.9524	0.9587	0.9329	0.8949	0.9548	0.6405	0.9387	0.9388	0.9075	0.8270	0.9514	OOM	**0.9706**	**0.9704**

Table 3: Comparison of different link prediction methods

	Heuristics	Embedding	WLK	WLNM	SEAL
Graph structure features	Yes	No	Yes	Yes	Yes
Learn from full h-hop	No	n/a	Yes	No	Yes
Latent/explicit features	No	Yes	No	No	Yes
Model	n/a	LR	SVM	NN	GNN

graph convolution layers as in (13), a SortPooling layer, two 1-D convolution layers and a dense layer, see [33]. We train DGCNN for 50 epochs, and select the model with the smallest loss on the 10% validation data to predict the testing links.

We further compare SEAL with two more subgraph-based link prediction methods, Weisfeiler-Lehman graph kernel (WLK) [46] and WLNM [14]. WLK is a state-of-the-art graph kernel for graph classification in terms of both accuracy and efficiency. We extract 1 or 2-hop enclosing subgraphs for WLK (same criterion as SEAL) and select its height parameter from $\{0, 1, 2, 3, 4, 5\}$ on 10% training links. WLK does not support continuous node information, but supports integer node labels. We use the same node labels from (12). For WLNM, we set $K = 10$ which is the best performing K after experimentation. We compare the characteristics of different link prediction methods in Table 3. The average AUC results are reported in Table 2 ("OOM" means out of memory).

Results. As we can see from Table 2, SEAL is overall the best link prediction method. First, we find that subgraph-based methods (WLK, WLNM, SEAL) generally have better performance than all heuristic methods including high-order ones, which further demonstrates the advantages of learning over handcrafting graph features. The better performance of subgraph-based methods than high-order heuristics also validates our theoretical results.

From Table 2, it is worth noting that although embedding methods have achieved excellent performance in node classification [20, 23], they actually do not excel in link prediction – simple heuristics can outperform them very often, not to mention the subgraph-based methods. This is because embedding methods only use latent features, which cannot effectively capture structural similarities of links that may be most useful for predicting links. However, we show that combining node embeddings with subgraph learning is able to boost the performance a lot. Compared with WLK and WLNM, our SEAL shows significantly improved performance on various networks by learning from both subgraph structures and node information matrices, which is empowered by the use of GNN. The ability to simultaneously learn from subgraphs, embeddings and attributes makes SEAL an outstanding off-the-shelf method for link prediction. The comprehensiveness of its features makes it robust, capable of handling vastly different networks.

Table 4: AUC results on medium to large networks.

	N2V	LINE	SPC	WLNM	SEAL
arXiv	0.9618	0.8464	0.8700	0.9919	**0.9940**
Facebook	0.9905	0.8963	0.9859	0.9924	**0.9940**
BlogCatalog	0.8597	0.9092	0.9674	0.9655	**0.9810**
Wikipedia	0.7659	0.7444	0.9954	0.9905	**0.9963**
PPI	0.7031	0.7282	0.9227	0.8879	**0.9352**

4.2 Experiments on medium to large networks

We further conduct experiments with the setting of node2vec [23] on five networks: arXiv (18,722 nodes and 198,110 edges) [47], Facebook (4,039 nodes and 88,234 edges) [47], BlogCatalog (10,312 nodes, 333,983 edges and 39 attributes) [48], Wikipedia (4,777 nodes, 184,812 edges and 40 attributes) [49], and Protein-Protein Interactions (PPI) (3,890 nodes, 76,584 edges and 50 attributes) [50]. For each network, 50% of random links are removed and used as testing data, while keeping the remaining network connected. For Facebook and arXiv, all remained links are used as positive training data. For PPI, BlogCatalog and Wikipedia, we sample 10,000 remained links as positive training data. We compare SEAL ($h = 1$) with node2vec, LINE, SPC, and WLNM ($K = 10$). For node2vec, we use the parameters provided in [23] if available. Table 4 shows the results. As we can see, SEAL consistently outperforms all embedding methods. Especially on the last three networks, SEAL (with node2vec embeddings) outperforms pure node2vec by large margins. These results indicate that in many cases, embedding methods alone cannot capture the most useful link prediction information, while effectively combining the power of different types of features results in much better performance. SEAL also consistently outperforms WLNM.

5 CONCLUSIONS

In this paper, we have proposed a novel link prediction framework, SEAL, which systematically transforms a link prediction problem to a subgraph learning problem. By incorporating latent and explicit features into node information matrices, SEAL uses a GNN to predict links leveraging both subgraph structures and node information matrices, achieving new state-of-the-art results. Our theoretical contributions include formally proving the feasibility of subgraph learning for link prediction, and showing that in the special case of $h = 0$, SEAL reduces to latent feature methods.

