Studying Explain-ability and Comparing Graph Neural Network models for Link prediction task

Abdullah Khan, Sahely Bhadra

Abstract

Link prediction has been an area of extensive research for applications built over graph-based databases. The problem of link prediction is not so new and several algorithms have been studied in this direction. However, the use of Graph Neural Networks is a novel technique that makes these solutions much more generalizable and at the same time make them more scalable. Given that the use of GNNs to encode the nodes of a graph is based not only on its features but also the topology of the graph itself, we are able to leverage the already known autoencoder framework to predict missing links. We compare various graph neural network architectures, to determine the affect they might as encoders for link prediction task. Node embeddings [1] are extremely useful as features for link prediction, where the goal is to predict missing edges, or edges that are likely to form in future.

Introduction

Link prediction has been an area of extensive research for applications built over graph-based databases. While primarily link prediction lies at the core of most recommender system [2], notable implementations include social network-based relationship prediction [3] (suggested friends, people you may want to follow etc.); biomedical implementations (drug-protein [4], protein-protein interactions [5] etc.). Various methods for link prediction have been studied, these include heuristic methods, embedding methods, and feature-based methods. Heuristic methods compute some heuristic node similarity scores as the likelihood of links [6], such as common neighbors, preferential attachment [7], and Katz index [8]. Recently GNNs have emerged as powerful tools for representation learning on various downstream machine learning tasks [1] including node classification, graph classification and link prediction. Various GNN architectures have been proposed which use a variation of such techniques to find node embeddings. We evaluate the effect of such models on link prediction task; the models used are GCN [9], GAT [10](attention mechanism in GNNs) and GraphSAGE [11] (inductive learning). Using an autoencoder framework

[12] we compare these different models, while also comparing GAE and VGAE [12] variations of the framework. With the advances of deep learning, current link prediction methods commonly compute features of the nodes of the complete graph. Such pairs of embeddings are then mapped to a singular value assigned to that link. After such a transformation, the link prediction task reduces to a classification task with sample points coming from the existing links (positive links) and a sample of non-existent links (negative links). The transformations that map a pair of embeddings to a single embedding, vary with most commonly used technique being Inner Product, other decoding techniques based on tensor factorization also exists such as DEDICOM [13]and RESCAL [13].

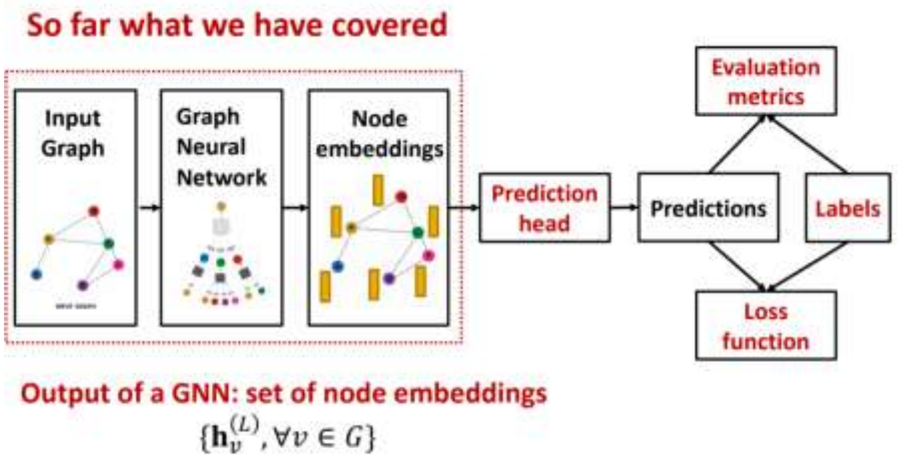


Figure 1: shows a general GNN pipeline. The loss function can be used for controlling both the node embeddings as well as prediction head generation

Our first experiment is to see the effects of using different GNN architectures for generating these node embeddings. We will compare some of the most common GNN architectures used which differ from each other not only in their way of aggregation of surrounding embeddings, but may also choose to operate in an inductive manner. The idea behind any graph neural network is that the computation graph formed by the neighbors of a node represent the topology around it and are influence in the properties of that node. Thus, we want to convolve the information of the surrounding neighbors of a node captured by its computation graph, and aggregate such information in order to encode the node such that similar nodes end up near each other in the high-level embedding space. This process involves generation of node embeddings (**Message Passing**) based on local network neighbors and then aggregating (**Aggregation**) it using a suitable function. Thus, a GNN model is completely defined by its Message Passing and aggregation scheme. Following is the MP + A schemes of various models being compared as our experiment.

