Link Prediction in Social Networks

by

Ajith Kusalakumar

An undergraduate thesis

Submitted to the School of Computer Science and Technology
in Partial Fulfillment of the Requirements for the Degree
of Bachelor of Computer Science (Honours)
at Algoma University

Ontario, Canada

2022

Link Prediction in Social Networks

by

Ajith Kusalakumar

APPROVED BY:

Simon Xu, Internal Examiner
School of Computer Science and Technology

Zamilur Rahman, Supervisor
School of Computer Science and Technology

April 8, 2022

Declaration

I hereby declare that this is a true copy of my thesis, including any final revisions, as approved by my examination committee. This thesis has not been submitted for the award of any degree or diploma to any other University or Institution.

Abstract

Social network analysis has gotten a resurgence in literature due to its wide applicability in various fields and its ability to represent complex relationships between objects. The study of link prediction in social networks has also seen much growth compared to traditional social network analysis methods due to its practical applications in recommendation systems, network completion, and network construction. Link prediction predicts the future connections between pairs of unconnected nodes based on the existing connections. A wide range of strategies is used for link prediction like similarity metrics, probabilistic methods, graph neural network-based methods, etc. In this thesis, we propose a similarity metric model and a graph neural network model. Similarity metrics are one of the most common methods used for link prediction and calculating the structure of two nodes based on their local topology. The existing methods are insufficient as most cannot consider complex relations between nodes. Few similarity metrics exist that can compute similarity scores for nodes that do not have common neighbours. Most similarity metrics treat the nodes the same regardless of their local neighbourhood structure. We addressed this issue and proposed a novel similarity metric. The proposed model is adaptive to the local neigh-

bourhood and can calculate the similarity score of nodes with no common neighbours. The proposed metric determines a neighbourhood vector that is locally adaptive for each node and uses the similarity between the neighbourhood vector to predict relationships beyond first-order neighbour relationships. Two experiments were conducted on a wide range of real-world datasets, which showed that the proposed similarity method was comparable in performance to the leading similarity metrics in general and outperformed these metrics when considering only nodes without any common neighbours.

A Graph Neural Network (GNN) is at the cutting edge in the study of neural networks and has recently been used in link prediction. In GNNs, each node learns its representation through an iterative process of updating its representation based on the neighbours' representation. The model can learn meaningful relationships between the nodes using this process. GNNs are used effectively to learn the representation of the graph and the relationship between nodes as vectors. These vectors can be used to calculate the probability of a link forming between two nodes. A model is proposed that utilizes GNN on various homogeneous graphs for link prediction. The proposed model learns the representation of the graph using message passing layers and exploits the learned relationships to make link predictions between nodes that are up to third-hop neighbours. An experiment was conducted on several real-world datasets to determine the model's accuracy. The model performed with an accuracy of at least 82% in each dataset and 86% on average.

Dedication

I would like to dedicate this research thesis to Dr. Rahman and all the professors at Algoma University for giving me many opportunities to prove myself, for helping me achieve my dreams.

Acknowledgements

Throughout this Thesis course, I receive tremendous support from Dr. Rahman. I would like to express my deepest gratitude to my supervisor Dr. Zamilur Rahman. I am thankful for their time and guidance; for meeting me twice weekly to check on my progress and pointing me towards the right direction in research, for pushing me to go further after each milestone. This would not have been possible without his continued guidance and corporation.

I would like to thank Dr. Simon Xu for his kind acceptance to be the internal examiner for my thesis defense. I appreciate your time and help.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A social network is a structure that is used to represent the relationships between individuals in society. The study of social networks has become more prevalent due to the large influences online social media platforms have on human interaction. Social networks can represent various behaviours of social entities, like criminal networks, conflict and cooperation networks. The social network analysis methods may also be generalized further to represent relationships between objects in general, like chemical reactions.

Social networks can be represented using graph data structures, where a node represents each social entity, and the edges of the graph represent the relationship between various entities. This formulation of social networks is effective as a relationship between entities can be precisely represented.

In the following sections, we collect some common terminologies used throughout the thesis in one place.

1.1 Social Networks

A *Social networks*, SN, is defined as a tuple of the form (G, f_V, f_E). where G is a graph, f_V and f_E are meta-functions [12]. The graph G is defined as a tuple $G = (V, E)$ where $V = \{v_1, v_2, ...v_n\}$ is the set of vertices and v_i for $i = 1, ..., n$ are vertices. Vertices are also called *nodes* or *actors*. The edges $E = \{(v_i, v_j) : v_i \text{ and } v_j \text{ form links}\}$, therefore, $E \subseteq V \times V$ is the set of edges, with equality only if G is a *complete graph*. Edges are also called *relations*, or *links*.

Figure 1.1: *(a) Undirected homogeneous graph, (b) Directed homogeneous graph*

The graph shown in Fig. 1.1(a) is an example of an undirected homogeneous graph. The nodes or actors in this graph are $\{v_1, v_2, v_3, v_4, v_5\}$ and the edges or links are $\{(v_1, v_2)$, (v_1, v_3), (v_1, v_5), (v_2, v_3), (v_3, v_4), $(v_4, v_5)\}$. Note that if an edge (v_i, v_j) exists in an undirected graph, then (v_j, v_i) are edges as well, even though they are not shown in the set of edges. In an undirected graph, the edges are represented without any direction. The relationship exists in both directions.

The graph shown in Fig. 1.1(b) is an example of a directed graph. The same set of

nodes is present in the graph as Fig. 1.1(a), but the edges are different. If an edge (v_i, v_j) exists then it is not necessary that (v_j, v_i) is an edge as well. The edges in a directed graph are represented with directions from the source node to the sink node.

Undirected graphs represent relationships that are symmetric and directed graphs represent asymmetric relationships. In other words, in undirected graphs the edges are unordered pairs, $(v_i, v_j) = (v_j, v_i)$. In directed graphs, the edges are ordered pairs, $(v_i, v_j) \neq (v_j, v_i)$.

The definition for edges can be expanded for weighted edges to include the weight. In weighted edge graphs, the edges between nodes v_i and v_j with weight w_{ij} can be defined as (v_i, w_{ij}, v_j) where w_{ij} is the weight between v_i and v_j and $w_{ij} \in W$, where W is the set of weights. The weights can represent the physical properties of the edges like time, distance, value, etc.

f_V is a function that maps each node to a particular type of node. f_E is a function that maps each vertex to a particular type of vertex. They are defined as follows: $f_V \colon V \longrightarrow T_V$, where T_V is the set of types of vertices. $f_E \colon V \longrightarrow T_E$, where T_E is the set of types of edges.

These functions are particularly important when the graph that is being represented is heterogeneous graph. On a homogeneous graph, the definition is trivial, and the sets T_V, T_E have a single element. For heterogeneous networks, $\|T_V\|$ or $\|T_E\| > 1$. In other words, homogeneous graphs have a single type of nodes and a single type of links, while heterogeneous graphs have multiple types of nodes or multiple types of links. For the graph

in Fig. 1.1, the meta-function are $f_V : v_i \longrightarrow \{\text{White}\}$ and $f_E : (v_i, v_j) \longrightarrow \{\text{Black}\}$.

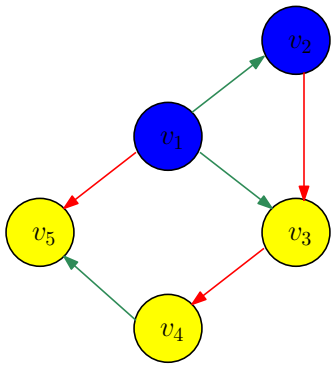


Figure 1.2: *Example of Heterogeneous undirected graph*

For the heterogeneous graph in Fig. 1.2, the function is more complicated. the meta-function are $f_V : v_i \longrightarrow \{\text{Yellow, Blue}\}$ and $f_E : (v_i, v_j) \longrightarrow \{\text{Red, Green}\}$. The meta-functions map the nodes and edges to a set of objects that give the nodes more attributes and differentiate them as different nodes, making the graph heterogeneous. Note that the graph can have multiple types of nodes or multiple types of edges to be heterogeneous; it does not necessarily require both. Heterogeneous social networks are called knowledge graphs when it encompasses all the known information about a node.

Notice that by definition $G \subseteq SN$, the social networks expand the definition of the already ubiquitous definition of a graph. Social networks can have undirected, directed, and weighted graphs as well. When discussing homogeneous graphs, the social network becomes a graph, as the meta-functions are trivial. Knowledge graphs are reduced to homogeneous social networks when the meta-function is ignored.

The scope of this thesis is limited to undirected homogeneous graphs, but the methods

used here can be easily expanded to heterogeneous and directed graphs with weight as well.

Each graph is a single snapshot of the social network at some time t. The nodes and the relationships between the nodes are changing in the graph. Therefore it is important to make predictions on how the social network changes over time. The prediction of future link formations as the network changes is one of the most important problems in social network analysis. This is the problem that is studied in this thesis.

1.2 Link Prediction

The problem of link prediction involves finding new or existing but hidden relationships between actors or nodes in a social network. Srinivas et al. defines link prediction as: $G_t = (V, E_t)$ at a given time t, we must predict the set of new links that can emerge in the time interval $[t, t^{'}]$ where $t^{'} > t$ [19]. It is assumed that the number of edges increases with time, so $E_t \subset E_{t'}$ and the number of nodes in the network remains constant through time.

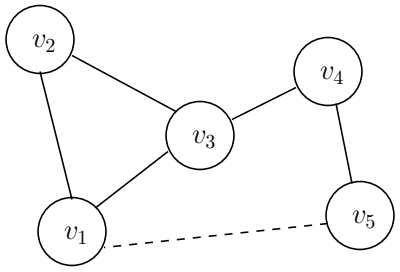


Figure 1.3: *Graph with solid lines representing links that are already known and dashed lines representing future link prediction or hidden link prediction*

The link prediction problem is depicted in Fig. 1.3. The links at time t are represented by solid lines, and the links that may be added at time $t^{'}$ are represented by dashed lines.

