Distance-Aware Positional Encoding in Transformers for Improved Link Prediction

Ca’ Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

Computer Science and Information Technology Master’s Thesis
Year 2024-2025

Supervisor prof. Sebastiano Vascon

Graduand Biagio Barchielli (881423)

Abstract

Link prediction task aims to detect hidden relations between nodes, leveraging the graph's underlying structure while often integrating node features. The presence or absence of a link is usually defined by a threshold based on a real value related to the similarity between the pair of target nodes. There are mainly three types of link prediction methods: heuristic-based methods, latent-based methods, and content-based methods. Heuristic-based methods use a specific heuristic to compute the similarity between the target nodes. Each heuristic (Common Neighbor, Jaccard score, Adamic-Adar, etc.) is based on a given assumption that does not fit all the possible graphs. Latent-based methods leverage the graph's latent structural properties by usually incorporating the adjacency or the Laplacian matrix. The structural information is processed to build meaningful embeddings, which are then used to score the relation between the target pair without using the nodes' features themselves. Content-based methods do not use the structural information of the received graph; instead, they leverage the single features of each node. More recent methods (GNN, MPNN, Graph-Transformers) integrate all these approaches or at least content-based and latent-based methods to achieve a better and more complete representation of the nodes, that are then used to feed a decoder, usually an MLP, returning the link likelihood. In this experiment, the link likelihood between pairs of nodes is computed via an encoder-decoder setup. The transformer-encoder is fed with the distance-based positional embeddings and the node features for each vertex in the h-hop subgraph around the target nodes. The decoder, an MLP, then receives the aggregation of the encoder output target nodes and yields the link likelihood.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Link prediction algorithms aim to predict the presence of a direct connection between two nodes in a graph-structured environment. In its simplest form, this task requires finding the right decision boundary, given a score, to infer the presence or absence of a link. The methods implemented to generate such a score, correlated with the similarity between the pair of target nodes, can range from simple structural feature extraction to ML-based algorithms. Structural features (SF) can be captured via the usage of Heuristics, such as Common Neighbor [18], Jaccard Score [12], and Katz index [13]. On the other hand, SF themselves can be included using machine learning approach as seen in [6], [33], [20], [27].

In recent years, various models leveraging the graph's topology have managed to gather contextual information around a center node via Message Passing mechanism. Graph Neural Networks (GNNs), which include Message-Passing Neural Networks (MPNNs) [7], Graph Convolutional Networks (GCNs) [15], and Graph Attention Networks (GATs) [31], leverage such a mechanism to extract and aggregate information directly from the neighbors of a center node.

In a heterogeneous graph, link prediction becomes a multi-class classification problem. The goal is not only to find the link itself but also to assign it the correct label. In the majority of cases, homogeneous setups can be generalized to heterogeneous ones by integrating the differences in node and edge type [41]. Link prediction has a broad range of applications in numerous real-world scenarios, such as protein-to-protein interactions (PPI) [29], drug discovery [1], knowledge graph completion [48], recommender systems [36].

Link prediction algorithms can be generalized into three classes: Heuristic-based, Content-based, and Latent-based. Heuristic-based methods exploit some kind of heuristic, high or low order, to extract SF features employed directly in the likelihood prediction. Latent-based methods instead extract indirectly SF by factorizing the matrix representation of the graph, usually Adjacency or Laplacian, to generate representative embedding for each node, which is then used to score the link probability. On the other hand, Content-based methods leverage the node features solely to extract the final score [41]. More recent techniques, GNN-based, Graph Transformers [25], include both SF, either via context aggregation or heuristic inclusion, and node features to generate the final result.

1.1 Local and global Heuristic

Heuristic methods differ from each other by gathering different aspects of the graph's structural information; every heuristic is based on a strong assumption on when two nodes should link; when such assumption is not met, the method fails [41]. Each method has a "range" of action; such range is directly correlated to the order of the heuristic itself. The lower the order, the shorter the range of action, thus implying that low-order heuristics gather local information, whereas high-order ones have a wider (global) notion of the graph structure. The research [21] showed how the graph topology affects the heuristic methods' performance, empirically proving that low-order heuristics perform better in denser graphs. On the other hand, High-order heuristics generate better results when the local topology is poor, thus requiring a wider range of action.

1.1.1 Low-order Heuristics

Common Neighbors [18] between two nodes $u, v \in \mathcal{V}$ of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ defined as $CN(u, v)$ count the cardinality of the intersection between the nodes inside the h-hop neighborhoods. Let $\mathcal{N}_m^h = \{n \in \mathcal{V} | d(m, n) \leq h\}$ represent the h-hop neighboring set for any given node m, where $d(m, n)$ represents the shortest distance needed to reach the node n from m, i.e. number of hops. Common Neighbors is then defined as:

$$CN(u, v) = |\mathcal{N}_u^{(1)} \cap \mathcal{N}_v^{(1)}| \in [0, |\mathcal{V}|] \tag{1.1}$$

Jaccard Score [12] computes the ratio between the actual common neighbor and the total possible neighbors given the extracted subgraph around the target nodes u, v:

$$f_{Jaccard}(u, v) = \frac{|\mathcal{N}_u^{(1)} \cap \mathcal{N}_v^{(1)}|}{|\mathcal{N}_u^{(1)} \cup \mathcal{N}_v^{(1)}|} \tag{1.2}$$

Both CN and Jaccard Score are defined as one-order heuristics due to the single hop needed to compute such metrics.

