Link Prediction with Graph Neural Networks

Sören Lohr

University of Kassel, 34117 Kassel, Germany

Abstract. The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: Link Prediction · Graph Neural Networks · VGAE · SEAL.

1 Introduction

Consider the following applications. Recommendation systems in online-shops or social media, which analyse the different links between shopped items or users and propose new items or friends. Paper correlation in citation networks, which is an important part of recommendation when searching appropriate papers with the ever increasing amount of scientific literature. Identifying hidden links in supply chains that can impact them in an unforeseen way. Circuit board completion, which identifies missing connections, can assist in reducing the complexity of circuit board design. Drug discovery, where the interaction of different proteins significantly influences the effect of medications. These are only a few applications of what is known as link prediction. It finds wide application in the biology, medical, social or even scientific domains.

Really, link prediction is one of some useful learning tasks whenever entities interact in a structured way. As the name suggests, it tries to predict interactions between entities based on entity attributes and their relations. The information about entities and relations is most often depicted in graphs, where, in general, entities are called nodes or vertices with node attributes, and relations between entities are called edges, which in turn also have attributes. The relationships between entities are generally assumed to be incomplete concerning a given application, which gives rise to link prediction.

Traditional techniques can be differentiated by which information they base their calculations on. First, topology-based. Multiple different heuristics calculate link probabilities of two nodes based on the topology in between, often using random walks. A different type of topology-based method utilizes low-dimensional latent representations of the graph and nodes. Second, there are the content-based techniques, where the underlying assumption is that similar nodes are more likely to have links. Therefore, these techniques are based on similarity measures of node and edge features.

Recent developments in deep learning spawned graph neural networks (GNNs), which can leverage the power of deep learning and extract meaningful information of structured data. GNNs are neural networks that incorporate structural information, most often in form of the graph adjacency matrix, and consider how information passes through the graph. As such, GNNs can also be used for the link prediction task. Given enough training data, The ability of machine learning allows GNNs to handle more difficult applications compared to classical approaches. An additional benefit is that GNNs bridge the gap between topology- and content-based methods, which further improves their predictive power. For the reasons described above, state-of-the-art methods often include GNNs.

These notes presents a state-of-the-art method for link prediction with GNNs called the SEAL framework. This technique supplies additional structural and positional information about potential links, which improves the learning process and reframes the learning task from edge-level to subgraph-level. The notes also highlight an earlier method, called variational graph autoencoder (VGAE), which creates network embeddings of the graph in question with their encoder

and predicts links between graph nodes with their decoder by similarity of those embeddings. The SEAL framework has been shown to outperform VGAEs, which the notes will also explain.

The following paragraph describes how the notes present the information about the topic. Section 2 contextualizes the methods that these notes focus on. It shows related work and establishes why the presented method is state-of-the-art. Section 3 establishes the necessary background knowledge to understand the topic. The section especially touches on the basics of graph neural networks and and machine learning tasks on graphs. Next, section 4 presents the methods that these notes are concerned with. In particular, subsection 4.1 gives a detailed overview of VGAEs, while subsection 4.2 covers the SEAL framework. Subsection 4.3 focuses on the comparison of the two methods. Subsequently, section 5 briefly shows two applications of link prediction with graph neural networks. Finally, section 6 summarizes the notes.

The intro is a bit informally written but quite understandable. I am missing a research question or any other from of justification why this paper exists and which problem you want to solve and moreover how you want to solve it. Finally there is not a single citation until now so why should i believe what i am reading?

2 Related Work

3 Background

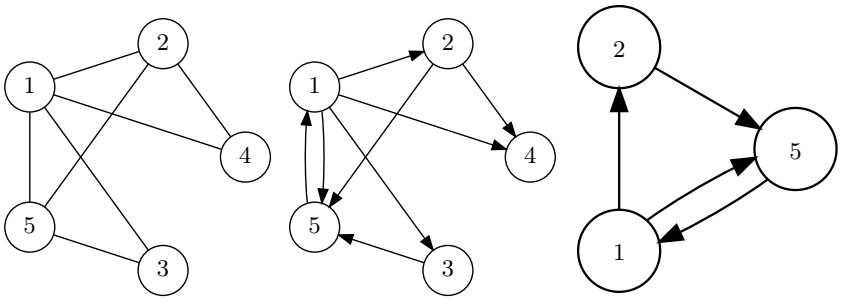
Section 3 covers the basic knowledge needed and definitions to understand the SEAL framework and VGAEs. It briefly touches on graph theory, link prediction, and graph neural networks.