The link prediction task can be classified into static and dynamic link prediction. Static

link prediction involves link prediction on a single snapshot of the network, while dynamic link prediction involves working with a series of snapshots over a period of time. Even though dynamic predictions significantly improve performance, limited data is available with multiple snapshots over time. It would also involve high computation costs compared to static prediction; therefore, most applications would involve a single snapshot of the network. Even in dynamic link prediction, the number of nodes is kept constant. If the number of nodes changes over time, this is called evolutionary link prediction, which is beyond the scope of this thesis. The scope of this thesis is limited to static link predictions. The following sections discuss several link prediction methods with their advantages and disadvantages.

1.2.1 Methods for Link Prediction

There are several methods are used for link prediction. These methods can be generally categorized into similarity-based metrics, probabilistic models, and machine learning-based models [10].

Similarity-based Models

Similarity-based methods use the topological properties of each node to predict how similar the nodes are. The idea behind this method is that similar nodes tend to have closer relationships; therefore, they tend to form links. The similarity metrics can be categorized into global and local measures based on whether global properties or local properties are used in the calculations. Local properties include node properties and properties like common

neighbours, while global properties include properties such as centrality, paths between nodes, etc. [10]. Local Similarity measures are more often used as they tend to be good at making predictions and have very low computational costs as global metrics tend to have high computational costs.

Probabilistic Models

Probabilistic models are models composed of several parameters that are approximated to best estimate the model given data [10]. Probabilistic methods tend to require more information than similarity measures; therefore, metafunctions play an important role in probabilistic methods. Parameter tuning is quite important for probabilistic models that limit their applicability [10]. These methods tend to create a probabilistic model of the behaviour of the social network. It is often difficult to make a general probabilistic method.

Neural Network Models

Link prediction using machine learning techniques has become more prevalent. The problem is redefined as an unsupervised binary classification problem based on the topological features of the graph. Neural networks tend to do very well in link prediction as they can represent the low-level details of links after dimensionality reduction. Graph embedding also allows the representation of a graph in lower-dimensional matrix representation while preserving the information about the structure of the graphs. This observation led to the use of graph neural networks. These are much more efficient computationally than probabilistic and similarity-based methods. Link prediction using graph neural networks is one of

the cutting-edge topics currently being studied. Graph embedding is done through message passing layers rather than external graph embedding techniques in graph neural networks. There are still many open questions that remain about this method, such as pre-training models.

Like similarity-based methods, graph neural networks are also based on topological properties. The graph neural network creates vector representations of each node using their topology. The idea behind link prediction based on GNN is that the model learns how to represent complex relationships in its embedding. A similar embedding has a stronger relationship, so it has a higher chance of forming links.

Similarity metrics and probabilistic metrics use certain assumptions about the dataset to reduce the complexity of the calculation, while the graph neural networks-based methods learn from the graph itself. One of the most common assumptions made is that social networks, especially human social networks, social networks tend to be scale-free networks.

1.3 Scale-free Networks

Social networks are classified into various types depending on their behaviour. The most common classifications are scale-free networks and small-world networks. It is often the case that these two types are very related. Scale-free networks are an important approximation that is used in classical link prediction methods. It has been an accepted theory in Social network analysis that most large-scale networks tend to be scale-free. Broido et al. have

shown that most real-world networks follow the power-law only weakly and can be described better by other distributions [2]. Yet, it is still widely accepted that most networks are scale-free, so it would still be a good assumption that most large social networks are scale-free.

Scale-free social networks are graphs whose degree distribution follows the power law. in other words, $P_{deg}(k) = k^{-\alpha}$ where $\alpha > 1$. This implies that as the degree k increases, the probability of finding a node with degree k in the graph decreases significantly. That is, there are only a few nodes with large degrees, while most nodes have less than the average amount of degrees.

The nodes with degrees larger than the average are called hubs. These play a large role in networks and are usually the central nodes of the network. In network security applications, they play a large role as they are often targeted as they can be single points of failure for the entire network. The scare-free factor or a variation of it is often used as a weight in classical link prediction techniques.

This thesis focuses on link prediction using classical similarity-based metrics (see chapter 2 and link prediction using graph neural networks (see chapter 3).

1.4 Organization of the Thesis

This thesis proposed two approaches to solve the link prediction problem. One approach is based on similarity metrics, and the other is based on neural networks. Link prediction using similarity metrics is discussed in chapter 2 of this thesis. In chapter 2, a brief introduction to

similarity metrics, along with some commonly used similarity metrics is given. The problem with existing similarity metrics is also discussed. We proposed a similarity metric (DILA) based on the latent neighbourhood similarity based on a locally adaptive similarity metric. Two experiments were conducted on a wide range of real-world datasets, and the proposed model is compared with the current state-of-the-art models.

Link prediction using graph neural networks is discussed in chapter 3 of the thesis. A brief introduction to neural networks is given, which motivates the need for graph learning. Graph neural networks are introduced, and a literature review is conducted on some link prediction models that use neural networks. A graph neural network link prediction model is proposed and compared to other neural network link prediction models. The results of the experiment are discussed in chapter 3.

A discussion regarding the proposed similarity metrics and graph neural networks models is given in chapter 4. This chapter summarizes the advantages and disadvantages of the methods and provides future work that can be conducted continuing this thesis.

1.5 Summary

A brief introduction to social networks was given and the precise formulations of the social networks were discussed, and various types of social networks were defined. The problem definition of link prediction was given along with the methods used for link prediction.

Chapter 2

Classical Link Prediction Methods

Classical link prediction methods include link prediction methods, link probabilistic models, and similarity measure-based models, whereas modern link prediction methods use neural networks to predict relations.

Some of the classical link prediction techniques using similarity measures are outlined in this chapter. We also discussed the flaws of these methods. In this chapter, we proposed a novel similarity measure with experiments and results.

2.1 Similarity Methods

The main idea behind the use of similarity metrics in link prediction is that nodes that are similar or exist in similar topological structures tend to have a higher chance of being related to each other. The topological structure's similarities are calculated using similarity metrics. Various similarity metrics are used. A few common similarity metrics are introduced here.

Let $S(m, n)$ be the similarity score between node m and node n.

2.1.1 Preferential Attachment

The preferential attachment metric is given by the product of the degree of the two nodes.

$$S(m, n) = d_m \cdot d_n$$

The idea behind this measure is that nodes with higher degrees tend to connect.

2.1.2 Common Neighbour Similarity

The common neighbour similarity measure is the number of common neighbours between node m and node n

$$S(m, n) = |\Gamma(m) \cap \Gamma(n)|, \text{ where } \Gamma(m) \text{ is the neighbours of node } m$$

This measure approximates the neighbourhood around the nodes. The idea is that nodes with more common neighbours tend to be similar, and therefore, should form links.

2.1.3 Jaccard Coefficient

The Jaccard coefficient is very similar to the common neighbour metric. It is just a simple normalisation of the common neighbour similarity metric.

$$S(m, n) = \frac{|\Gamma(m) \cap \Gamma(n)|}{|\Gamma(m) \cup \Gamma(n)|}$$

The Jaccard coefficient gives the probability of finding common neighbours among all possible neighbours [10].

2.1.4 Adamic Adar Index

The Adamic Adar index assigns a larger value to the neighbours that have a lower degree.

$$S(m,n) = \sum_{z \in \Gamma(m) \cap \Gamma(n)} \frac{1}{\log(d_z)}$$

2.1.5 Resource Allocation index

The resource allocation index is described as

$$S(m,n) = \sum_{z \in \Gamma(m) \cap \Gamma(n)} \frac{1}{d_z}$$

This measure is very similar to the Adamic Adar index, except that only the degree is used rather than the logarithm of the degree. This metric penalizes nodes with a higher degree than the Adamic Adar index. Both Adamic Adar and the Resource Allocation Index follow the power law. They assign a larger similarity score to neighbours with a lower degree, which is akin to the power law, which states that the number of nodes with higher degrees tends to be small.

Notice that there are only small differences between the various metrics, but that can result in large differences in the measurement as the small changes to the similarity metric change what is given importance. These metrics also have many problems that must be addressed.

2.2 Problems with Existing Metrics

Metrics are not good at predicting relationships between nodes that are not neighbours.

Interestingly, out of all the similarity metrics mentioned, only the preferential attachment measure can even calculate the probability of forming links between nodes that are not neighbours as the other metrics all rely on the common neighbour method in some way. Yet the preferential attachment metric is not very good at predicting relationships between nodes that do not have common neighbours as it wasn't created for that purpose. The topology of the common neighbours is the most commonly used metric as it tends to give the best results, but these results are still lacklustre for finding relationships between nodes that do not have common neighbours. The common neighbour method must be generalized to give better results for nodes that do not have common neighbours. There is a need for metrics that can predict relationships between nodes that are not directly related.

The similarity score assigns the same weight regardless of the local neighbourhood [19]. According to the previous methods, two nodes with the same degree or the same number of common neighbours have the same similarity score. This doesn't take into account their importance in the structure.

The similarity measure is not adaptable to the network or the neighbourhood [19]. The similarity scores have different accuracy depending on the neighbourhood and on whether the data set is dense or sparse [19]. It is noted that the different metrics work better on different types of networks. The similarity measures depend on the size and density of the neighbourhood. It can be crucial for large neighbourhoods to give less weight to degrees compared to smaller neighbourhoods. So a metric that differs based on the neighbourhood

gives better results.

Several papers attempt to address these issues individually, but these issues have not been addressed by a singular method. Zareie et al. introduce a similarity metric based on latent relationships between nodes called Direct-Indirect Common Neighbours (DICN) [24]. Latent relationships generalize the idea of neighbours to include the neighbours of neighbours or second-order neighbours.

A question may arise noting if we can generalize this further to consider the neighbours of the neighbours of neighbours, and so on. Nearly all scale-free networks also tend to be small-world networks as well. Small-world networks are defined as networks whose average distance L between two nodes follows the rule $\log(L) \propto \log(N)$, where L is the shortest number of hops required to get from one node to another, and N is the number of nodes [13]. A notable feature of such a graph is that each node tends to have a maximum of 6 degrees of separation [13]. So on average, two degrees of separation are sufficient, therefore it is acceptable to consider only the latent neighbour and not generalize the neighbourhood vector any further.

In Zareie et al.'s model, each node has a corresponding vector, which captures the structure of the neighbourhood of the vector, called the neighbourhood vector, denoted N_i.