Adamic-Adar [2] is a second-order heuristic computing the nodes' similarity as a value inversely proportional to the degree of the common neighbors around the input node u, v. Such computation discounts the nodes being connected to high-degree neighbors, weighting them as less relevant connections:

$$f_{AA} = \sum_{n \in \mathcal{N}_u^{(1)} \cap \mathcal{N}_v^{(1)}} \frac{1}{\log |\mathcal{N}_n^{(1)}|} \tag{1.3}$$

Resource Allocation [45] is a more aggressive version of Adamic-Adar which does not down-weight the degree of the analyzed nodes, removing in fact the $\log(\cdot)$ function.

$$f_{RA} = \sum_{n \in \mathcal{N}_u^{(1)} \cap \mathcal{N}_v^{(1)}} \frac{1}{|\mathcal{N}_n^{(1)}|} \tag{1.4}$$

Both AA and RA require computing the score up to two hops away from the center node, thus locating them in the second-order class.

1.1.2 High-order Heuristics

Katz Index computes a weighted sum of all the possible walk of length l $\forall l = 1, \ldots, \infty$ between the received nodes, as in AA and RA there is a penalty which in this case down-weight the longer paths.

$$f_{Katz}(u, v) = \sum_{l=1}^{\infty} \beta^l |\text{walks}^{(l)}(u, v)| \tag{1.5}$$

Where $\beta \in (0, 1)$ acts as a discount factor according to the walk length.

Rooted PageRank (RPR) compute the stationary distributions $[\pi_u], [\pi_v]$ for the received nodes u, v. Each stationary distribution $[\pi_n]$ for a node n is computed as a random walk starting from the node n with probability α to move randomly and $(1-\alpha)$ to teleport back to n. The heuristic is then computed as the sum between the likelihood in the long term to reach from u the node v and vice versa.

$$f_{RPR} = [\pi_u]_v + [\pi_v]_u \tag{1.6}$$

As reported in [41], heuristic methods can capture the similarity between the pairs of nodes only if the graph itself is aligned with the employed heuristic, and usually, most of the methods work just with a homogenous setup. Furthermore, the overlapping ratio between positive pairs of nodes scored with different heuristics is usually low, underlying the high amount of dissimilarity between the different methods [20].

1.2 Latent-feature methods

Latent-feature methods employ the matrix representation of the graph, which can either be the Adjacency matrix A or the Laplacian L [41]. For a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the Laplacian matrix is computed as the difference between the degree matrix $D \in [0, \Delta]^{|\mathcal{V}| \times |\mathcal{V}|}$ and the adjacency matrix $A \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$. The degree matrix D holds non-zero values only in the diagonal, i.e., diagonal matrix; such values represent the degree of each node in the graph $\mathcal{G}$. The parameter Δ represents the highest degree for a node in the graph $\mathcal{G}$ [4].

$$D = \begin{bmatrix} d_1 & 0 & 0 & \cdots & 0 \\ 0 & d_2 & 0 & \cdots & 0 \\ 0 & 0 & d_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d_{|\mathcal{V}|} \end{bmatrix} \tag{1.7}$$

$$L = D - A \tag{1.8}$$

The Laplacian matrix holds different properties, the first of which is that symmetry is preserved since we are summing two symmetric matrices themselves. This implies the matrix L to be positive-semi-definite (PSD), i.e. $\lambda_1, \ldots, \lambda_{|\mathcal{V}|} \geq 0$, furthermore each row of the Laplacian matrix sum up to 0, indicating that 0 represent the first eigenvalue λ_1 of L. Notably Spectral Clustering [23] and Normalized Cut

[26], both of them compute the cluster by applying K-Means algorithm onto the matrix $U \in \mathbb{R}^{\mathcal{V} \times l}$, or its normalized version, where the scalar l represents the number of eigenvectors taken into consideration as representative for the nodes projections. The recursive version of Normalized cut employs the usage of the second smallest eigenvalue of L, which represents the algebraic connectivity; such value indicates the connectivity strength in the starting graph [4].