The section first recounts some definitions of graphs and their properties. It only gives definitions for simple graphs, as they are the subject of the presented methods.

Definition 1. *A **graph** is a pair $G = (V, E)$, where V is an ordered set of vertices or nodes, denoting V_i for the ith vertex, and $E \subseteq V^2$, where $V^2 = \{(x, y) \mid x, y \in V\}$.*

Definition 2. *A graph G is called **undirected**, if $(x, y) \in E \iff (y, x) \in E$ for every $x, y \in V$, and **directed** if not.*

Definition 3. *a **subgraph** of a graph G is a pair $H = (V_H, E_H)$, where $V_H \subset V$ and $E_H = \{(x, y) \mid x, y \in V_H, (x, y) \in E\}$*



(a) Undirected graph. (b) Directed graph. (c) Subgraph of 1b.

Fig. 1: Graph visualizations.

Definition 4. *The **neighborhood** of a node v in a given graph G is defined as $\mathcal{N}(v) = \{x \in V \mid (v, x) \in E \vee (x, v) \in E\}$*

Definition 5. *An **adjacency matrix** of a graph G is a matrix $A \in \mathbb{R}^{V \times V}$, where $a_{ij} \neq 0$ if $(V_i, V_j) \in E$, else 0.*

The adjacency matrix plays an important role for graph neural networks. It allows for an efficient computation of information propagation in the graph neural network by using matrix multiplications.

Definition 6. *A **feature space** is a set of values, for example $\mathbb{N}_0$, $C = \{\emptyset$, 'red', 'blue'$\}$ or $\mathbb{N}_0 \times C$.*

Definition 7. *A **node attribute** or **feature function** $w : V \to X$ assigns each vertex of a graph G to an element of a feature space X. Similarly, an **edge attribute** or **feature function** $u : E \to Y$ assigns each edge of a graph G to an element of a feature space Y.*

Node and edge attributes or features also play an important role for GNNs and other methods that utilize similarity or network embeddings. VGAEs for example use node features to create the network embeddings. The SEAL framework also employs a network embedding procedure.

3.1 Machine learning tasks on graphs

Because graphs have a different structure than images or tabular data, machine learning tasks generally focus on different levels.

Definition 8. *A **graph-level** task concerns a graph as a whole.*

Typical graph-level tasks are graph classification or creating graph embeddings.

Definition 9. *A **node-level** task concerns the individual nodes or node features of a graph.*

The most prominent node-level tasks are node classification, predicting node features or creating node embeddings.

Definition 10. *An **edge-level** task concerns the individual existing or nonexisting edges or edge features of a graph.*

Edge-level tasks include edge classification, edge feature prediction and edge or link prediction.

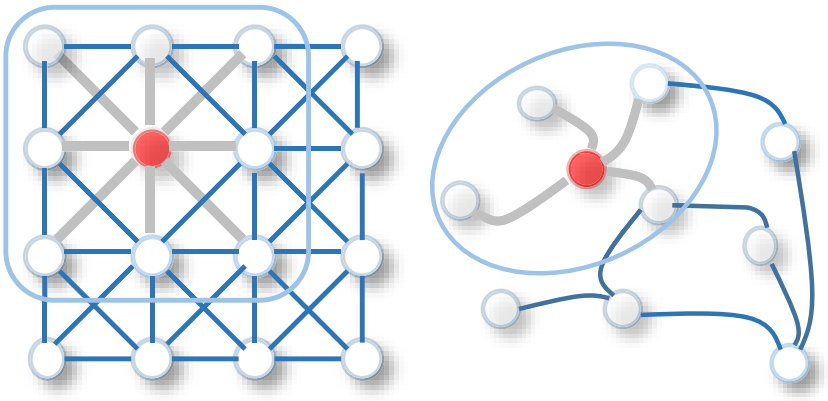
Definition 11. ***Link prediction** is an edge-level task, where the goal is to predict the likelihood that two nodes are connected with an edge.*

evaluation methods? AUC/AP

3.2 Graph Neural Network

In general, the goal of a GNN is to create a representation of the graph or nodes and edges, which can be used in further typical machine learning tasks, e.g. classification of graphs or nodes.