N_i is defined as :

$$N_i[z]_{z=1,2,\ldots,|V|} = \left\{ \begin{array}{ll} d_i, & \text{for } z = i \\ CN_{iz}, & \text{for } v_z \in \Gamma_i^2 \\ CN_{iz} + 1, & \text{for } v_z \in \Gamma_i \\ 0, & \text{otherwise} \end{array} \right\}$$

Where Γ_i is the set of neighbours of node i and Γ_i^2 is the second-order neighbour, in other words, the set of nodes that are neighbours of the neighbour nodes of node i [24].

The similarity score, $DICN_{i,j}$, is given by $(1 + CN_{i.j})(1 + corr_{i,j})$, where $corr_{i,j}$ is the Pearson correlation between the two neighbourhood vectors N_i and N_j [24]. The use of the Pearson correlation is a generalization from using similarity between nodes to using the similarity between neighbourhoods. The Pearson correlation $corr_{i,j}$ is calculated as

$$corr_{i,j} = \frac{\sum_{z \in UN_{ij}} (N_i[z] - \overline{N_i})(N_j[z] - \overline{N_j})}{\sum_{z \in UN_{ij}} \sqrt{(N_i[z] - \overline{N_i})^2} \sum_{z \in UN_{ij}} \sqrt{(N_j[z] - \overline{N_j})^2}}$$

where $\overline{N_i}$ is the mean value in the union neighbourhood [24].

The DICN similarity score allows the calculation of similarity scores between nodes that are not direct neighbours, but this still suffers from the problem of treating all the nodes equally regardless of their importance or neighbourhood. Hyoungjun et al. proposed the idea of a community adaptive similarity measure [9]. The community network is divided into multiple communities using a modularity density function [9]. Two different scores are calculated when calculating the similarity score between the nodes, depending on whether the two nodes exist in the same community.

Hyoungjun et al. propose two similarity measures.

$$CACN(i,j) = \left\{ \begin{array}{ll} \lambda CN(i,j), & \text{if } link(i,j) \in S_{intra} \\ (1-\lambda)CN(i,j), & \text{if } link(i,j) \in S_{inter} \end{array} \right\}$$

$$CARN(i,j) = \left\{ \begin{array}{ll} \lambda RA(i,j), & \text{if } link(i,j) \in S_{intra} \\ (1-\lambda)RA(i,j), & \text{if } link(i,j) \in S_{inter} \end{array} \right\}$$

where S_{intra} is the set of intra-links, that is, the links within the community, and S_{inter} are links that exist between the two communities [9]. λ is a value between 0 and 1.

The value of λ is selected so that $CACN(i,j)$ and $CARN(i,j)$ similarity scores are maximized for the links in the same communities that already exist. The $CACN(i,j)$ and $CARN(i,j)$ measures introduce a rudimentary adaptive similarity measure that depends on whether the two nodes are in the same neighbourhood. This does not take into account the neighbourhood similarity in any way. These measures are still limited by the fact that they are unable to calculate the similarity score between nodes that are not direct neighbours because the resource allocation index and common neighbour methods require common neighbours to be calculated.

Srinivas et al. propose a locally adaptive measure defined as

$$LA(a,b) = \sum_{z \in \Gamma(a) \cap \Gamma(b)} \frac{1}{(d_z)^{c\frac{\alpha_l}{\alpha_g}}}$$

where d_z is the degree of node z, $\Gamma(a)$ is the set of neighbours of node a, and α_g and α_l are the global and local power-law coefficients. c is the smoothing parameter with $0 \leq c \leq \frac{\alpha_g}{\alpha_l}$ [19]. α_g is the global distribution and is fixed, while of α_l varies locally as it is dependent on the neighbourhood [19].The maximum likelihood estimate of a_l is as follows:

$$\alpha_l = 1 + \frac{k}{\sum_{i=1}^{k} \ln \frac{x_i}{x_m in}}$$

where x_i is the degree of node i and k is the size of the neighbourhood [19]. The values of

$\frac{\alpha_l}{\alpha_g}$ can be approximated by $\frac{\alpha_l}{\alpha_g} \propto \frac{avg_g}{avg_l}$, where avg_l and avg_g are the average degree of local and global neighbourhoods, respectively [19]. Since $avg(g)$ is the global average it is fixed, while $avg(l)$ is calculated for each node based on its local neighbourhood.

The locally adaptive metric transitions from common neighbour ($c = 0$) to Resource allocation index ($c = \frac{\alpha_g}{\alpha_l}$) depending on the value of c. The metric fluctuates between these two metrics depending on the value of c [19]. Yet a further study of the c constant to be tuning parameter is left as future work. The adaptive nature of this similarity measure takes into account the surrounding neighbourhood of the node. Using this similarity measure to calculate the similarity score between nodes that are not direct neighbours is still not possible as it relies on common neighbours for its basic calculations.

2.3 Proposed Metric

The elements from each of the papers mentioned above have been incorporated into the proposed metric. We called it the Direct-Indirect Locally Adaptive (DILA) model. The neighbourhood vector (N_i) for each node i is calculated by

$$N_i[z]_{z=1,2,\ldots,|V|} = \left\{ \begin{array}{ll} d_i, & \text{for } z = i \\ LA_{iz}, & \text{for } v_z \in \Gamma_i^2 \\ LA_{iz} + 1, & \text{for } v_z \in \Gamma_i \\ 0, & \text{otherwise} \end{array} \right\}$$

where

$$LA_{ab} = \sum_{z \in \Gamma(a) \cap \Gamma(b)} \frac{1}{(d_z)^{\frac{\log(avg_g)}{\log(avg_a)}}}$$

Here, avg_g and avg_a are the global average degree of the nodes and the average degree of

the nodes in their common neighbours. This approximation can be used to approximate the power-law distribution constant [10]. This implicitly assumes that the network is a scale-free network, but since most networks can be approximated as scale-free, this assumption is valid and useful.

Notice that compared to the DICN, the main difference is that the common neighbour similarity measure is replaced by a locally adaptive measure from Kumar et al [10]. The common neighbour treats all nodes equally, but this technique is locally adaptive to the neighbourhood and considers the importance of each node's neighbourhood similarity, which produces a greater general similarity.

The Pearson correlation $corr_{ij}$ of the two neighbourhood vectors of node i and node j is calculated. This is used to calculate the similarity score by using the following formula

$$DILA(i,j) = (1 + LA_{i,j})(1 + corr_{i,j})$$

This allows link prediction between nodes that are not common neighbours, as DICN had, but this method also compares if each node in the neighbourhood reflects the global degree properties accounting for local variance in the neighbourhood.

2.3.1 Methodology

The proposed similarity method was implemented using Python [20] and the Networkx library [7] for Python. It was executed on a PC with an i5 1.6 GHz processor.

The accuracy of the scores is calculated using Area Under the receiver operating charac-

teristic curve (AUC). This metric is the same one used in [24] and is used to easily compare the proposed similarity metric with other existing similarity metrics.

The AUC is calculated by selecting the β percentage of the existing links as training edges E^{train}, and $1-\beta$ percentage of existing edges as testing edges, E^{test}. A graph, G_{train} is created using the E^{train} edges while adding back all the existing nodes, even if the nodes do not have any connecting links.

The DILA score is calculated for each node that shares links in E^{test}. Each of those DILA scores is compared with the DILA score of two nodes with links in E^{C}, where E^{C} is the set of all the links that do not already exist. β is selected to be 0.8 for this experiment. As 0.8 was selected since Zareie et al. has selected 0.8, the same number needs to be selected for the AUC to be compared [24].

If the metric is working well, the similarity score of nodes that already have links must have a higher score than the nodes with links that do not already exist.

The AUC metric is calculated using

$$AUC = \frac{n_1 + \frac{1}{2}n_2}{n}$$

where n_1 is the number of edges that already exist with a similarity score higher than the similarity score of edges that do not yet exist, and n_2 is the number of edges that already exist with a similarity score equal to the similarity score of edges that not yet exist. n is the total number of edges considered.

This metric works based on the assumption that nodes with higher similarity scores tend to form relations. So links that already exist must have a higher score compared to links that do not already exist. This also shows the predictability of the similarity measure if it consistently gives a higher score for links that already exist and a lower score for links that does not exist; then it should give a higher score for links that can be added in the future as well, making it a good model.

Two experiments were conducted using the metric mentioned above. In the first experiment, all nodes with connecting edges in T^{test} are compared with all nodes with randomly selected connecting edges in E^{C}. The average AUC score is calculated and compared with other similarity metrics for each dataset.

In the second experiment, a similar setup was used. From each of the edges in E^{test} and E^{C}, all edges without any common neighbours and a degree of at least 1 in the graph G_{train} are determined, γ edges are randomly selected, the AUC score is calculated and, compared with the similarity metrics for each dataset.

2.3.2 Experimental Results

Two experiments were conducted on the following datasets Zachary Karate club (KRT) [23], Dolphins (DLN) [11], US Airline (UAL) [17], Email (EML), and Yeast (YST) datasets [3]. These datasets all fall into different categories of social networks. KRT and HAM are social networks, while DLN is an animal network. YST is a biological network, while UAL and EML are traffic and communication networks. The details of the characteristics of

these datasets are shown in Table 2.1. We compare the performance of the proposed DILA method to the methods in Zareie et al. [24].

The following similarity metrics are chosen: Common Neighbours (CN), Preferential Attachment Index (PA), Jaccard Index (JC), and Direct Indirect Common Neighbours (DICN) for comparison.

The AUC tends to have a large variance, and the AUC value can range from 65 to 92 in some cases, so to get the most accurate answer, the AUC score is calculated 15 times, and the average AUC is considered. Even when averaging, the AUC score still has a large variance, but to a lesser degree. A larger average wasn't taken as it is infeasible to do, as it would take a much longer time. Zareie et al. have chosen to take the average of 15 calculations, so to make it comparable, the AUC score is averaged over 15 calculations.

Network	$\|V\|$	$\|E\|$	$\|E^c\|$	γ
KRT	34	78	527	527
DLN	62	159	1,732	1732
UAL	332	2,126	52,820	15
EML	1,133	5,451	635,827	15
YST	2,284	6,646	2,600,540	15

Table 2.1: *Table of characteristics of datasets and the value chosen number of non-existing links*

Table 2.1 shows the characteristics of the dataset and shows the number of non-existing links that were chosen for comparison. It wasn't feasible to calculate the AUC for some of the larger data sets. Since AUC scores are calculated against all of the links that do not exist in the graph, this can be a large number. A single calculation for larger data sets

tends to take more than 24 hours and a few days for a few of the data sets if one were to compute the AUC score for all of the non-existing links. For example, if the AUC score was computed on the entire non-existent links on the US Airlines dataset, it would take 18 days. This wasn't even the largest dataset.