In LP [17], the link likelihood is computed as the standard dot product of the node embedding retrieved from the matrix decomposition process. In general such methods start from either L or A decomposition into two submatrices $U \in \mathbb{R}^{|\mathcal{V}| \times k}$ and $V \in \mathbb{R}^{k \times |\mathcal{V}|}$ where the scalar k is an hyper-parameter, defining the chosen dimensionality reduction. The resulting matrices are multiplied to generate the final latent embedding $Z \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ used to predict the adjacency matrix $\hat{A}$. The latent matrices U and V are updated via gradient descent while optimizing the means-squared error between the predicted adjacency $\hat{A}$ and real one A.

$$\hat{A}_{ij} = z_i^T z_j \tag{1.9}$$

$$\mathcal{L} = \frac{1}{|\mathcal{E}|} \sum_{(i,j) \in \mathcal{E}} (A_{i,j} - \hat{A}_{i,j})^2 \tag{1.10}$$

Another approach is to employ the Laplacian L and compute the loss with the objective of minimizing the Euclidean distance, i.e., squared norm-two, between the latent embedding sharing a link:

$$\mathcal{L} = \sum_{(i,j) \in \mathcal{E}} ||z_i - z_j||_2^2 \tag{1.11}$$

The solution of this equation corresponds to the k eigenvectors, related to the k smallest eigenvalues. According to [25], even though latent embeddings encapsulate a notion of global information, they might worsen the sparsity due to the dimensionality reduction and the locality due to more "blended" features of the graph itself. Younger solution of latent-based methods [8] implemented an MLP as a link decoder, rather than a simple dot product, the model was fed with the aggregated target pair embeddings, using one of the following function: Hadamard, mean, absolute difference, squared difference as candidates.

1.3 Content-based methods

Content-based methods rely solely on the node features without extracting information from the surrounding structural context. Although the absence of any positional information leads to lower performance, on the other hand, differently from Heuristic-based methods and Latent-based methods, content-based methods do not suffer from **cold start** problem [41]. When a new node is inserted in the processed graph, its connectivity is lower than that of the other nodes, resulting in a lower degree and less overall richer topology. As shown in the study [34], the number of paths between two nodes enriches the enclosing topology, leading to higher LP performance. Furthermore, they analyzed that an untrained predictor (GNN-based) improves its performance linearly as the degree of the predicted node increases. Content-based methods are usually employed together with the previous techniques to achieve better performance, as also seen in [44], [34], [27].

1.4 Message passing based networks

GNN-based methods, gather contextual information via the implementation of message passing mechanism, in general such mechanism is performed in two steps. In the first phase, the information is gathered from the neighboring nodes (aggregation), with a predefined operation; in the second phase, the gathered message is combined with the center node itself (update). Thanks to this mechanism, GNN-based networks learn inherently to generate similar embedding for the connected nodes [10].

MPNN [7] message passing, applies a learnable function $M_t(\cdot)$ used to gather the information from the neighboring nodes. The gathered information is then combined via summation, invariant operation, to generate the message m_v^{t+1} for the center node v.

$$m_v^{t+1} = \sum_{u \in \mathcal{N}_v} M_t\big(h_v^t, h_u^t, e_{vu}\big) \tag{1.12}$$

The learnable function $M_t(\cdot)$ also receives the embedding of the edge connecting the center node with its neighbor u, whereas the parameter t represents the index of the actual MP layer. The center node v is updated by applying a learnable function $U_t(\cdot)$ combining the message m^{t+1} with the embedding of the center node h_v^{t+1}:

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{1.13}$$

GCN [15] performs the aggregation operation as a weighted sum of the node embeddings; the weight themselves can be computed in a similar fashion as Resource Allocator heuristic (1.4). Such weights are computed as the squared root of the neighborhood cardinalities product between the center node v and its neighborhood. The resulting weighting system down-weightes nodes with a higher degree, aggregating, in fact, more information from low-degree nodes.

$$h_v^t = \sigma\left(W^t \cdot \sum_{u \in \mathcal{N}_v \cup \{v\}} \frac{h_u}{\sqrt{|\mathcal{N}_v| \cdot |\mathcal{N}_u|}}\right) \tag{1.14}$$

The resulting sum is blended with a linear layer W and a non linear activation is applied, such as ReLU.

GAT [31] computes the attention weight in a similar fashion to [3], by using the same projector W, i.e., linear layer, for the attention weights and aggregation operation. The coefficients for the center node i e_{ij} $\forall j \in \mathcal{N}_i$ are computed as the dot product between a learnable vector a and the concatenation of the projected embedding Wh_i and Wh_j. Before normalizing the coefficient, LeakyReLU function is applied, followed by a softmax.

$$e_{ij} = \text{LeakyReLU}\left(a^T[\mathbf{Wh}_i || \mathbf{Wh}_j]\right) \tag{1.15}$$

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(e_{ik}\right)} \tag{1.16}$$

The attention weights $\alpha_{ij} \in [0,1]$ are employed in the weighted sum between the projected neighbors of i; differently from GCN, the center node in GAT does not retain any information of its previous state before the message passing operation. GAT extends the mp-attention mechanism to have multiple heads K; the different head results are aggregated either via concatenation or average to form the new node representation.