Similar to convolutional neural networks (CNNs), GNNs assume a local interdependency of neurons, with the important distinction that the interdependency of neurons in GNNs is irregular. The convolutional layer of a CNN will aggregate all neurons within a local receptive field (or a kernel) - this field can be understood as a graph, where the neurons are nodes that are connected with all adjacent neurons. The output of the convolutional layer is then simply the aggregation of node features that share an edge. GNNs generalize this operation into the so-called graph convolution, where the requirements on connections within



(a) Typical convolution in a CNN. (b) Graph convolution.

Fig. 2: Comparison of CNN and GNN convolutions as shown in [6].

the convolution are relaxed. Figure 2 shows how the input of a CNN compares to a graph.

There are many different types of GNNs which incorporate various machine learning ideas, for example convolutional GNNs, recurrent GNNs Graph Autoencoder (GAE), or Graph attention networks (GATs). For a more in-depth overview, refer to [6] and [4]. In general, all of these GNNS can be used in link prediction tasks.

These notes focus on so-called **message passing neural networks** (MPNN), which is a generalization of convolutional GNNs, and on a popular convolutional GNN, called Graph Convolutional Network (GCN). the GCN can also be formulated as an MPNN.

There are essentially two convolutional GNN types, spectral-based and spatial-based GNNs. Without going into much detail, spectral-based GNNs use the fact that the convolution of two functions can be expressed as multiplication after a transformation into a spectral domain. They use the graph Fourier transform, which is based on the eigenvectors of the adjacency matrix. To calculate the output of the convolution, a learnable filter is multiplied with the transformed node features in the spectral domain. The inverse graph Fourier transform is then applied to the product. Spatial-based GNNs operate much more like CNNs, where an aggregation function passes over the neighborhood of a node with learnable weights and an activation function to calculate the convolution.

Definition 12. *A **Graph Convolutional Network** (GCN) is defined as follows:*

$$\mathbf{H}^{(l)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l-1)}\right) \tag{1}$$

with $\mathbf{H}^{(0)} = \mathbf{X}$*, where* $\mathbf{X}$ *is the feature matrix of all nodes,* $\mathbf{H}^{(l)}$ *is the hidden representation,* $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ *is the transformation matrix called the normalized Laplacian and* $\mathbf{W}^{(l)}$ *is the filter as a learnable weight matrix.*

TODO: Explain Laplacian and degree matrix in graph section (adjacency matrix) TODO: Word2vec here or in methods? -> "embeddings" section. besser in SEAL erwähnen

The GCN bridges the gap between spectral-based and spatial-based convolutional GNNs. Spectral-based GNNs are computationally expensive. Simplifications and approximations of spectral convolutions to make them more efficient lead to the form described in the definition above. The convolution applied in the GCN then closely resembles that of a spatial-based GNN. For more details, refer to [6].

As mentioned above, MPNN is a framework that generalizes the convolution of such convolutional GNNs. TODO: switch with GCN

Definition 13. *A* ***graph convolution*** *in an* ***MPNN*** *takes the following general form:*

$$h_u^{(l)} = \phi_l \left(h_u^{(l-1)}, \bigoplus_{v \in \mathcal{N}(u)} \psi_l \left(h_u^{(l-1)}, h_v^{(l-1)} \right) \right) \tag{2}$$

with $h_u^{(0)} = x_u$*, where* x_u *is the feature vector of node* u*,* $h_u^{(l)}$ *is the hidden representation produced by the convolution* l*,* ϕ_l *is a function with learnable parameters, typically an activation function,* ψ_l *also a function with learnable parameters, usually multiplication, and* $\oplus$ *some form of aggregation like sum, mean or max.*

While the definition of the GCN is in matrix form, one can see that by choosing the sigmoid activation function for ϕ_l, ignoring the first argument, and the multiplication with a weight matrix for ψ_l, the MPNN takes the form of a GCN. The adjacency matrix is accounted for in the aggregation function, iterating over the neighborhood of the node u.