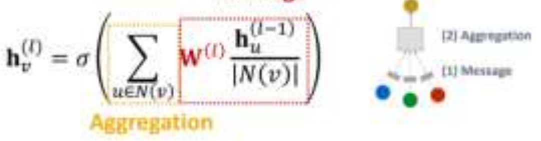
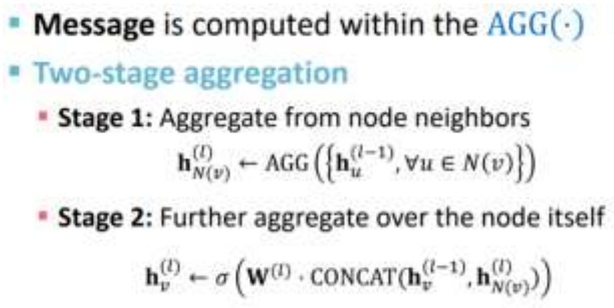

Message
$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$
Aggregation
(2) Aggregation
(1) Message

Figure 2: shows the GCN layer. $h_v^{(l)}$ represents the embedding of node v for the layer l. $h_u^{(l-1)}$ represents the embeddings of the nodes on the neighborhood of v calculated in the previous layer.

GCN uses a normalized message passing scheme, which is normalized by the node degree. While its aggregation function is simple summation of all these messages. GraphSAGE uses a generalized aggregation scheme which is taken as an input from the user and also uses self-embedding for final aggregation. This user input may be any aggregation scheme such as MEAN, Pool, or LSTM, or any other custom defined scheme.



- **Message** is computed within the AGG(·)
- Two-stage aggregation
  - **Stage 1:** Aggregate from node neighbors

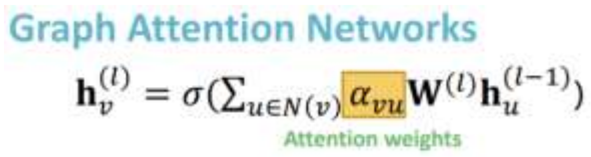
$$\mathbf{h}_{N(v)}^{(l)} \leftarrow \mathrm{AGG}\left(\left\{\mathbf{h}_u^{(l-1)}, \forall u \in N(v)\right\}\right)$$

  - **Stage 2:** Further aggregate over the node itself

$$\mathbf{h}_v^{(l)} \leftarrow \sigma\left(\mathbf{W}^{(l)} \cdot \mathrm{CONCAT}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_{N(v)}^{(l)})\right)$$

Figure 3: GraphSAGE Message Passing and Aggregation scheme. Aggregation happens in two passes, and is concatenated afterwards. L2 normalization may be applied after every layer in case embedding vectors have different scales.

GAT adds the idea of attention weights to the aggregation schemes, by assigning different learnable importance factor (α_{uv}). GATs also allow for various aggregation functions, but the most commonly used one is SUM and thus, we will also use the same in our experiments.



Graph Attention Networks

$$\mathbf{h}_v^{(l)} = \sigma\left(\textstyle\sum_{u\in N(v)} \underbrace{\alpha_{vu}}_{\text{Attention weights}} \mathbf{W}^{(l)}\mathbf{h}_u^{(l-1)}\right)$$