$$\mathbf{h}_i^{\text{single}} = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}\mathbf{h}_j\right) \tag{1.17}$$

$$\mathbf{h}_i^{\text{multi}} = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{k}\mathbf{W}^{k}\mathbf{h}_j\right) \tag{1.18}$$

$$\mathbf{h}_i^{\text{multi}} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{k}\mathbf{W}^{k}\mathbf{h}_j\right) \tag{1.19}$$

In **GraphSAGE** [10], the neighborhood information is aggregated via a tunable operation, Mean or Max aggregator. The neighboring nodes are combined differently depending on the type of aggregator function implemented. The Mean function computes a simple average pooling in the one-hop subgraph of a center node v, the result is multiplied with the matrix W, and then a nonlinear function is applied.

$$h_v^k = \sigma\left(W^k \cdot \text{MEAN}(\{h_v^{k-1}\} \cup \{h_{\mathcal{N}(u)}^{k-1} \,| u \in \mathcal{N}(v)\right) \tag{1.20}$$

On the other hand, the Max aggregator function computes max pooling on the set of projected embeddings, center node neighbors, given the matrix W_{pool}, a vector b, and a nonlinearity. Such result $h_{\mathcal{N}(v)}^k$ is concatenated with its center node h_v^{k-1} representation and multiplied with a learnable matrix W^k.

$$h_{\mathcal{N}(v)}^k = \max\left\{\sigma(W_{\text{pool}}h_u^{k-1}) \mid u \in \mathcal{N}(v)\right\} \tag{1.21}$$

$$h_v^k = \sigma\left(W^k \cdot \text{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k)\right) \tag{1.22}$$

GraphSAGE implements a custom loss function, which goals are to minimize the negative log-likelihood between the target node v and its positive pairs $-\log\left(\sigma(z_v^T z_u)\right)$ $\forall(u,v) \in \mathcal{E}^+$, while maximizing the negative log-likelihood expected value of the negative pairs $-\log\left(\sigma(-z_v^T z_n)\right)$ $\forall(u,v) \in \mathcal{E}^-$.

$$\mathcal{J}_G(z_v) = -\log\left(\sigma(z_v^T z_u)\right) - Q \cdot \mathbb{E}_{n\sim P_{neg}(v)} \log\left(\sigma(-z_v^T z_n)\right) \tag{1.23}$$

The parameter Q defines the number of negative samples whereas $P_{neg}(v)$ represents the negative distribution of link for the node v.

Although Message passing mechanism can capture contextual information, it lacks the ability to include long-range dependencies, complex interactions, and heterogeneous structures [25].

Graph Transformers represent a family of models implementing the attention mechanism [30], [3], while including structural features to leverage the graph spatial information. Such models can overcome the limit of message passing mechanism by effectively capturing both local and global relationships [24]; on the other hand, pure **Multi Head Attention** (MHA) has a high computational cost scaling quadratically with respect to the number of received nodes. The Attention mechanism lacks any spatial notion requiring the integration of structural features. Such features can be included via the implementation of local positional encoding, as distance-based OHE [31], or global ones by leveraging the matrix representation of the graph as shown in 1.2. Other works [24], [11] customize directly the attention mechanism to include spatial notions in the resulting embeddings:

$$\alpha_{ij}^{\text{local}} = \frac{\exp\left(g\left(\mathbf{x}_i, \mathbf{x}_j\right) \cdot b_{ij}\right)}{\sum_{k \in \mathcal{N}(v_i)} \exp\left(g\left(\mathbf{x}_i, \mathbf{x}_k\right) \cdot b_{ik}\right)} \tag{1.24}$$

$$\alpha_{ij}^{\text{global}} = \frac{\exp\left(g\left(\mathbf{x}_i, \mathbf{x}_j\right)\right) + c\left(\mathbf{A}, \mathbf{D}, \mathbf{L}, \mathbf{P}\right)_{ij}}{\sum_{k=1}^{N} \exp\left(g\left(\mathbf{x}_i, \mathbf{x}_k\right)\right) + c\left(\mathbf{A}, \mathbf{D}, \mathbf{L}, \mathbf{P}\right)_{ik}} \tag{1.25}$$

The parameter b_{ij} is the local attention bias term which is adjusted given the distance between the nodes $i$and j. On the other c is the function computing the global attention bias using the **Adjacency** matrix $\mathbf{A}$, the **Degree** matrix $\mathbf{D}$, the **Laplacian** matrix $\mathbf{L}$ and the **PageRank** 1.1.2 vector $\mathbf{P}$.

GNN-based methods are usually divided into two paradigms, **Node-based** methods, and **Subgraph-based** methods; the former regards the application of a GNN model directly to the graph, whereas the latter extract a subgraph around the target nodes, which is then fed to the GNN model. Each paradigm employs the usage of a link decoder to score the relations between the returned embedding; such decoder can be a learnable function, i.e., MLP, or a simple binary operation, i.e., dot product. In Node-based setup, the link decoder is fed with the target nodes embedding, whereas in Subgraph-based methods, the approach might vary from using just the target nodes to employing the whole extracted subgraph as in [35].