These graph convolution layers can be chained to create deep networks. Similar to CNNs, other operations such as pooling, aggregation or dropout can also be applied in a GNN.

4 Methods

4.1 Variational Graph Autoencoders

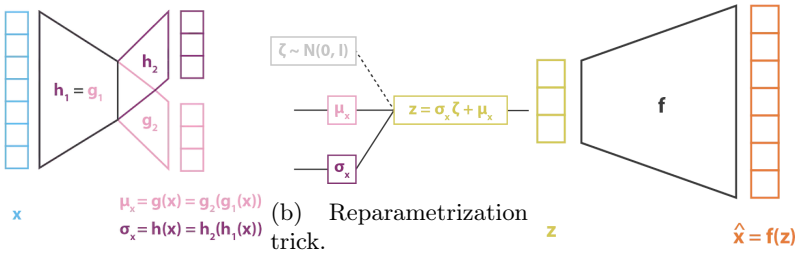
In [2], the authors Kipf and Welling apply the principle of variational autoencoders in combination with GNNs to solve the problem of link prediction. Specifically, they try to predict the existence of links in three citation network benchmark datasets for link prediction tasks, namely Cora, Citeseer and Pubmed. Table 1 gives a brief overview.

Table 1: Citation network datasets used in [2], source [7].

Dataset	Classes	Nodes	Edges
CiteSeer	6	3,321	4,732
Cora	7	2,708	5,429
PubMed	3	19,717	44,338

Recall that (variational) autoencoders are special neural networks that can be decomposed into encoder, decoder and bottleneck. The goal of an autoencoder is to create low-dimensional latent representations of the input. It does so by reconstructing the input, thus the input dimension equals the output dimension. In the bottleneck, a layer in the autoencoder that typically has a lower neuron number than the input or output, a data compression takes place, which results in said low-dimensial latent representations. In case of a variational autoencoder, the latent representations are distribution parameters, from whose distributions latent samples are drawn for the decoder.

VGAEs employ the same strategy, however for the encoder, one would use GNNs. The specific model structure used in [2] is shown in Figure 3.



(a) VGAE Encoder. (b) Reparametrization trick. (c) VGAE Decoder.

Fig. 3: Encoder of the chosen model in [2], source [3].

It employs a two layer GCN for mean μ and standard deviation σ, described in equation 3.

$$\mathrm{GCN}_{\sigma|\mu}(\mathbf{A}, \mathbf{X}) = \tilde{\mathbf{A}}\mathrm{ReLU}\left(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W_0}\right)\mathbf{W_1}, \tag{3}$$

where $\mathbf{W_0}$ is shared by both GCN_μ and GCN_σ. They use the reparametrization trick, which pulls the random sampling out of the network. This trick enables the usage of backpropagation. Their decoder calculates the reconstruction of the normalized adjacency matrix $\hat{\mathbf{A}}$ by applying a sigmoid function to the dot product similarity of the sampled latent representation of each of the nodes $\mathbf{Z}$.

$$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^\mathbf{T}\right) \tag{4}$$

They also propose a non-probabilistic version (GAE), where the encoder directly calculates a fixed latent representation.

$$\mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A}) \tag{5}$$

Since there are unknown latent variables and random distributions involved in the learning process, it is customary to optimize the variational lower bound $\mathcal{L}$ w.r.t. to the parameters $\mathbf{W_i}$, which is expressed as follows.