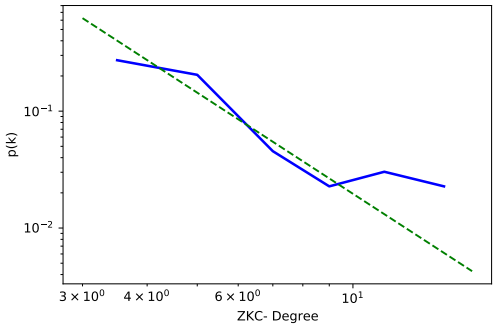
Therefore, unlike the Zareie et al., $\gamma = min(|E^{\text{test}}|, 15)$ non-existing links from E^C were selected to calculate the AUC score for larger datasets, and $\gamma = \frac{(|V|)(|V|-1)}{2} - |E|$ was used for the smaller datasets, This does not affect the AUC score because the AUC score is normalized to the number of non-existent edges. Therefore, the AUC score should still be comparable.

In experiment 2, instead of choosing all of the links from E^{test} and γ links from E^C, 100 from each set were randomly selected; this calculation has proven to take much longer than the calculations performed in experiment 1.

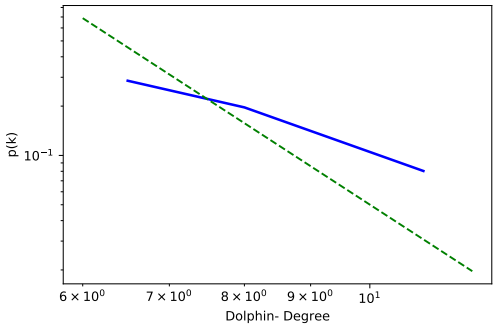
Network	CN [24]	PA [24]	JC [24]	DICN [24]	DILA
KRT	0.6884	0.6976	0.6884	0.7654	**0.7917**
DLN	0.7296	0.6072	0.6303	**0.7943**	0.7731
UAL	0.9289	0.8852	0.7802	**0.9367**	<u>0.9335</u>
EML	0.8186	0.7767	0.8158	**0.8932**	<u>0.8852</u>
YST	0.7786	0.8090	0.7592	**0.8839**	0.8692

Table 2.2: *Table of AUC values on different methods on between randomly selected nodes. The values for the other similarity metrics are taken from Zarie et al. Best results are shown in bold, and results comparable to best are underlined*

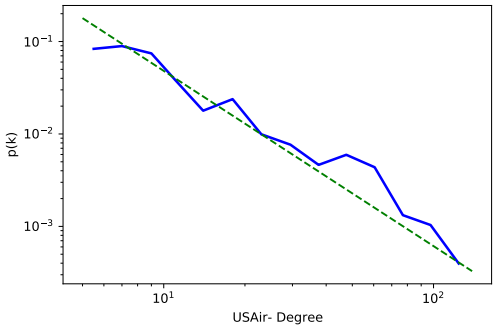
The results of experiment 1 are shown in Table 2.2. Notably, the DILA has done much better than all the classical metrics like CN, PA, and JC. DICN model performs better than the DILA model in most datasets, but DILA beats all other methods in the KRT dataset.



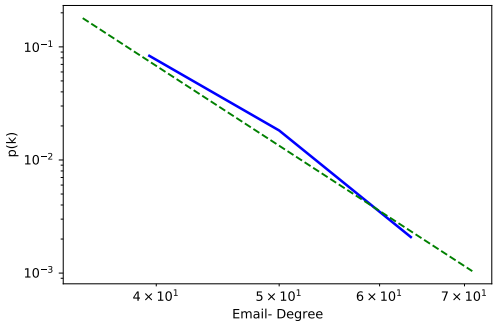
(a) *Degree distribution for KRT*



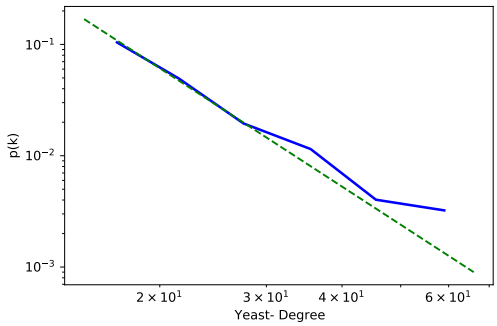
(b) *Degree distribution for DLN*



(c) *Degree distribution for UAL*



(d) *Degree distribution for EML*



(e) *Degree distribution for YST*

Figure 2.1: *The degree distributions of each of the datasets along with the power law line. The closer the degrees follow the power law, the better the model performs on these datasets*

We know that using the latent neighbourhood relationship that looks at neighbourhood similarity performs much better than looking at similarity between just nodes. The classical methods tend to overlook these valuable similarities. Although the DICN method performed

better in most of the datasets, the DILA method is performed comparably; It was less than 1% less accurate than DICN in the UAL and EML datasets. The model, on average, performed at a rate of 85.05% across all datasets in experiment 1.

The true potential of this model is seen in experiment 2, in predicting links between nodes that do not have any other common neighbours. Note that the other traditional models do not have a way of calculating the similarity score for nodes that do not have common neighbours. These metrics were discarded. The results of experiment 2 are shown in Table 2.3. Notably, the DILA model performs better than the traditional state-of-the-art models in most datasets except the EML and YST datasets, but in both cases, it does comparably well as the best-performing methods. The DILA model performs significantly better than the other leading metrics for other datasets. The model, on average, performed at a rate of 82.66% across all datasets in experiment 2.

Network	PA [24]	DICN [24]	DILA
KRT	0.7519	0.8319	**0.8510**
DLN	0.4954	0.7028	**0.8726**
UAL	0.6833	0.7979	**0.8907**
EML	0.6637	**0.7595**	0.7504
YST	**0.7755**	0.7609	0.7683

Table 2.3: *Table of AUC values on different methods between randomly selected nodes with no common neighbours. The values for the other similarity metrics are taken from Zarie et al. Best results are shown in bold and comparable results are underlined*

The variable performance of the metric on different datasets and during different measurements can be explained by the nature of the dataset and the model. The DILA model implicitly assumes that the graph dataset is a scale-free network. Figure 2.1 shows the

power-law distribution of the graph. When looking at the graph, it is clear that not all the graphs are perfectly following power-law distribution. The graph model performs better in experiment 1 in the graphs that fit the scale-free model. In experiment 2, the results are more varied. This is because only 100 links were randomly selected from E^c and E^{test}. The links that follow the power-law locally perform better; if not, their performance is worse. Since only 100 links were selected, this can have large variability for experiment 2. This could have been more clear if the experiment could have been done on a larger set, but since these links were randomly taken and averaged 15 times, the answer is still around the same ballpark.

2.4 Summary

Some of the classical and state-of-the-art similarity metric-based link prediction methods were given and were used to motivate the problem with similarity metrics. The problems discussed were that the models are terrible at predicting links between nodes that do not have common neighbours. The similarity metrics are not locally adaptive and assign the same weight based on the number of neighbours rather than the importance of the nodes. A brief literature review was conducted on models that attempt to fix these problems, and the problems with those models were discussed. Then a similarity metric that addresses these problems was proposed and compared with the existing models.

Chapter 3

Link Prediction Using Graph Neural Networks

Modern link prediction methods are most often characterized by using machine learning techniques to determine the characteristics of a social network. Social network analysis is used extensively in various applications and disciplines, including network propagation modelling, network modelling and sampling, user attribute and behaviour analysis, community-maintained resource support, location-based interaction analysis, social sharing and filtering, recommendation systems development, and link prediction. Graph Neural Networks (GNNs) are a class of deep learning methods that can be directly applied to graphs and provide an easy way to do node-level, edge-level, and graph-level prediction tasks. GNNs have shown amazing potential in link prediction tasks. Graph neural networks tend to be much faster than classical methods. Since machine learning models tend to be linear equations, they have a much smaller time complexity. GNNs tend to reduce the steps required for traditional artificial neural networks through message passing and remove the requirement

for graph embedding.

This chapter briefly introduces neural networks, and graph neural networks, and proposes graph neural networks for link prediction.

3.1 Artificial Neural Networks

Before an introduction to Graph neural networks, it is worth understanding neural networks in general. It gives us an understanding of the underlying characteristics of the graph neural network while motivating the need for graph neural networks.

Artificial neural networks are motivated by the networks of biological neural networks in the brain. It was largely inspired by the way that the biological neural network can learn patterns to make decisions.

Artificial neurons are at the core of artificial neural networks, which take one or more binary inputs and return a single binary output. These neurons can be connected in several different ways to make any logical computation. These neurons are put together in a densely connected layer with weights that can be changed to make arbitrary logical decisions.

3.2 The Perceptron

The idea of artificial neurons was later generalized into the threshold logic unit (TLU). The inputs of the TLU units are real numbers, whereas the simple artificial neuron only took binary values[5]. The output of the TLU are binary values. Instead of performing logical operations the TLU performed a sum of its inputs and a step function. In other words the

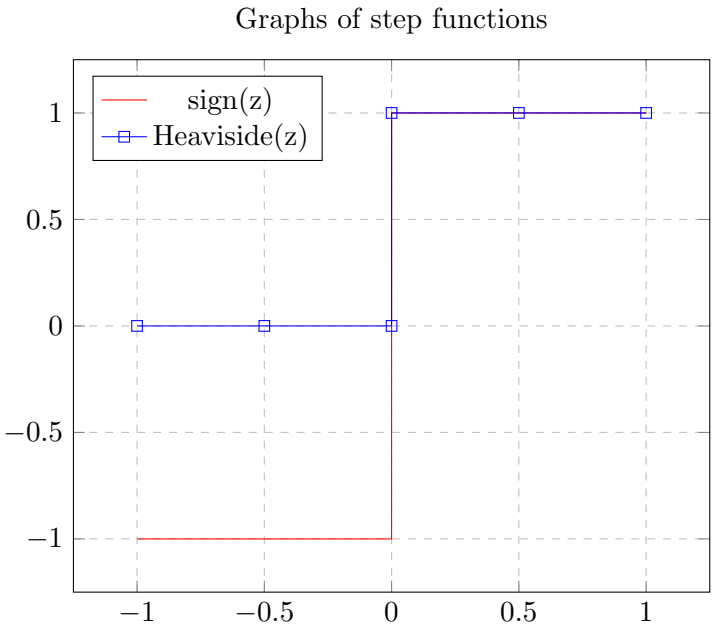


Figure 3.1: This is a graph of the Sign and Heaviside function, plotted in the range of (-1,1)

output of the TLU unit is, $h_w(x) = step(z)$, where $z = \mathbf{x}^\intercal \mathbf{w} + \mathbf{b} = \sum_{i=1}^{n} w_i x_i + 1$ [5]. Where x_is are the real number inputs, and w_is are the weights associated with the input. The b value is a bias constant is always chosen to be 1.