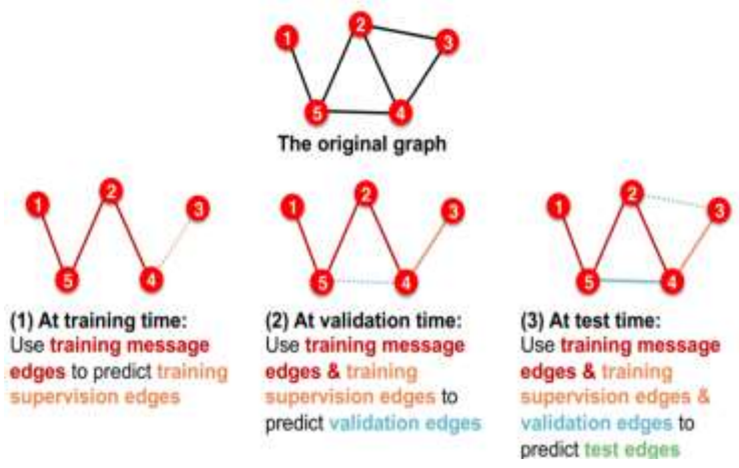
Figure 4: shows a general GAT layer. The attention weights are parametrized and many modern GAT implementations use multi-head attendtions.

Current Work

Our current implementation involves splitting the dataset into train and test edges. We need to hide some of the edges from the GCN and let it predict if those edges exist. We divide the given edges in a graph into two sets, *Message Edges* and *Supervision Edges* where message edges are used in message passing of the GCNN algorithm, while supervision edges are used for supervision of edge predictions made by the GNN model while in the training phase. They are kept away from the model, and are further split into train / test / val edges which make the datapoints of the classifier. We use a **transductive** [11] approach in this dataset split, i.e., the entire graph topology is fed to the encoding model as opposed to the **inductive** [11] approach which feeds separate graphs as train and test sets. This process is presented in a concise manner by Figure 5.

Negative Samples

Now, we have got a well split dataset with links that actually exist. However, at the very least we are training a binary classifier to predict the edges from the non-edges (the negative samples of our datapoints). We do so by taking one end point from the present link and match it with another node such that the link does not exists in our original graph in order to produce a negative sample for our classifier. We generate same number of negative links as there as positive in order to remove the hassle of dealing with skewed data.



The original graph
(1) At training time: Use training message edges to predict training supervision edges
(2) At validation time: Use training message edges & training supervision edges to predict validation edges
(3) At test time: Use training message edges & training supervision edges & validation edges to predict test edges

Figure 5: The dataset is split in a total of 4 types of edges:

1. training message
2. training supervision
3. validation
4. test.

The number of edges known to the GNN model keeps on increasing as the supervision edges become known to the model after the training stage.

Auto-encoder

The traditional autoencoder [14] is a neural network that contains an encoder and a decoder. The encoder takes a data point X as input and converts it to a lower-dimensional representation (embedding) Z. The decoder takes the lower-dimensional representation Z and returns a reconstruction of the original input X-hat that looks like the input X.

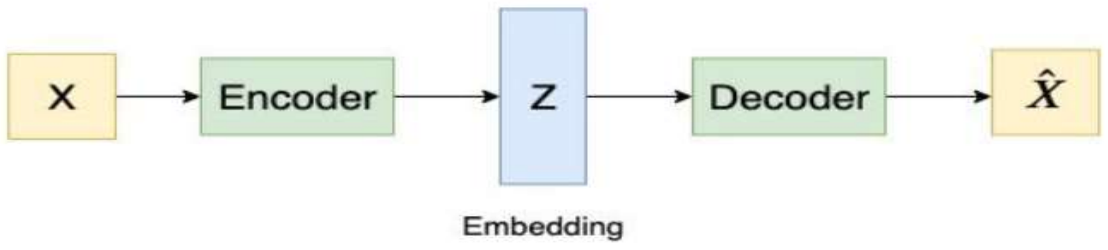


Figure 6: A general autoencoder framework

The main idea of a variational autoencoder is that it embeds the input X to a distribution rather than a point. And then a random sample Z is taken from the distribution rather than generated from encoder directly.