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}|\mathbf{Z})] - \mathrm{KL}[q(\mathbf{Z}|\mathbf{X}, \mathbf{A})||p(\mathbf{Z})], \tag{6}$$

where $q(\mathbf{Z}|\mathbf{X}, \mathbf{A})$ is the distribution of the sampled output of the decoder, $p(\mathbf{A}|\mathbf{Z})$ is the distribution of the reconstruction based on the sampled output of the decoder and $p(\mathbf{Z})$ the prior distribution of the sampled output. In [2], they chose an isometric gaussian prior $p(\mathbf{Z}) = \prod_i \mathcal{N}(\mathbf{z}_i|0, \mathbf{I})$.

In terms of evaluation of their method, they split the datasets into training, validation and test graph with 85%, 5% and 10% of edges respectively, where the validation graph is used to tune hyperparameters and the test graph is used for final evaluation. They chose a 32 dimensional hidden layer and 16 dimensions for the latent representations $\mathbf{Z}$. They proceed to train their model for 200 epochs, using the Adam optimizer with a learning rate of 0.01 on the training graph as input.

They mainly compare the ability of the VGAEs and GAEs to produce embeddings, so for their baselines they employ the same method of reconstructing the adjacency matrix, shown in equation 4. As baselines, they chose two other methods with which they create the node embeddings $\mathbf{Z}$. Spectral clustering (SC), which takes a portion of the eigenvectors of the Laplacian as node embeddings, and DeepWalk, which creates a number of random walks of length t and applies word2vec. Important to note is that those baselines do not include the node features, as opposed to the GCN.

As results, they report AUC and AP scores for each model on the dataset, performing multiple runs with random initializations and showing the average and standard deviation. VGAE mainly outperforms the other methods with a score of around 91.5, with the exception of the PubMed dataset, where GAE has a score of around 96.

They conclude that adding node features significantly helps the predictive power and that pairing a gaussian prior with a dot product decoder might be disadvantageous, as the decoder generally tries to move away from 0.

4.2 SEAL

SEAL, which stands for learning from Subgraphs, Embeddings and Attributes for Link prediction, is not a model per se but a general framework. The authors of [2] have already shown that including node features significantly improves the predictive power of models. The authors of [8] take it further by including additional information like structural knowledge.

Additionally, they show that topological information is only effective up to a certain node distance. Heuristics have been used in the past for the link prediction task. They rely on topological information by calculating scores of distance of nodes, often by random walks from one node to another. The higher the distance, the higher the order of the heuristic. By generalizing popular higher order heuristics like Katz index, PageRank or SimRank into what they call a γ-decaying heuristic, they show that the information that can be transmitted by nodes that are far away exponentially decreases. The generalization shows that these higher order heuristics really only have an effective order that is much lower.

Definition 14. *A* ***γ-decaying heuristic*** *is given by the following equation.*

$$\mathcal{H}(x, y) = \eta \sum_{l=1}^{\infty} \gamma^l f(x, y, l), \tag{7}$$

where γ is a decaying factor between 0 and 1, η a positive constant or a positive function of γ that is upper bounded by a constant, f is a nonnegative function of x, y, l under the given network.

This finding is important for their first addition to the framework. By showing that topological information is only effective up to a certain distance, or hops, they propose to use subgraphs that encapsulate this distance instead of the entire graph. GNNs, which take the adjacency matrix $\mathbf{A}$ and the node information matrix $\mathbf{X}$ of those subgraphs, can automatically learn an appropriate heuristic that best describes the formation of those links. This is also the point where the authors re-frame link prediction from an edge-level task to a graph-level task by making use of so-called enclosing subgraphs.

Definition 15. *An* ***enclosing subgraph*** *is a subgraph $G^h_{x,y} = (V^h_{x,y}, E^h_{x,y})$ of graph $G = (V, E)$ given two nodes $x, y \in V$, where $V^h_{x,y} = \{i \in V \mid d(i, x) \leq h \vee d(i, y) \leq h\}$*

So the enclosing subgraph includes all nodes that have a maximum distance of h from either node x or y. Enclosing subgraphs have the important property that they focus the link prediction on one specific link in the graph, namely the

link between x and y. Instead of reconstructing an entire adjacency matrix, the model just needs to predict the existence of one link, which effectively transforms the link prediction into a graph classification task.