There are many interesting future directions, e.g., generalizing SEAL to knowledge graphs and recommender systems, developing smart ways for nonuniform subgraph extraction, and so on. We believe that the SEAL framework can not only help link prediction tasks, but also enhance other network inference tasks such as node

ranking and node classification, where handcrafted graph features are still the prevailing method.

REFERENCES

[1] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[2] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[3] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[4] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[5] Tolutola Oyetunde, Muhan Zhang, Yixin Chen, Yinjie Tang, and Cynthia Lo. Boostgapfill: Improving the fidelity of metabolic network reconstructions through integrated constraint and pattern-based methods. *Bioinformatics*, 2016.

[6] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.

[7] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[8] Taher Haveliwala, Sepandar Kamvar, and Glen Jeh. An analytical comparison of approaches to personalizing pagerank. Technical report, Stanford, 2003.

[9] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9 (Sep):1981–2014, 2008.

[10] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *SDMâĂŹ06: Workshop on Link Analysis, Counter-terrorism and Security*, 2006.

[11] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394. ACM, 2017.

[12] Maximilian Nickel, Xueyan Jiang, and Volker Tresp. Reducing the rank in relational factorization models by including observable patterns. In *Advances in Neural Information Processing Systems*, pages 1179–1187, 2014.

[13] He Zhao, Lan Du, and Wray Buntine. Leveraging node attributes for incomplete relational data. *arXiv preprint arXiv:1706.04289*, 2017.

[14] Muhan Zhang and Yixin Chen. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 575–583. ACM, 2017.

[15] Sergey Brin and Lawrence Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer networks*, 56(18):3825–3833, 2012.

[16] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[17] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009.

[18] Douglas J Klein and Milan Randić. Resistance distance. *Journal of Mathematical Chemistry*, 12(1):81–95, 1993.

[19] Boris Weisfeiler and AA Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2 (9):12–16, 1968.

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

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

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

[23] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[24] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifyingdeepwalk, line, pte, and node2vec. *arXiv preprint arXiv:1710.02971*, 2017.

[25] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. Local methods for estimating pagerank values. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 381–389. ACM, 2004.

[26] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543. ACM, 2002.

[27] Xu Jia, Hongyan Liu, Li Zou, Jun He, Xiaoyong Du, and Yuanzhe Cai. Local methods for estimating simrank score. In *Web Conference (APWEB), 2010 12th International Asia-Pacific*, pages 157–163. IEEE, 2010.

[28] Ulrike V Luxburg, Agnes Radl, and Matthias Hein. Getting lost in space: Large sample analysis of the resistance distance. In *Advances in Neural Information Processing Systems*, pages 2622–2630, 2010.

[29] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[30] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[31] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

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

[33] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification.

[34] Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2024–2033, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/lei17a.html.

[35] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[36] Vladimir Batagelj and Andrej Mrvar. http://vlado.fmf.uni-lj.si/pub/networks/data/, 2006.

[37] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

[38] Robert Ackland et al. Mapping the us political blogosphere: Are conservative bloggers more prominent? In *BlogTalk Downunder 2005 Conference, Sydney*. BlogTalk Downunder 2005 Conference, Sydney, 2005.

[39] Christian Von Mering, Roland Krause, Berend Snel, Michael Cornell, Stephen G Oliver, Stanley Fields, and Peer Bork. Comparative assessment of large-scale data sets of protein–protein interactions. *Nature*, 417(6887):399–403, 2002.

[40] Duncan J Watts and Steven H Strogatz. Collective dynamics of âĂŸsmall-worldâĂŹnetworks. *Nature*, 393(6684):440–442, 1998.

[41] Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring isp topologies with rocketfuel. *IEEE/ACM Transactions on networking*, 12(1):2–16, 2004.

[42] Muhan Zhang, Zhicheng Cui, Shali Jiang, and Yixin Chen. Beyond link prediction: Predicting hyperlinks in adjacency space. In *AAAI*, 2018.

[43] Christopher Aicher, Abigail Z Jacobs, and Aaron Clauset. Learning latent block structure in weighted networks. *Journal of Complex Networks*, 3(2):221–248, 2015.

[44] Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

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

[46] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[47] Jure Leskovec and Andrej Krevl. {SNAP Datasets}:{Stanford} large network dataset collection. 2015.

[48] Reza Zafarani and Huan Liu. Social computing data repository at asu, 2009. *URL http://socialcomputing. asu. edu*.

[49] Matt Mahoney. Large text compression benchmark, 2011.

[50] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. Biogrid: a general repository for interaction datasets. *Nucleic acids research*, 34(suppl_1):D535–D539, 2006.