The most common step function used in Perceptrons is the Heaviside function and the sign function. These functions are defined as

$$\text{Heaviside(z)} = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sign(x)} = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ 1 & \text{if } z > 0 \end{cases}$$

These functions here assume a threshold value of 0, but in practice, a higher threshold value must be used. The threshold value is used to classify the value into either existing in that category or not. 1 is usually assumed to represent belonging to the category. The step and Heavyside function are shown in the Fig. 3.1.

Since the output is binary values, it can be used to classify between two classes. More TLUs can be combined to classify between multiple classes. When there is a single fully connected layer of TLUs, that is, a layer where all inputs are connected, it is called a Perceptron [5]. A Perceptron can classify instances simultaneously into multiple different classes, which makes it a multi-output classifier.

The Perceptron is the smallest possible neural network layer, as it is a single layer of fully connected TLUs. The equation for each TLU in the Perceptron layer can be computed using simple matrix multiplication using the equation

$$h_{W,b}(\mathbf{X}) = \phi(\mathbf{XW} + b)$$

where $\mathbf{X}$ is the $n \times m$ input feature matrix, with n instances and m features [5], $\mathbf{W}$ is the $n \times k$ weight matrix for each of the connections, where n is the number of input instances and k is the number of TLUs per layer, and $\mathbf{b}$ is the bias vector of size k each holding a bias term, usually selected to be 1 [5]. The ϕ function is called the activation function. The layer can perform different logical manipulations on the input by altering the weight.

The weight matrix is the most important aspect of the equation. This weight matrix determines how much each TLU should affect the output. The higher the weight, the more significant impact the TLU unit has on the output of the layer. By altering the weight, the layer can perform different logical manipulations on the input.

These weights matrices are initialized with randomly chosen values, and through mul-

tiple epochs of training, the weights are altered to produce a trained model, which can later be used for predictions. The rule that governs the training of these weights is called Hebbian learning [5]. This model changes the weight based on the output and expected result. If the output is the same as the expected result the weight is increased, if it is not, then the weight is decreased. The function that governs this behaviour is

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(y_j - \overline{y}_j)x_i$$

[5]. This equation corrects the weight of the connection between neurons i and j by some learning constant η depending on how close the predicted value y_j is to the expected value $\overline{y}_j$ for the input x_i [5].

It is often the case that a perceptron, that is, a single layer of TLUs, is not enough; multiple layers of perceptrons are required to model complex relationships. Such a model is called a multi-layer perceptron. A multi-layer perceptron is an example of a deep neural network.

3.3 Deep Neural Networks

A single layer of perceptron can approximate many linear relationships between the input and target data given enough training, but multiple layers are required to approximate more complex relationships.

Deep neural networks (DNN) are artificial neural networks with a single pass through input layers and one or more hidden layers of TLUs followed by a single output layer of

TLUs [5]. Each layer but the output layer also has a special neuron called a bias neuron that adds bias to the inputs of each layer. Each layer is fully connected; that is, each input connects to every TLU in the next layer.

3.3.1 Training a DNN

A single Perceptron layer is easy to train and develop as each weight connects the input, and the output can be adjusted according to the output and label using the Hebbian equation above based on what the model predicts and the expected value. The problem with training multiple layers of Perceptrons or DNN are that there are multiple layers, and each TLU is far removed from the output that it is difficult to say which neurons affect the output. This brings two problems; it is unclear which layers increase the error rate and need to be adjusted and which TLU's weight must be adjusted per layer.

It turns out that a method involving forward pass and backpropagation is required to train a DNN. During the forward pass, the input is taken by the input neurons and passed onto each layer, computing the result in between layers until the final output layer. The results of each intermediate layer are saved [5]. The error in the output is measured using a loss function to compare the actual output and the expected output [5]. Each output's contribution to the error is measured.

In the backpropagation step, the gradient is calculated in each layer to determine how much each layer contributes to the error until the input layer is reached [5]. Then a gradient descent step is performed to tweak the weight using the error gradient that was calculated

earlier.

3.3.2 Activation Functions

You might be asking about the difference between the step function of a TLU and an activation function of a Perceptron. The step functions are usually flat, while activation functions are curves; other than that, they are used for the same reason. The curve activation functions are chosen for twofold reasons. Not being flat, the activation function allows the gradient to be calculated, allowing the backward propagation training method possible.

Furthermore, the activation function gives the output of the layers' nonlinearity. This allows the layers to represent complex relationships. If the activation function was linear, then all the layers can only represent linear relationships, which can be done with a single layer [5].

The common activation functions used in literature are the Sigmoid function, Hyperbolic tangent function, and Rectified linear unit function (ReLU). A graph of these functions can be seen in Fig. 3.2.

It is noted that the activation function should be differentiable everywhere. However, ReLU is not differentiable at $x = 0$ but still works very well and is used as the default activation function [5].

3.3.3 Classification Models

Classification models predict the probability that an instance falls into one of the given categories. Regressions models predict a single value or multiple values in a continuous

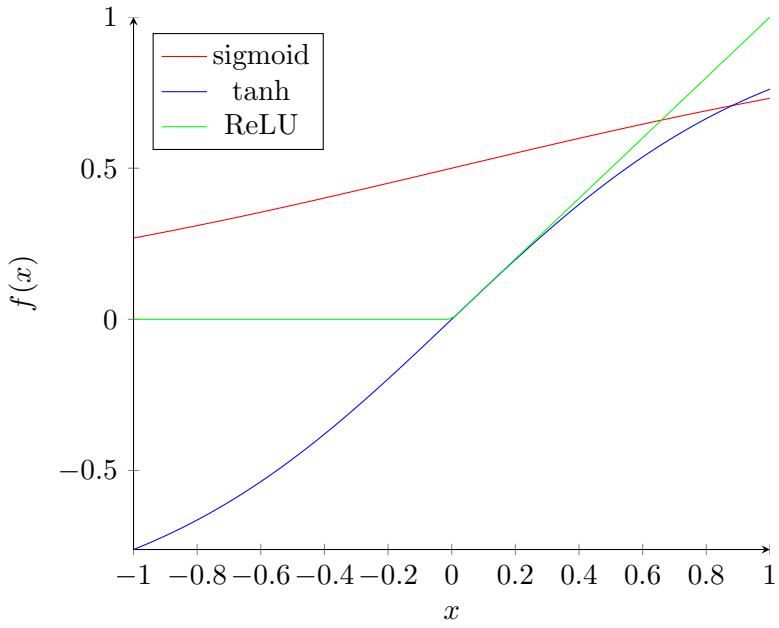


Figure 3.2: *Graphs of common activation functions*

domain rather than predicting where an instance falls into one of the given categories. Only one output neuron per category is required for classification tasks. [5]. In the case of link prediction, there is a single category, whether there is a link or an absence of links, so only a single output neuron is required. This neuron returns the probability of the link forming between the two given nodes.

A logistic activation function is usually used for the cases of binary classification [5]. Soft-max is used when there are categories that are mutually exclusive while the logistic function is used when the categories are not exclusive.

3.3.4 Batch Normalization Layers

The Batch Normalization layers are used to normalize each input of a hidden layer and shift the mean to 0 by estimating the standard deviation and mean using the batch of inputs [5].

So having a tiny batch might make the computation unreliable. This layer allows the model to learn the optimal scale and meaning of each layer's input. It can also prevent overfitting.

3.3.5 Dropout Layers

Dropout layers are often used for the regularization of the data. Regularization means preventing overfitting of the data, making the model flexible enough to apply to unseen data. The dropout layer gives every input neuron and TLU neuron a probability p of being temporarily ignored in the calculation of the current step during training [5]. The value p is typically sent around 50% for convolutions networks. The dropout layer is ignored in the calculations to make predictions after training. The purpose of this layer is to prevent overfitting through regularization.

If we have many layers, it becomes important to use dropout layers to avoid overfitting the pattern to the training set. The model's degree of freedom increases as the number of layers increases, allowing it to easily learn the noise as data. The dropout layer forces each neuron to learn a general behaviour of the datasets by forcing each neuron to spread information across each layer [5].

3.4 Graph Representations

When one thinks of graphs, they consider a visual representation of a graph, usually with circles as nodes and lines as edges. This view of graphs has very little analytical use, as it is difficult to put these images into formulas or a neural network. The traditional methods

took a topological approach to represent graphs, where each node is represented by its topology or its characteristics in the graph. The similarity metrics are used to calculate these topologies; the problem with this methodology is some metrics work better for some graphs than others. For example, the DILA metric works better for representing scale-free social networks as the model implicitly assumes that the social network is scale-free. Therefore, if the model is not scale-free, it does not produce a good representation of the data. We require a method that can be used for any general dataset and can adapt to the dataset to represent it, which the similarity metrics do not provide.

A general way that graphs are represented is through the use of an adjacency matrix. The graph in Fig. 1.2 can represented as

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

This matrix representation assumes that there is a certain order. In this case, the order is 1,2,3,4,5. There are $n!$ such permutations. One may assume that we can input a flattened version of the adjacency matrix into the neural network, but the adjacency matrix is dependent on their permutations. It is difficult for any neural network to identify them as the same graph structure. Therefore, we require any representation of the graph to be permutation invariant to be used in a neural network [8]. Permutation invariant means that

it is not dependent on the ordering.