Definition 16. *Given a graph $G = (E, V)$, the set of **sampled positive training links** is a set $E_p \subset E$ which serves to train the model on existent links. The set of **sampled negative training links** is a set $E_n \cap E = \emptyset$ which serves to train the model on nonexistent links.*

The authors will later on create a dataset of subgraphs containing positive training links and negative training links to properly train the model on the classification task.

The second important addition is topological node labeling. The goal is to include information about the role of a node in the subgraph and what the relative position to other nodes is. This especially helps differentiate positions of nodes that might be otherwise indistinguishable from a topological viewpoint. One can imagine that nodes x and y of the enclosing subgraph hold a more important role than nodes far away from those, because these two nodes are the subject of whether there is an edge or not. So, each node gets assigned a number depending on how far away the node is to the center nodes x and y. Nodes that have a same distance will receive the same label, while the center nodes have the distinct label of 1. The authors employ a certain node labeling algorithm called the doulbe radius node labeling.

Definition 17. *The **double radius node labeling** is a labeling algorithm with the following rules, where the double radius is $(d(i,x), d(i,y))$.*

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

with a function to directly calculate the node label

$$f_l(i) = 1 + \min(d_x, d_y) + (d/2)[(d/2) + (d\%2) - 1], \tag{8}$$

where $d_{x|y} = d(i, x|y), d = d_x + d_y$, $f_l(i)$ a labeling function $V \to \mathbb{N}$.

In other words, when the sum of distances in the double radius of a node i is less than j, node i receives a lower label. If they are equal, the product of the distances in the double radius decides which receives a lower label. Figure 4 illustrates an example.

To explain, nodes with distance $(1, 1)$ receive label 2, because nodes with distance $(0, 1)$ or $(1, 0)$ have to have a lower label. Besides, only the center nodes can have these distances. Looking at distance 3, there are two possibilities for the double radius, $(1, 2)$ or $(2, 1)$, therefore they receive label 3. Nodes that have a switched double radius actually count has being on the same radius, so they also receive the same label. Moving on to distance 4, there are actually three

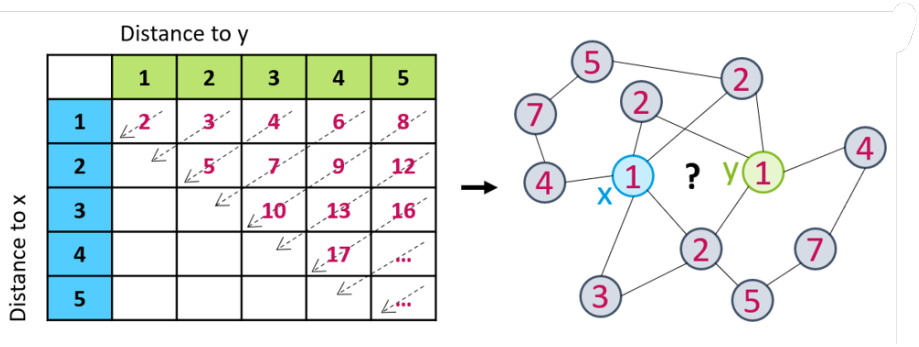


Fig. 4: Example of double radius node labeling [1].

possibilities, $(1, 3)$ or $(3, 1)$ and $(2, 2)$. $(1, 3)$ and $(3, 1)$ receive the same label, and because $1 \cdot 3$ is less than $2 \cdot 2$, they also receive the lower label compared to $(2, 2)$, so 4 and 5 for $(2, 2)$.

The node labels are then one-hot encoded and serve as the node information matrix $\mathbf{X}$ that is an input to the GNN. Last, similar as in [2], they include the node features as an input to the GNN. Since the GNN serves a different purpose in SEAL, they use node2vec as an embedding algorithm for the node features and concatenate the node embeddings into the aforementioned node information matrix. However, they found that simply generating node embeddings from the graph itself hurts the generalization performance of the model. Since these node embeddings also incorporate topological information, the existence of links is also incorporated. The GNN exploits this knowledge that is baked into the embeddings and overfits on the subgraphs in the training set. To remedy this exploitation they add the sampled negative training links temporarily into the graph and create the embeddings afterwards. The authors call this trick negative injection.