Experiments

We take the Cora Dataset [15] for our preliminary testing which is consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

To analyze the structural properties of of our datasets, we compare following network characteristics:

- Degree Distribution
- Hop distribution

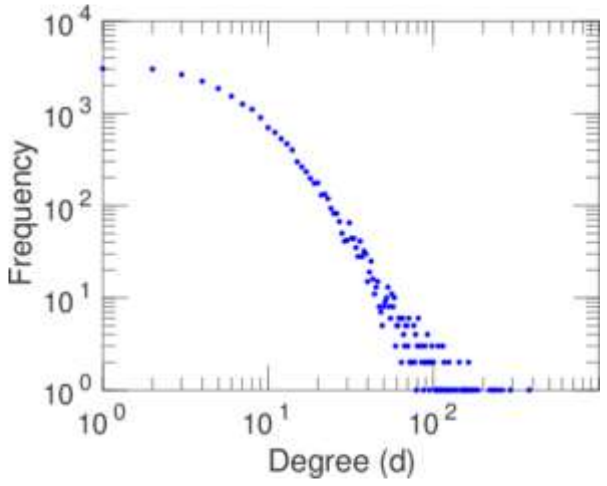


Figure 7: Degree Distribution of Cora Dataset

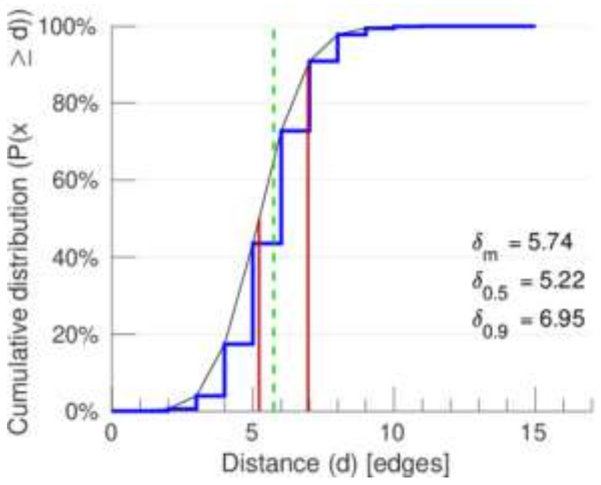


Figure 8: Hop Distribution of Cora Dataset

We also analyze our results on the PubMed Dataset, which consists of 19717 scientific publications from PubMed database pertaining to diabetes classified into one of three classes. The citation network consists of 44338 links. Each publication in the dataset is described by a TF/IDF weighted word vector from a dictionary which consists of 500 unique words.