3.4.1 Node Embedding

Node embedding is a method of representing the graph that addresses the flaws discussed in the previous paragraph. The node embeddings are permutation invariant and can adapt to the dataset and learn based on the graph. Node embeddings tend to represent a node in a graph as a vector, based on the node's topology in the graph and its local neighbourhood.

Molokwu et al. define node embeddings, or vector-space embeddings, X of nodes as a projection based on a function f such that each of the nodes of the graph is mapped to a q-dimensional vector space $\mathbb{R}^{\shortparallel}$ such that the mapping f preserves the existing links between any given pair of nodes (u_i, u_j) due to the homomorphism between the set of nodes V, and X [12].

$$f : V \rightarrow \mathbb{R}^{\shortparallel}$$

In other words

$$f : (u, v) \rightarrow \mathbb{R}^{\shortparallel}$$

The function f is called the objective function. There are several types of objective functions. It is usually a vector distance function like the cosine distance function or random walk-based mappings. The embedding is optimized so that the distance in $\mathbb{R}^{\shortparallel}$ is reflective of the position and relationship in the graph [12]. The lower the distance in $\mathbb{R}^{\shortparallel}$, the similar the two nodes are in the graph. That is, if the links already exist then the distance is minimized

and if the links do not exist, then the distance is a maximum in the embedding space [8]. These functions have the additional requirement that they are permutation invariant.

Node2vec embedding is a common method of node embedding to represent the nodes of the graphs [6]. There exist mappings between$\mathbb{R}^{|}$ and the graph. These functions are called decoding functions. It is not necessary for the discussion of neural networks, but in some cases, it is required to convert back from a node embedding to the graph. It is good to choose the homomorphism f such that it is both injective and surjective, so an inverse exists between the two domains.

The advantage these methods have is that they are not dependent on the permutation of the graph. The graph can be labelled in any way. The embedding method is generalized and can be used in any dataset, whereas some of the traditional methods like DILA can only be used in certain types of social networks.

There are also a few drawbacks of node embeddings methods. The embedding does not use all of the node features in the embedding. A method that considers the node features generally has much better results. The node embedding methods are not suitable for link prediction tasks as inherently the links change over time, and new embeddings must be produced each time there is a new link [8]. The model must be retrained along with the embedding each time there is a new node. These drawbacks can be solved by using Graph neural networks [8].

3.5 Graph Neural Networks

Graph neural networks (GNNs) differ from regular neural networks as they do not require an external node embedding step. Graph neural networks create representations of nodes that include the links between nodes as well as the topology of their neighbourhood internally. These models learn to represent the graph through unsupervised learning on the graph, therefore do not depend on the type of dataset, and they are permutation invariant.

3.5.1 Message Passing

Graph Neural Networks create a vector representation for a node through a process called message passing or recursive neighbourhood diffusion. The message passing method works as each node collects information about its direct neighbours, aggregates the information as a message and passes the message to the target node [8]. This process is done iteratively so that each neighbouring node becomes the target node and gathers information from its neighbours to pass onto the original node. Multiple message passing layers allow the graph neural network to learn about latent neighbours [8].

The GNN embedding for each node u is in the $k+1$ iteration is given by

$$\mathbf{h}_u^{k+1} = \text{update}^k(\mathbf{h}^k, \mathbf{m}_{N(u)}^k)$$

where $\mathbf{m}_{N(u)}^k$ is the message gathered from the neighbourhood $N(u)$ after k iterations using some aggregate function [8]. update^k is the k^{th} iteration of the update function used to gather and update the $\mathbf{h}^k$ embedding [8]. The initial $\mathbf{h}_u$ when $k = 0$ is the input features of

the node. The output of each layer of GNN is an embedding of the node with the k^{th} hop neighbourhood aggregation.

The formula above is given in general terms. There are several different implementations of message passing networks, each with its own use and purpose, but they all more or less follow the same formula. The most basic being

$$\mathbf{h}_u^k = \sigma\left(\mathbf{W}_{\text{self}}^k \mathbf{h}_{\mathbf{u}}^{k-1} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in N(u)} \mathbf{h}_v^{(k-1)} + b^{(k)}\right)$$

where $\mathbf{W}_{\text{self}}^k$ and $\mathbf{W}_{\text{neigh}}^{(k)}$ are trainable parameters, and σ is some nonlinear activation function discussed earlier [8]. For this model, the aggregate or the message, $\mathbf{m}_{N(u)}^k$, is the simple sum of embeddings of the neighbours with a bias term $b^{(k)}$, and the update^k function is the simple addition function [8]. Most state-of-the-art methods are made through simple tweaks to this model so that they focus on certain aspects of the graph. The various graph neural networks differ based on the aggregate and activation function.

Molokwu et al. propose a model called Representation learning via Knowledge Graph embeddings and convolution operations (RLVECO) that is based on both graph embedding and dimensional reduction using convolutional layers [12].

Their model uses five neural network layers for link prediction. First, a graph embedding layer is used. The graph embedding layer is a representation learning layer that maps each node to a vector in $\mathbb{R}^q$ [12]. Their embedding layer uses the logarithm of cosine distance to quantify the proximity of the target and source vertex [12]. A second one-dimensional

convolutional operation feature learning layer is used to reduce dimensionality. A ReLu layer maps each value x to $max(0, x)$ as the third layer and a max pool layer. The purpose of this is to introduce non-linearity as the model works better when the model is non-linear. The last is a multilayer perceptron layer, which takes the inputs from the previous layer and does a binary classification on the data [12].

Graph attention networks (GAT) are among the more common types of GNN. The main difference between other popular GNN architectures and GAT is that other GNN architectures weigh all the neighbours equally, while GAT models give importance based on a weighted average of the nodes in neighbourhood N_i during aggregation [1]. A scoring function $e(h_i, h_j)$ is used to measure the importance of each embedding h_j for node j for all nodes in the neighbourhood of i [1]. This score is normalized across all the nodes in the neighbourhood.

Brody et al. introduced a dynamic GAT, named GATv2 improves the previous state-of-the-art GAT model by Veličković et al. [21], which Brody et al. [1] claims is a static attention network while their improved method is a dynamic attention network. The static attention network computes the score to be the highest scoring key from all possible scoring functions $e \in F$, where F is the set of all scoring functions, whereas the dynamic attention function computes the score to be the highest scoring key for any scoring function $e \in F$ [1]. The GATv2 model can represent structures that the GAT cannot, and it can decay edge noise much faster [1].

The model proposed by Brody et al. is

$$e(\mathbf{h}_i, \mathbf{h}_j) = \mathbf{a}^\intercal \text{LeakyReLU}(\mathbf{W} \cdot [\mathbf{h}_i || \mathbf{h}_j])$$

which is just a simple reordering of the GAT model proposed by Veličković et al. [1][21], where the $\mathbf{a}^\intercal$ is brought out of the LeakyReLu function to give a layer of nonlinearity between $\mathbf{W}$ and $\mathbf{a}^\intercal$ [1]. $||$ denotes the concatenation operation. $\mathbf{W}$ and $\mathbf{a}$ are learned by the layer [1].

3.6 Proposed Model

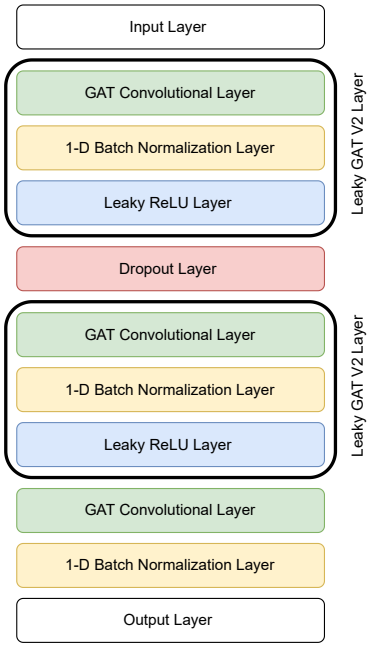


Figure 3.3: *The layers of the proposed neural network are shown here*

The proposed GNN model consists of 9 layers that create a deep node embedding of the graph and make link predictions using the embedding. The layers in this model are shown in Fig. 3.3. We call this model a Leaky dynamic GAT Link prediction model (LDGLP).

After the input layer, the first layer is a GATv2 layer proposed by Brody et al. This layer is used representation learning of the 1-hop neighbours of the nodes by using the message passing techniques mentioned above.

The second layer in the model is a 1-dimensional normalization layer used to normalize embedding values returned by the GATv2 layer. This normalizes the input and shifts the mean to 0. The benefits of this layer are twofold. It helps the activation function centered around 0 and can prevent further dead neurons due to leaky ReLu functions. It also reduces the sensitivity of the node embeddings to the previous embeddings or nodes in the neighbourhood with high degrees.

The third layer is a leaky ReLU layer. This layer does the operation

$$\text{Leaky ReLU} = \max(0, x) + \alpha \cdot \min(0, x)$$

on each element, where α is chosen to be 0.01. The purpose of this layer is to add much-needed non-linearity to the layers. This is especially important because GATv2 does a linear operation after the nonlinear activation function compared to GAT, which had a nonlinear activation function last. So without this non-linearity later, all of the layers after this can converge into a single linear layer. This layer is crucial so to mapping more complicated relations.

These three layers together are called the Leaky GATv2 layer as indicated in Fig.3.3. This layer is used to add the linearity back into the GATv2 layer to have both the advantages

of the GATv1 and GATv2 layers.

The fourth layer is the dropout layer. The dropping rate p is chosen to be 0.5. Several values between 0 and 1 were tested, with 0.5 returning the best results. This layer forces the neurons to rely less on their neighbouring neurons, allowing each neuron is better tuned to the result.

The next three layers are the GATv2 layer, followed by a 1D normalization layer and a Leaky ReLU layer. This is another leaky GATv2 layer. It is used to update the embedding to the 2-hop neighbours. There is not another dropout layer as dropouts slow down the convergence of the model.

Then there is another GATv2 layer followed by a normalization layer used to update the embedding to aggregate the 3-hop neighbours.

Then to make the link prediction layer, which is essentially a perceptron layer, where the output and the weight of the last layer are multiplied together and summed. This returns a value between 0 and 1 indicating the probability if there is a link between the two nodes or not. Closer to 1 is a higher chance of a link between the two nodes.