To summarize, the SEAL framework functions the following way.

1. Sample positive and negative training links.
2. Perform negative injection and create node embeddings for the given graph as the node information matrix.
3. Create dataset of enclosing subgraphs from training links.
4. Create one-hot encoded node labels for each of the subgraphs and concatenate them to the node information matrix of the subgraph.
5. Optimize a GNN model on the training set.

The authors evaluate their framework with a wide variety of datasets and baselines. They prepare their data by removing 10% of positive test links and sampling 10% of negative test links. The rest of the links are used for training. They report the mean and standard deviation of AUC and average precision in 10 runs with random initialization. They use the DGCNN [5] as a GNN model and node2vec as an embedding algorithm, however the selection is only of

secondary importance to the SEAL framework, as any GNN model or embedding algorithm principally works with the framework. They also restricted the choice of the hyperparameter for the hop-distance h to 1 or 2 to validate their theoretical findings. Table 2 gives an overview of the used datasets. For more information about the datasets, refer to the appropriate citations in [8], Appendix C.

Table 2: Citation network datasets used in [8].

Dataset	**Nodes**	**Edges**	**Node degree**
USAir	332	2,216	12.81
NS	1,589	2,742	3.45
PB	1,222	16,714	27.36
Yeast	2,375	11,693	9.86
C.ele	297	2,148	14.46
Power	4,941	6,594	2.67
Router	5,022	6,258	2.49
E.coli	1,805	14,660	12.55

First, they compare against heuristic methods like Katz index, PageRank, SimRank etc., where they leave out the node information for their method, since the heuristics only depend on the graph structure. The SEAL method with their chosen hyperparameters and default models and algorithms outperform the baselines across the board, with exceptions on the USAir and C.ele datasets. The different between methods in these exceptions is around 0.01 AUC, whereas on the other datasets, the SEAL framework produces an AUC score of around 96.5.

Second, they test against baselines that also include latent features. Amongst others, they chose spectral clustering and also the VGAE. Like with the heuristic methods, the chosen baselines do not include explicit node features, therefore they excluded those for their methods. Again, SEAL outperforms the other methods on almost all datasets with one exception.

From their experimental evaluation, they conclude that specific, learned heuristics are superior to the more general heuristics, showing that the subgraph approach yields better results. Additionally, the usage of GNNs significantly increases the predictive power compared to kernel-based or fully-connected neural networks. From their experiments with latent feature baselines, they conclude that

4.3 Comparison

This section compares the two methods presented in the last two sections. A major difference is that SEAL is a framework, while VGAEs is a model architecture. As such, VGAEs, albeit with slight modifications, could also be employed as a GNN model in SEAL. Another important distinction is the reframing of

the link-prediction task. In VGAE, the link prediction is formulated as an edge-level task, reconstructing the adjacency matrix of a graph. In SEAL, the link prediction is transformed into a graph-level classification task, deciding whether a subgraph contains an edge between the two center nodes or not. Already mentioned in the previous section, in the experimental evaluation of SEAL in [8], the authors have chosen the VGAE as one of the baselines. They showed that their method actually outperforms VGAEs on a variety of graphs with different properties.

Due to the differences described above, in [8] where only the performance was tested, the comparison of a model architecture and a framework seems to be skewed. The authors used a different model architecture, DGCNN, which uses different, more up-to-date concepts than the VGAE. It could be entirely possible that the DGCNN alone is already superior. In VGAE, the encoder creates node embeddings, while in SEAL, this task is done by the node2vec algorithm. The purpose of the GNN is entirely dedicated to the classification, while in VGAE, the prediction is performed by a simple dot product similarity. From this standpoint, it is not surprising that node2vec and DGCNN outperforms VGAE. It would have been interesting to also see a performance comparison of the VGAE and node2vec plus DGCNN without the framework in some form of ablation study.