1.4.1 Node-based methods

Nonprobabilistic **Graph AutoEncoder** (GAE) [16] compute the node representations using the GCN architecture (1.14), dot product is implemented as simple link decoder receiving the target nodes representations. The GCN architecture is updated by minimizing the binary cross entropy loss between the predicted adjacency matrix $\bar{A}$ and the actual one A:

$$\tilde{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{1.26}$$

$$Z = \text{GCN}(X, A) = \tilde{A}\text{ReLU}(\tilde{A}XW_0)W_1 \tag{1.27}$$

$$\bar{A} = \sigma(ZZ^T) \tag{1.28}$$

$$\mathcal{L} = \sum_{i \in \mathcal{V}, j \in \mathcal{V}} -A_{ij} \log\left(\bar{A}_{ij}\right) - \left(1 - A_{ij}\right) \log\left(1 - \bar{A}_{ij}\right) \tag{1.29}$$

The parameter D is the degree matrix, whereas the tensor X represents the graph nodes' features; in the absence of such features, GAE can employ a simple OHE matrix I, where each row is characterized by a unique vector with just one entry equal to one.

Variational Graph AutoEncoder (VGAE) [16] computes the probability distribution $p(A|Z)$ of the real adjacency matrix A given the sampled node embedding matrix Z. The distribution $p(Z)$ of the generated embedding Z follows a standard Normal distribution $\mathcal{N}(z_i|0, I)\ \forall i = 1, \ldots, |\mathcal{V}|$. Given the intractability of the posterior distribution, $p(Z|X, A)$ VGAE uses two two-layer GCNs, GCN_μ and GCN_σ to compute respectively the mean vector μ_i and the standard deviation vector σ_i for each node. The final score is computed as the dot product between the target pair mean vector embeddings $\hat{A}_{ij} = \sigma(\mu_i^T \mu_j)$.

$$\mu_i = \mathrm{GCN}_\mu(X, A) = A \cdot \mathrm{ReLU}(AXW_0)W_\mu, \tag{1.30}$$

$$\log\sigma_i = \mathrm{GCN}_\sigma(X, A) = A \cdot \mathrm{ReLU}(AXW_0)W_\sigma, \tag{1.31}$$

$$z_i = \mu_i + \epsilon_i \cdot \sigma_i, \quad \text{where } \epsilon_i \sim \mathcal{N}(0, I), \tag{1.32}$$

$$\bar{A}_{ij} = \sigma(\mu_i^T \mu_j) \tag{1.33}$$

VGAE objective is to maximize the evidence lower-bound $\mathbb{E}_{q(Z|X,A)}$ between the log-likelihood $p(A|Z)$ and the Kullback-Leibler divergence (KL) of the posterior distribution $q(Z|X, A)$ and $p(Z)$.

$$p(A|Z) = \prod_{i\in\mathcal{V}} \prod_{j\in\mathcal{V}} p(A_{ij}|z_i, z_j), \tag{1.34}$$

$$q(Z|X, A) = \prod_{i\in\mathcal{V}} q(z_i|X, A), \quad \text{where } q(z_i|X, A) = \mathcal{N}(z_i|\mu_i, \sigma_i^2). \tag{1.35}$$

$$\mathcal{L} = \mathbb{E}_{q(Z|X,A)}\left[\log p(A|Z)\right] - \mathrm{KL}\left[q(Z|X, A)\|p(Z)\right]. \tag{1.36}$$

Node-based methods don't need to generate the node embedding for every pair to score, thus making the training process more efficient. On the other hand, computing the node embeddings without considering the target node pair reduces the expressivity of the models themselves [41]. The research [33] presented a node-based setup that mitigates such a problem by integrating SF after the embedding computation, preserving the efficiency while improving the expressivity of the node representations.

1.4.2 Subgraph-based methods

Subgraph-based methods extract a subgraph around the target nodes pair (u, v), which is processed and fed to a link encoder generating the node embeddings. The extraction process can be computed by picking all the nodes having distance being at most h-hop away from at least one of the target nodes $\mathcal{G}^h_{u,v} = \{n \in \mathcal{V} | d(u, n) \le h\ \vee\ d(v, n) \le h\}$, as in SEAL [42]. As shown in [27], the subgraph extraction can also be performed employing a (random walk)-based algorithm. The research [37] instead generated the subgraph as the set of all nodes occurring at least in one path of length at most $h + 1$ between the target nodes u, v:

$\mathscr{G}^h_{u,v} = \{n \in \mathcal{V} | \exists\ l \in [1, h+1],\ \ n \in \text{walks}^{(l)}(u,v)\}$. Posterior to the graph extraction, a node labeling is applied to the aggregated neighborhood; such labeling helps the link encoder to distinguish the pair of target nodes from the extracted vertices [41]. The link encoder is then fed with the extracted subgraph yielding the subgraph embeddings; the link decoder receives the aggregated embeddings and scores the link likelihood.