3.6.1 Methodology

The experiments are conducted on four graph datasets. These are the same datasets that are used by Molokwu et al. in their paper. Since these are the same datasets used, the accuracy of this experiment can be directly compared with the accuracy of the RLVECO and other models that they have compared with. The four datasets that were used in the

experiment are CiteSeer, Cora, PubMed-Diabete [22], and Facebook Page-Page [18].

All four of these datasets are heterogeneous graphs. CiteSeer and Cora are citation networks with each node belonging to various categories representing the field. PubMed-Diabetes is also a citation network with each node categorized into one of three types of diabetes. In each of these three datasets, the links represent the citations between the papers. Facebook Page-Page is a social network in which the nodes are classified into four categories as classified by Facebook. The links represent common likes between the various pages. All of these datasets are available in the PyTorch Geometric module as prepossessed datasets, with all the categorical values already represented as discrete values [14].

Since the input is given in graph format, labels must be generated from the graph dataset depending on if the link is in the graph or not. This is a complex process on a graph dataset. This process was done using DeepSNAP [16]. The DeepSNAP library is used to split the dataset into message passing edges, validation edges, and testing edges at a ratio of 0.85, 0.5, 0.1, respectively [16]. The message passing set is the edges used to train the model, similar to E^{train} in the experiment in chapter 2. The GNN is trained on this set. The validation edges are used to calculate the loss function for each epoch of training. The testing edges are used to determine the accuracy of the trained model after training. Some negative edges that exist that are not present in the graph dataset are also generated and placed in the validation and training sets at a 50:50 ratio. These edges are labelled accordingly, corresponding to 1 if they exist and 0 if they do not exist in the graph dataset.

These three sets are selected to be mutually disjoint.

The GNN link prediction model proposed in this thesis was implemented in Python using PyTorch and PyTorch-Geometric libraries [4] [14].

The model is evaluated using the following metrics: Accuracy (AC), Area Under the Receiver Operating Characteristic Curve (RO), F1-Score (F1), Precision (PC), and Recall (RC) and Support (SP). These metrics were found using the sci-kit learn library [15].

3.6.2 Hyperparameter Tuning

The hyperparameters were optimized by testing out various values for each parameter while keeping other parameters constant. The p-value for the dropout layer was selected by training the model with the following values for $p = \{0.05, 0.1, 0.2, 0.3, 0.4,0.5, 0.6, 0.7, 0.8, 0.9, 0.95\}$. Through multiple trials, keeping the other hyperparameters the same, the best value for p was determined to be 0.5.

Similarly, the hidden dimension is kept constant for each hidden layer as it is the best practice. The best-hidden value was determined by training the model with the hidden dimension set to the following values: {20, 40, 60, 80, 100, 120, 140, 180, 200}. Through this it was determined that the best-hidden value occurs when the hidden dimension is 80, so to get a more precise value, this was attempted again using the following values: {65, 70, 75, 80, 90, 95, 100}. The best value occurs at 85. This was selected to be the hidden dimension parameter.

The number of epochs was determined by training the model for 1000 epochs and keeping

track of the RO metric for each epoch on the training edges and validation edges. It was determined that the training and validation score increases throughout the 1000 epochs without the model overfitting, but the rate of increase significantly decreases after 100 epochs. Since it was infeasible to do 1000 epochs of training each time, 100 epochs were selected to train the model.

Since the model returns the probability of the link existing in the graph, closer to 1 means that the link exists and closer to 0 means that the link does not exist. All the metrics but RO expects a categorical value to be given rather than a probabilistic value, so a value $\gamma = 0.35$ was selected to place the probability into the categories.

$$\text{Cat(x)} = \begin{cases} 0 & \text{if } 1 - x \geq \gamma \\ 1 & \text{if } 1 - x < \gamma \end{cases}$$

The value of γ was determined by setting values of γ to the following values: {0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95}. The recall score and the precision scores were calculated for each value and chose the value that maximizes both, as we want to reduce false negatives and false positives given by choosing high γ values. The best value was found to be 0.3, but to get a more precise answer, this was repeated for the values of {0.25, 0.275, 0.3, 0.325, 0.35, 0.375, 0.4}. The best value was determined to be 0.35 and was used in the experiments to calculate the metrics for PC, RC, F1, and SP, while probability without any change was used for the RO score.

Model	Metric	Seer	Cora	Page-Page	Pubmed	Avg
LDGLP	PC	0.8366	0.8410	0.8969	0.9091	0.8709
	RC	0.8048	0.8110	0.9151	0.9010	0.8579
	F1	0.7918	0.8448	0.8992	0.8370	0.8432
	AC	0.8394	0.8280	0.8874	0.8346	0.8473
	RO	0.8902	0.9099	0.9527	0.9149	0.9169
	SP	1824	2116	68404	17732	22519
RLVECO [12]	PC	1.0	0.995	1.0	0.995	1.0
	RC	0.995	0.975	0.88	0.99	96
	F1	0.995	0.99	0.935	0.99	0.98
	AC	0.995	0.99	0.97	0.99	0.99
	RO	0.995	0.99	0.995	1.0	1.0
	SP	1941.5	1895.5	40126.5	15007	14743
HolE [12]	PC	0.87	0.875	0.925	0.955	0.91
	RC	0.845	0.84	0.985	0.95	0.91
	F1	0.845	0.85	0.95	0.955	0.90
	AC	0.85	0.86	0.97	0.96	0.91
	RO	0.85	0.84	0.98	0.95	0.91
	SP	1722	1808.5	39584.5	13315	14108

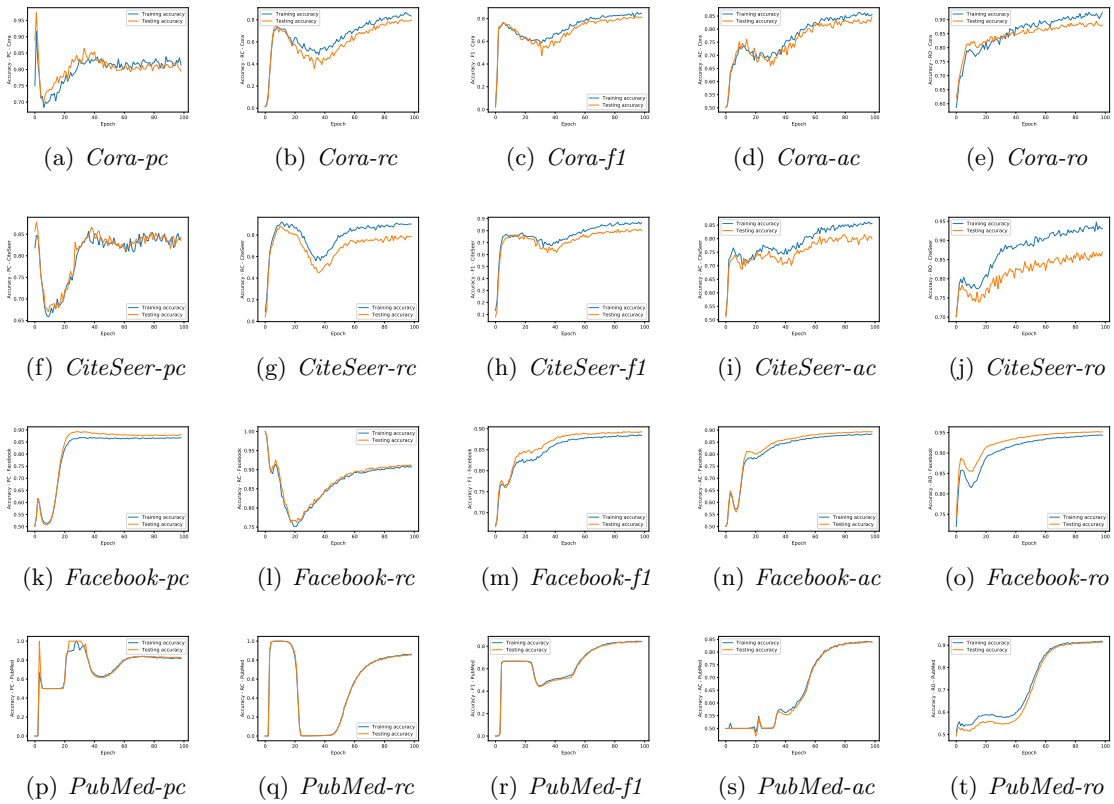
Table 3.1: *Table of the predictive power of LDGLP compared to a few standard models. The values of LDGLP was found during the experiment. The values of RLVECO and HoLE is gotten from Molokwu et al. with the values of positives and negative ties averages together.*

3.6.3 Experimental Result

Table 3.1 shows the performance scores of LDGLP compared with the standard vector embedding models used for link prediction when the various performance metrics are measured on the 4 datasets mentioned above. The metrics are measured on both positive and negative links. It is apparent that the RLVECO model outperforms the LDGLP model, but the LDGLP model can outperform the HoLE model in the RO metric. In all other metrics, the model performs comparatively to the HoLE model. It has an overall average accuracy of 87% across all of the datasets and all metrics. Since the LDGLP model outperforms the

HoLE model in the RO metric, it could be an indication that the model can outperform the HoLE model if γ was calculated in a more precise way since γ was used to categorize the probability into one of two categories for the PC, RC, F1, AC, and SP metrics.

Another possible reason why these models outperform the LDGLP model is that the LDGLP model is a homogeneous graph model, while both HoLE, and RLVECO are heterogeneous or knowledge graph models. Since all of the datasets tested are knowledge graphs rather than homogeneous graphs, the HoLE and RLVECO models have more information to use, while the LDGLP model ignores this information. The performance of these models may be different on a dataset that is a homogeneous graph.