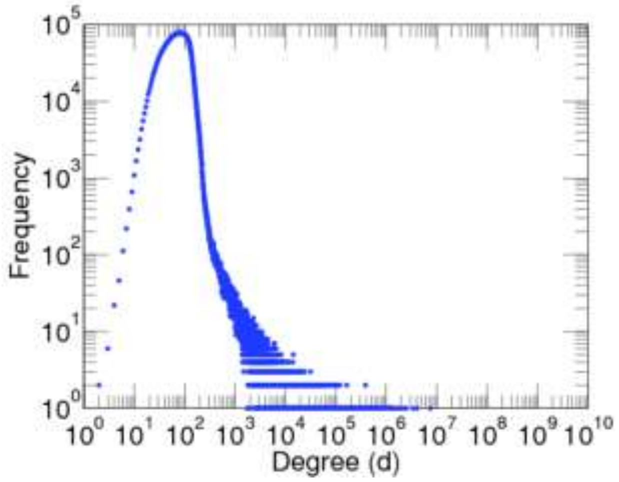


Figure 9: Degree Distribution of PubMed Dataset

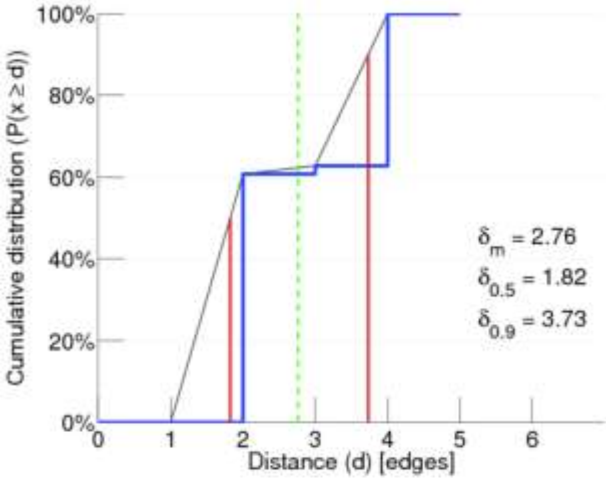


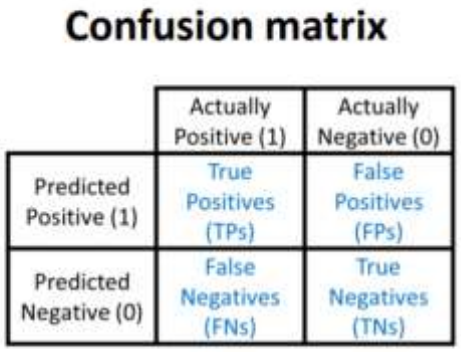
Figure 10: Hop Distribution of PubMed Dataset

From the above graphs, we can see that the PubMed dataset is not only larger but is also denser. The mean distance of the graph is 2.76 edges. Whereas, the Cora dataset is sparser with a mean distance of 5.74 edges. We have chosen these datasets to represent the two broad types of graph datasets found in real world – sparse graphs and dense graphs.

The dataset is split in the ratio of 0.55 : 0.15 : 0.3 ≡ train : val : test. We run the model for 200 epochs and evaluate its performance on the maximum test set value achieved by the model for the metrics: AUCROC and average precision (AP). We also measure the recall rate on the final test set as well as produce a confusion matrix for each of the models.

Metrics

AUC ROC: captures the tradeoff in TPR and FPR as the classification threshold is varied for a binary classifier. It gives us the intuition of the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.



Confusion matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Precision (P): $$\frac{TP}{TP + FP}$$

Recall (R): $$\frac{TP}{TP + FN}$$

Precision: is a metric that quantifies the number of correct positive predictions made. Precision, therefore, calculates the accuracy for the minority class.

Recall: gives us a performance indication of the model on predicting actual links correctly. It quantifies the number of correct positive predictions made out of all positive predictions that could have been made. Unlike precision that only comments on the correct positive predictions out of all positive predictions, recall provides an indication of missed positive predictions.

Maximizing precision will minimize the number false positives, whereas maximizing the recall will minimize the number of false negatives

We run tests over both datasets, for all the models as well as the two frameworks; GAE & VGAE. The above stated metrics are calculated and averaged over 10 runs with the following hyperparameters:

Trainer		
Lr: 0.01	Epochs: 200	Weight decay: 0.0005
Models: GCN, GAT, GraphSAGE		
Num_layers: 2	Hidden_channels: 16	Droprate: 0.2

Table 1: Hyperparameters

Results

Table 2 shows the results obtained by performing the experiment on the Cora Dataset. The scores of the three models are pretty similar, thus indicating that the choice of model does not make much difference. The models find different embeddings however, the decoder combines the embeddings and the whole end-to-end system is optimized by the Binary Cross Entropy Classification loss. The difference between VGAE and GAE scores are less than 0.03 % of each other with either performing better in some cases. Thus, only GAE scores have been shown, as it is usually the more commonly used framework.

Model	**AUC ROC**	**AP**	**Recall**
GCN	88.03 ± 0.02	87.85 ± 0.00	61.15 ± 0.02
GAT	87.98 ± 0.00	88.13 ± 0.00	68.27 ± 0.03
GraphSAGE	90.39 ± 0.01	92.12 ± 0.01	82.82 ± 0.01

Table 2: Metrics across various models for Cora Dataset.

Table 3 shows the results obtained by performing the experiment on the PubMed Dataset. The scores of the three models are vary quite a lot in case of GCN and we see a detrimental effect of using other models. However, the point to note is that recall scores point a different picture, with GCN suffering a disadvantage. Thus, we cannot rely on a single metric, neither a combination. We should choose a metric which is suited for the final task, giving a high accuracy for the solutions which gives desired results. Again, the difference between VGAE and GAE scores are less than 0.01 % of each other with either performing better in some cases; thus, only GAE scores have been reported.