SEAL extract the h-hop subgraph around the target nodes and compute the node labeling for each extracted node. The method applies a **Double Radius Node labeling** (DRNL), in which the target nodes u, v receive the label 1, whereas, for any other node n, its label is calculated as the sum of the distances between n and the target nodes $l(n) = d(u,n) + d(v,n)$. DRNL assigns larger labels to further nodes, i.e., with a larger radius from the center nodes. Thus, two nodes share the same label if their double radius is the same. Another proposed configuration of DRNL computes the label for nontarget node as:

$$d_v = d(n,v) \tag{1.37}$$

$$d_u = d(n,u) \tag{1.38}$$

$$d_{uv} = d(n,u) + d(n,v) \tag{1.39}$$

$$l(n) = 1 + \min(d_u, d_v) + \frac{d_{uv}}{2}[\frac{d_{uv}}{2} + (d_{uv} \mod 2) - 1] \tag{1.40}$$

The DRNL labels are transformed into OHE vectors and concatenated with the node features, if available. The resulting features are fed to a GNN, and aggregated target pair representations are scored using an MLP. Binary cross entropy between the predicted adjacency matrix and the real one was implemented as the objective function.

Subgraph-based methods model the nodes representation, forcing the embedding to be "target-node" aware; such condition allows these techniques to have a higher expressive ability at the cost of running the whole process for every node pair to score [41].

1.4.3 Positional encoding

Positional node embedding can be implemented to encode global or local structural information of the graph, attention-based models can leverage such embeddings to overcome the limitation, (lack of order), of the attention mechanism [25]. Positional encoding can integrate the topological information depending on the range of the represented structural features.

Local positional encoding, as described in [25], can be computed by gathering information from the extracted subgraph or from the direct neighbors. One approach is to generate for each node v two OHEs of size h where h represents the maximum number of hops needed to connect the furthest point n from the target nodes u, v. For the node n, its positional representation p^v_n with respect to the node v has a one only in the i-th cell where the integer i represents the shortest path distance between n and v.

$$p^v_n = [\mathbb{I}_{d(n,v)=1}, \dots, \mathbb{I}_{d(n,v)=\max}] \tag{1.41}$$

Where $\mathbb{I}$ represent the indicator function, as proposed in [22], the final positional encoding for the node n can be computed as simple sum $p_n = p_n^u \oplus p_n^v$.
Another possible approach is to compute a kernel function K, which can be a learnable function or a heuristic, directly on the extracted node subgraphs. Such kernel computes the similarity between the received subgraph, yielding a real value; each node is assigned with a positional encoding of size l where l represents the number of sampled nodes:

$$p_i = [K(G_i, G_1), \ldots, K(G_i, G_l)] \tag{1.42}$$

Global positional encoding, as described in [25], represents the position, within the graph embedding space, of the nodes themselves. One approach is to leverage the spectral properties of the Laplacian matrix L [4] by assigning to the i-th node in the graph the related i-th eigenvector of L up to component k.

$$p_i = [u_{i,1}, u_{i,2}, \ldots, u_{i,k}] \tag{1.43}$$

Another approach is to compute positional encoding by utilizing diffusion or random walk techniques such as PPR 1.1.2.

$$p_i = [\pi_{i,1}, \pi_{i,2}, \ldots, \pi_{i,|\mathcal{V}|}] \tag{1.44}$$

Where $\pi_{i,j}$ represents the probability in the long term to move from node i to node j after performing a random walk.

1.4.4 Labeling trick

Labeling Trick assigns to each node $n \in V'$ in a subgraph $\mathcal{S} = (V', E')$ extracted from a graph $\mathcal{G} = (V, E)$, where $V' \subset V$ and $E' \subset E$, a label $l_n \in \mathbb{R}$. Such label represents the state of the node n with respect to the target nodes u and v; sufficient condition for this labeling system is to render the nodes u, v distinguishable from the rest of the vertices in the subgraph $\mathcal{S}$ [41]. For instance, in the research [35], they assigned the label zero to the target nodes $l_u, l_v = 0$, whereas to the one-hop neighborhood nodes of u received the labels one while the one-hop neighborhood of v was assigned with label two. The nodes whose distance was greater than one from the target nodes received label three. As shown in [43] also, a distance encoding-based labeling trick can be computed, where each nontarget node is assigned with the distance from the target nodes set. Another solution, as mentioned in [32], is a simple zero-one labeling where nontarget nodes receive the label zero otherwise one, or vice-versa. SEAL [42] framework applies the DRNL labeling system, which assigns the unique label zero to the nodes u, v while iteratively incrementing the label values as the analyzed nodes get further from target nodes. While computing the distances $d(n, v), d(n, u)$ for a node n given the nodes u, v, DRNL masks the opposite target node to use the real distance of the node with respect to the analyzed target node. In other word when DRNL computes $d(n, v)$, the target node u and all its edges are masked, same scenario for $d(n, u)$ [41]. GIN [40] apply a subset-(labeling trick), in which not all the target nodes receive a label but just a subset of them, i.e., the center node. In this research, they assigned a unique color just to the center nodes of the extracted subgraphs before the message-passing mechanism.