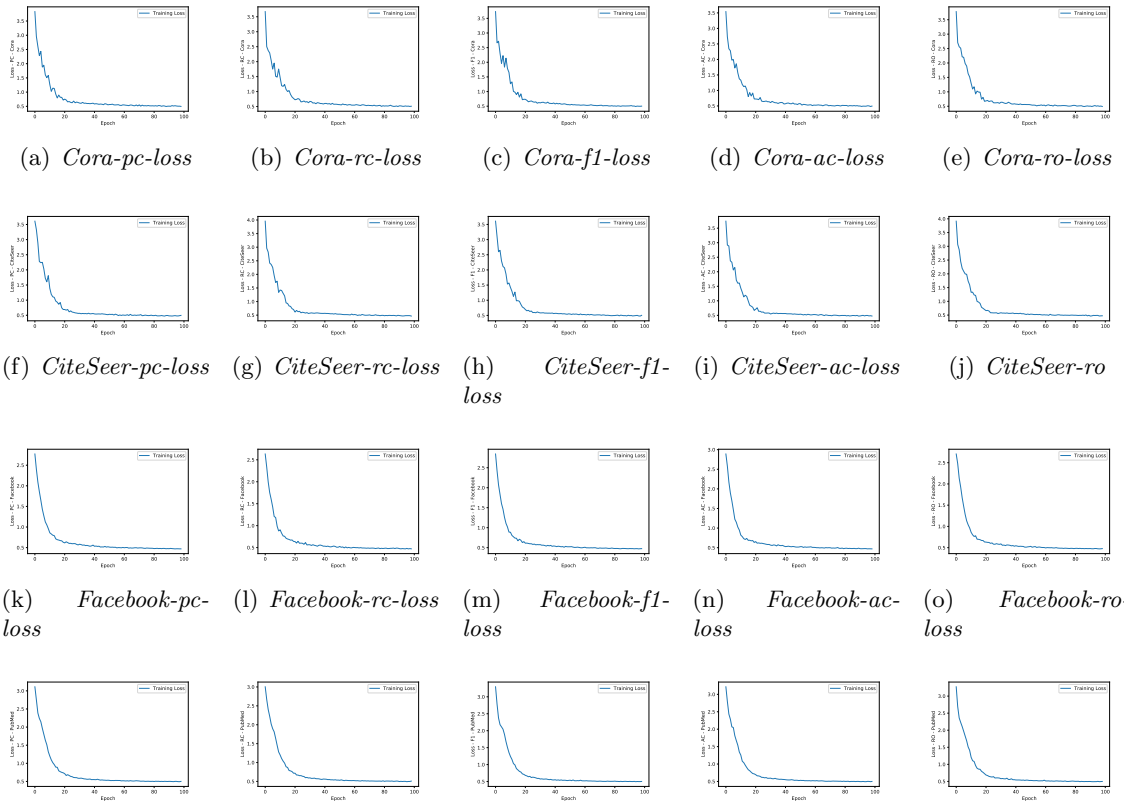
(a) *Cora-pc* (b) *Cora-rc* (c) *Cora-f1* (d) *Cora-ac* (e) *Cora-ro*

(f) *CiteSeer-pc* (g) *CiteSeer-rc* (h) *CiteSeer-f1* (i) *CiteSeer-ac* (j) *CiteSeer-ro*

(k) *Facebook-pc* (l) *Facebook-rc* (m) *Facebook-f1* (n) *Facebook-ac* (o) *Facebook-ro*

(p) *PubMed-pc* (q) *PubMed-rc* (r) *PubMed-f1* (s) *PubMed-ac* (t) *PubMed-ro*

Figure 3.4: *Training and validation accuracy for each metric that was tested during training. Blue represents training data and orange represents training data.*



(a) *Cora-pc-loss* (b) *Cora-rc-loss* (c) *Cora-f1-loss* (d) *Cora-ac-loss* (e) *Cora-ro-loss*

(f) *CiteSeer-pc-loss* (g) *CiteSeer-rc-loss* (h) *CiteSeer-f1-loss* (i) *CiteSeer-ac-loss* (j) *CiteSeer-ro*

(k) *Facebook-pc-loss* (l) *Facebook-rc-loss* (m) *Facebook-f1-loss* (n) *Facebook-ac-loss* (o) *Facebook-ro-loss*

(p) *PubMed-pc-loss* (q) *PubMed-rc-loss* (r) *PubMed-f1-loss* (s) *PubMed-ac-loss* (t) *PubMed-ro-loss*

Figure 3.5: *Training loss data for each metric that was tested during training*

Figure 3.4 and 3.5 show the training and validation graph and the loss function graph calculated during training after each epoch.

It is notable that the validation and training accuracy curves are high initially but quickly decrease, then gradually increase. This decrease is caused by the presence of the dropout layer. This prevents overfitting the data by reducing the chance that the neurons train on the noise. Notably, the model is not overfitted as the validation and testing curves are nearly identical. The loss function decreases exponentially but starts to slow down. This indicates that the training can be continued, but the changes in accuracy are not drastic.

The loss data is the same regardless of the various metric used, as it depends only on the loss function.

3.7 Summary

This chapter gave a basic introduction to neural networks and how the neural network is trained. A brief review of various layers is provided that are used in the proposed model. This naturally leads to the discussion of graph representation in neural networks. Graph embedding techniques were discussed as a method of graph representation for neural networks. Graph Neural Networks were introduced in this chapter. A graph neural network layer was proposed. An experiment was conducted to test the accuracy of the model compared with the other models in the literature review.

Chapter 4

Conclusions and Future Works

The link prediction problem on social networks has garnered a significant amount of research in the last few years due to the rise of online social networks. Existing literature demonstrated various approaches to link prediction analysis in social networks. These methods can be broadly classified into similarity metrics, probabilistic, and neural network methods. This thesis proposed two approaches to solve the link prediction problem. One approach is based on the similarity metrics, and the other one is based on the neural network. Both of these methods have their advantages and disadvantageous. The similarity metrics make assumptions regarding the model to make calculations simpler, while GNN can learn the dataset without model assumption.

A similarity metric (DILA) based on the latent neighbourhood similarity based on a locally adaptive similarity metric was proposed in this thesis. The main purpose of this metric was to find similarity scores between nodes that do not have any common neighbours. To do so, the metric determines the similarity of the neighbourhood vector which is calculated using the locally adaptive metric. The accuracy of the metric was measured using the av-

erage AUC score over 15 trials. In experiment 1, when considering all the nodes, the DILA model performed on average at 85.05% while the best performing existing model performed on average at 85.47%; which is less than a 1% difference in performance. This model had performed significantly better than the traditional state-of-the-art methods of performing link prediction in nodes without any common neighbours. In experiment 2, considering nodes without any common neighbours, the model performs on average of 82.66%, while the next best model performed on average at 77.06%. This is significantly better than all of the other metrics. This model assumed that given graph dataset was scale-free, but most of the datasets used for comparison were not scale-free. Future work can look to add a global constant to this model to measure how well a fit the dataset is to the power law and use that constant to correct the error due to the scale-free assumption. The proposed model can be tested to determine its performance on scale-free datasets. The notion of similarity can be extended further to test this model for third and fourth-order neighbours as well.

A graph neural network model (LDGLP) was proposed in this thesis. This is a homogeneous link prediction model. It was compared with two heterogeneous or Knowledge graph-based models. The LDGLP model utilizes graph neural network techniques to learn the dataset rather than using an external embedding technique that uses some objective function. This allows the model to be integral in a dataset that changes over time. This is one of the strengths of the proposed model. This model implicitly assumes that the dataset is homogeneous, but the datasets used for comparison are heterogeneous. This model did

not perform as well as the knowledge graph models, but it performed comparably in some metrics. The accuracy of the method was measured using six well-known metrics. The LDGLP model was able to achieve an average accuracy of 86.724% across all the metrics over all datasets. This model can be extended to be a knowledge graph model and compare its performance on the same datasets for future work.

Bibliography

[1] Brody, S., Alon, U., and Yahav, E. How attentive are graph attention networks? In *International Conference on Learning Representations* (2022).

[2] Broido, A. D., and Clauset, A. Scale-free networks are rare. *Nature Communications 10*, 1 (2019), 1–10.

[3] Bu, D. Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucleic Acids Research 31*, 9 (2003), 2443–2450.

[4] Fey, M., and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019).

[5] Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow - Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd Edition (2019)*, vol. 44. 2011.

[6] Grover, A., and Leskovec, J. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge*

Discovery and Data Mining (New York, NY, USA, 2016), KDD '16, Association for Computing Machinery, p. 855–864.

[7] Hagberg, A. A., Schult, D. A., and Swart, P. J. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference* (Pasadena, CA USA, 2008), G. Varoquaux, T. Vaught, and J. Millman, Eds., pp. 11 – 15.

[8] Hamilton, W. L. *Graph Representation Learning*, vol. 14. 2020.

[9] Jeon, H., and Kim, T. Community-adaptive link prediction. *ACM International Conference Proceeding Series Part F128770* (2017), 2–6.

[10] Kumar, A., Singh, S. S., Singh, K., and Biswas, B. Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications 553* (2020), 124289.

[11] Lusseau, D., Schneider, K., Boisseau, O. J., Haase, P., Slooten, E., and Dawson, S. M. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology 54*, 4 (2003), 396–405.

[12] Molokwu, B. C., Shuvo, S. B., Kobti, Z., and Kar, N. C. Social network analysis using knowledge-graph embeddings and convolution operations. *Proceedings - International Conference on Pattern Recognition* (2020), 6351–6358.

[13] Newman, M. E. J., and Watts, D. J. Scaling and percolation in the small-world network model. *Phys. Rev. E 60* (Dec 1999), 7332–7342.

[14] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[15] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[16] Rex Ying, Jiaxuan You, Z. Z. X. H. R. S. J. L. Deepsnap. https://github.com/snap-stanford/deepsnap/, 2020.

[17] Rossi, R. A., and Ahmed, N. K. The network data repository with interactive graph analytics and visualization. In *AAAI* (2015).

[18] Rozemberczki, B., Allen, C., and Sarkar, R. Multi-scale attributed node embedding. *Journal of Complex Networks 9*, 2 (2021).

[19] Srinivas, V., and Mitra, P. *Link Prediction in Social Networks. Role of Powe Law Distribution.* 2016.

[20] Van Rossum, G., and Drake Jr, F. L. *Python tutorial.* Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[21] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. *International Conference on Learning Representations* (2018). accepted as poster.

[22] Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (2016), ICML'16, JMLR.org, p. 40–48.

[23] Zachary, W. W. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research 33*, 4 (1977), 452–473.

[24] Zareie, A., and Sakellariou, R. Similarity-based link prediction in social networks using latent relationships between the users. *Scientific Reports 10*, 1 (2020), 1–11.