Model	**AUC ROC**	**AP**	**Recall**
GCN	90.63 ± 0.00	87.68 ± 0.01	45.05 ± 0.01
GAT	82.98 ± 0.01	80.39 ± 0.00	47.12 ± 0.01
GraphSAGE	84.06 ± 0.01	83.66 ± 0.01	50.80 ± 0.00

Table 3: Metrics across various models for PubMed Dataset

Thus, we can conclude from the above observations that not all models perform equally well in all cases. In case of sparse graphs such as Cora, the models performed similarly well. However, when the graph is dense, like the PubMed dataset, the computational graph of every node also becomes more expensive to compute, thus the simple GCN architecture gains an advantage, while more compute intensive models such as GAT and GraphSAGE remains at a disadvantage. However, the embeddings of GCN with higher AUC ROC scores in second dataset were consistently performing poor in case of classifying positive links, which implies that its higher AUC-ROC scores are due to its ability to correctly classify negative edges. Thus, in case of dense graph, not only do various architectures differ significantly, we also have to realize our evaluation based on the downstream task. A fraud detection system will prefer to classify the positive edges correctly in order to find the outlier among them, while a recommendation system, will focus on classification of unseen edges.

Future Work

For the future part of this project, we intend to implement parameterized, which are based on tensor factorization such as DEDICOM and RESCAL [13], and evaluate their effects on the overall performance of the models compared to non-parameterized Inner Product decoder. We also propose a new approach to such parametrized decoders by allowing them to parameterize directed edges.

We also intend to extend our current work to relational datasets which include edge labels in the form of relation types as well as multi-modal graphs. This will allow us to apply our solution to various datasets dealing with drug-protein interactions aimed at finding side effects of similar drugs without the need for pharmaceutical companies to extensively search every possible combination.

We also intend to explain the decisions of these models using various State of the art explainable AI techniques to allow for external human verification in critical tasks. We intend to experiment by selecting partial input feature which tend to be important towards a decision made by the model to give it in a feedback loop to the model in hopes of increasing efficiency.

References

[1] [1] [2] W. L. Y. R. &. L. J. Hamilton, "Representation learning on graphs: Methods and applications," 2017.

[3] [2] [4] D. F. J. H. a. S. S. 2. J. Ben Schafer, "Collaborative filtering recommender systems," 2007.

[5] [3] [6] P. X. B. W. Y. &. Z. X. Wang, "Link prediction in social networks: the state-of-the-art," 2015.

[7] [4] [8] Z. C. M. a. K. M. Stanfield, "Drug response prediction as a link prediction problem," 2017.

[9] [5] [10] Z. B.-J. a. J. K.-S. Yanjun Qi, "Evaluation of different biological data and computational classification methods for use in protein interaction prediction," 2006.

[11] [6] [12] D. a. K. J. Liben-Nowell, "The link-prediction problem for social networks," 2007.

[13] [7] [14] A.-L. a. A. R. Barabasi, "Emergence of scaling in random networks," 1999.

[15] [8] [16] L. Katz, "A new status index derived from sociometric analysis," 1953.

[17] [9] [18] T. N. &. W. M. Kipf, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv preprint arXiv:1609.02907.,* 2016.

[19] [10] [20] P. C. G. C. A. R. A. L. P. &. B. Y. Veličković, "Graph attention networks," 2017.

[21] [11] [22] W. L. R. Y. a. J. L. Hamilton, "Inductive representation learning on large graphs," 2017.

[23] [12] [24] T. N. &. W. M. Kipf, "Variational graph auto-encoders," 2016.

[25] [13] [26] E. Papalexakis, "Tensors for data mining and data fusion, models, applications, and scalable algorithms.," 2017.

[27] [14] [28] C. Doersch, "Tutorial on Variational Autoencoder," 2016.

[29] [15] [30] P. C. Cora, "Planetoid Datasets," [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html#torch_geometric.datasets.Planetoid.