Chapter 2

Related Works

2.1 Link Prediction with Persistent Homology: An Interactive View

Persistent homology is a mathematical framework that studies the structural information of a space. Structural information is usually analyzed in a graph $G = (V, E)$ by detecting connected components, loops, or cavities [38]. The mathematical framework remains persistent, usually through a filtration process that leverages the presence of the so-called simplicies. The research proposed a method based on the persistent Homology framework, which exploits the number of loops near the target nodes to predict their connection. The number of loops is directly related to the structural complexity in the proximity of the target nodes. According to the research, structural complexity can be an indicative sign of a link's presence. Given the target nodes u and v, the h-hop enclosing subgraph is extracted for each node, $V_u^h = \{m | d(u, m) \leq h\}$ and $V_v^h = \{m | d(v, m) \leq h\}$. The persistence image $PI(u, v)$ is computed using the subgraphs intersection $V_{uv} = V_u \cap V_v$. The simplices matrix of size $m = |E| + |V|$, which represents the relationships between all the nodes and edges in G, is reduced through an iterative algorithm to extract the persistent image. In each subgraph the related target node u, v are used as anchors to compute the filtering function f, such function return the the sum of the shortest path distance between the analyzed node and the target ones.

$$f(n) = d(n, v) + d(n, u) \tag{2.1}$$

The study implemented as link encoder an l-layer GCN, the target nodes embedding h_u and h_v are aggregated as $h_{uv} = (h_u - h_v)^2$ and then concatenated to the persistent image $PI(u, v)$. The resulting embedding was then fed to a link decoder consisting of a two-layer MLP. The MLP yielded the predicted distance between the target nodes, which was tampered within the unit range with the sigmoid function $\sigma(d(u, v)) = \frac{1}{1+e^{d(u,v)-2}}$.

2.2 Learning from Counterfactual Links for Link Prediction

Link prediction tasks inherently deal with the lack of completeness due to the edge masking operation applied throughout the training process [14]. Counterfactual links are synthetic-type of links that are generated to analyze the causal relationship between nodes. The presence of a new hypothetic link can unveil hidden relationships between neighborhoods of nodes. The research [44] presented a new data augmentation technique, which leveraged a pre-trained node embedder to spot the counterfactual links. For each sample in the training set, the counterfactual links, both positive and negative, were generated, so the link encoder could exploit each of the new hypothetical connections during the training epochs. A clustering algorithm was applied to the input graph to generate the so-called treatment matrix $T \in \{0,1\}^{|V|\times|V|}$. The cell T_{ij} was set to true if the node pair ij belonged to the same cluster. To detect the counterfactual link, they used the state-of-art of that time to learn the node embeddings $X \in \mathbb{R}^{|V|\times F}$. Formally, for each link (v_i, v_j) in the training set, an optimization problem to find the "closest" pair of nodes (v_a, v_b) not already connected by a link within the radius of the hyper-parameter γ:

$$\begin{aligned}(v_a, v_b) = \arg \min_{v_a, v_b \in V} \{&d(x_i, x_a) + d(x_j, x_b) \\ &\mid T_{a,b} = 1 - T_{ij}, d(x_i, x_a) + d(x_j, x_b) \leq 2\gamma\}\end{aligned} \tag{2.2}$$

$$T^{CF}_{i,j}, A^{CF}_{i,j} = \begin{cases} 1 - T_{i,j} \cdot A_{a,b}, & \text{if } \exists (v_a, v_b) \in \mathcal{V} \times \mathcal{V} \text{ satisfies Eq. (2.2);} \\ T_{i,j} \cdot A_{i,j}, & \text{otherwise.} \end{cases} \tag{2.3}$$

In this setup, they implemented a GNN as a link encoder and an MLP as a link decoder to predict the real adjacency matrix A and the counterfactual one A^{CF}. Throughout the training, the encoder-decoder setup learned to minimize the distance between the cross-entropy of the existing link w.r.t. the cross-entropy of the counterfactual ones.