That being said, the benefit of SEAL is the added structural, latent, and explicit information, which, as shown, is in no doubt an improvement to any model that can make use of this information. The authors actually went ahead and extensively studied the comparison of VGAE and SEAL. They have worked out a variety of reasons why SEAL attains a better performance than VGAE. Most importantly, they have shown in previous works and explained in [9] that VGAEs actually can not incorporate some aspects of SEAL. For example structural information aka the node labels, even though they would really need them due to the way VGAE constructs the node embeddings. Figure 5 shows an example graph where the node embeddings of VGAE fall short.

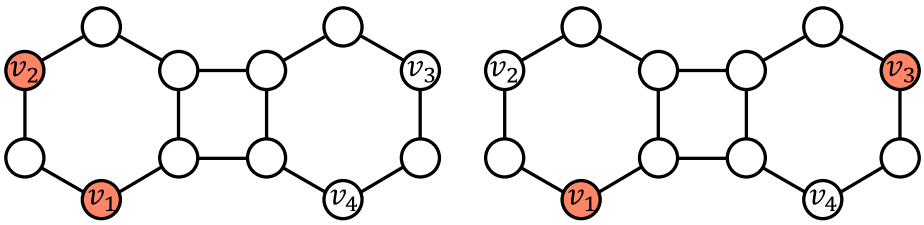


Fig. 5: Shortcomings of VGAE node embeddings.

Looking at node $v2$ and $v3$, they are structurally the same - no matter the chosen h distance for the embedding, both nodes have the same neighborhood. If there are no node features involved, or the node features are the same, the VGAE will create the same embeddings for both nodes due to the structural

similarity. Since the embeddings can not distinguish between the two nodes, this means that a VGAE will assign the same possibility of the existence of a link $(v1, v2)$ and $(v1, v3)$. The authors of SEAL argue that, yes, with node features this problem would not arise, however it would hurt the VGAE's inductive learning capability. This basically means that the models are no longer permutation invariant, resulting in an inability to assign the same structural embeddings to isomorphic links ($(v1, v2)$ and $(v3, v4)$). This inability diminishes the generalization power of the VGAE greatly. A key difference to SEAL is that we can not use the same double radius node labeling in a normal VGAE, because the double radius node labeling is assigning roles to the nodes regarding a specific link, however the VGAE is focused on all links at once.

In summary, SEAL is clearly the more advanced technique, shown by the authors of SEAL, being more flexible in the selection of models, embedding algorithms, and choice of information, which rightly makes it a state-of-the-art method.

In the definitons you said there are many different types of GNNs (convolutional, recurrent, AE, graph attention) for link prediciton. why are you not considering all of these methods in your paper? This could be explained in a research question.

5 Applications

6 Conlusion

References

1. Calemsy: How to determine the index of the nodes in the enclosing subgraph?, https://github.com/muhanzhang/SEAL/issues/8, accessed: 2024-08-03
2. Kipf, T.N., Welling, M.: Variational graph auto-encoders (2016), https://arxiv.org/abs/1611.07308
3. Rocca, J.: Understanding variational autoencoders (vaes), https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73, accessed: 2024-08-03
4. Thomas, J.M., Moallemy-Oureh, A., Beddar-Wiesing, S., Holzhüter, C.: Graph neural networks designed for different graph types: A survey (2023), https://arxiv.org/abs/2204.03080
5. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph cnn for learning on point clouds (2019), https://arxiv.org/abs/1801.07829
6. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems **32**(1), 4–24 (2021). https://doi.org/10.1109/TNNLS.2020.2978386
7. Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings (2016), https://arxiv.org/abs/1603.08861
8. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018), https://proceedings.neurips.cc/paper_files/paper/2018/file/53f0d7c537d99b3824f0f99d62ea2428-Paper.pdf
9. Zhang, M., Li, P., Xia, Y., Wang, K., Jin, L.: Labeling trick: A theory of using graph neural networks for multi-node representation learning. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems. vol. 34, pp. 9061–9073. Curran Associates, Inc. (2021), https://proceedings.neurips.cc/paper_files/paper/2021/file/4be49c79f233b4f4070794825c323733-Paper.pdf