2.3 A Topological perspective on demystifying GNN-based link prediction

The study proposed a new heuristic called Topological Concentration (TC), which measured the average level of intersection between the node and its neighbors in the h-hop subgraphs. They empirically proved how TC is a more significant metric for Link Prediction than any other node-level topological metrics such as degree or subgraph density. According to the research [34], a higher intersection rate between the subgraphs indicates a denser local topology, leading to higher LP performance.

$$C_i^{K,i} = \mathbb{E}_{v_j \sim \mathcal{N}_i^j} I(\mathcal{S}_i^K, \mathcal{S}_j^K) = \mathbb{E}_{v_j \sim \mathcal{N}_i^j} \frac{\sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \beta^{k_1+k_2-2} |\mathcal{H}_i^{k_1} \cap \mathcal{H}_j^{k_2}|}{\sum_{k_1=1}^{K} \sum_{k_2=1}^{K} \beta^{k_1+k_2-2} g(|\mathcal{H}_i^{k_1}|, |\mathcal{H}_j^{k_2}|)}$$

Where the term $|\mathcal{H}_i^{k1} \cap \mathcal{H}_j^{k_2}|$ represents the actual cardinality of the intersection between the two neighborhoods, on the other hand $g(\mathcal{H}_i^{k_1}, \mathcal{H}_j^{k_2})$ represent the total possible number of intersections between neighbors that are k_1 and k_2 hop away. As the **Katz index** heuristic (1.5), the parameter $\beta \in [0, 1]$ represents a decaying factor that discounts further nodes. The study also showed that by updating the adjacency matrix used in the message massing process, with the nodes contributing more TC, the LP performance on the used datasets incremented.

$$\hat{A}_{ij}^{\tau} = \begin{cases} \hat{A}_{ij}^{\tau-1} + \gamma \frac{\exp g_{\theta_g}(N_i^{\tau-1}, H_j^{\tau-1})}{\sum_{j=1}^{n} \exp g_{\theta_g}(N_i^{\tau-1}, H_j^{\tau-1})} & \text{if } A_{ij} = 1 \\ 0 & \text{if } A_{ij} = 0 \end{cases}$$

Where g_{θ_g} is the link predictor, $H_j^{\tau-1}$ represents the nodes embeddings yielded by the link encoder (GNN) whereas $N_i^{\tau-1} = \hat{A}H^{\tau-1}$ represents the average node embeddings.

2.4 Few-shot graph link prediction with domain adaptation

The research [46] presented an adversarial training framework for LP, implementing a GNN as link encoder and generator while reducing the number of needed samples in the dataset with an MLP as a discriminator. The role of the generator was to yield domain-agnostic embeddings for the target node subgraphs. At the same time, the discriminator was trained to recognize if the received subgraph embeddings belonged to the same domain. A graph pooling operation for each subgraph embedding matrix was applied to get a vector representation of each neighborhood. The link decoder, 2-layer MLP, received the concatenated target nodes embedding of the GNN and yielded the link likelihood. The research reported its study on m domains, of which one with just one "shot," i.e., small graph, whereas the other $m-1$ with k "shots." The study objective was to develop a model that leverages the domain differences to improve the performance in the domain with one shot of data. The GNN and the link decoder underwent through 20-iteration warm-up training without the discriminator. According to the study, the warm-up iterations make the learning process for the discriminator easier since a non-optimal parameters initialization for the generator could make it challenging to perplex the discriminator. The total loss is computed as a subtraction between the link encoder-decoder cross-entropy and the discriminator loss.

$$\mathcal{L}_{total}(g,c) = \mathbb{E}\left[\sum_{i=1}^{m} \ell(c(g(S^{i}_{u,v})), y^{i}_{u,v}) - \alpha \cdot \sum_{i=1}^{m}\sum_{j=1}^{m} \ell(D(g(S^{i}_{u,v}), g(S^{j}_{u',v'})), d^{i,j})\right]$$

Where g and c represent the GNN and the link decoder MLP respectively, the labels $y^{i}_{u,v}$ and d^{ij} represent the ground truth for the link likelihood and the domain information for the discriminator. The parameter α weights the contribution of the discriminator error in the learning process.

2.5 Linkless Link Prediction via Relational Distillation

Message-passing based network, as GNN, manage to gather the contextual information around the neighboring nodes which can lead to high computational time [9]. Simpler models which do not rely on this mechanism, such as MLP, can be instead implemented with less cost. On the other hand MLP can't gather topological information from the neighborhood of a given node, such problem can be circumnavigated by distilling knowledge from a more complex model. For each anchor node v the neighborhood $\mathcal{C}_v$ was extracted representing the union between the nearby neighbors $\mathcal{C}_v^N$ and some random sampled nodes C_v^R. The research presented three different approaches to distill knowledge from a teacher model, a GNN, into a simpler model, MLP. In the first set up they simply trained the MLP as a standalone model with binary cross entropy between the predicted MLP labels and the ground truth, $\mathcal{L}_{sup}(\hat{y}_{i,j}, a_{i,j})$. The second set up $\mathcal{L}_{LM}$ aimed to minimize as before the error of the student model with $\mathcal{L}_{sup}(\hat{y}_{i,j}, a_{i,j})$ while also trying to match the MLP prediction with the teacher model $\mathcal{L}_{match}(\hat{y}_{i,j}, y_{i,j})$.

$$\mathcal{L}_{LM} = \sum_{(i,j)\in\mathcal{E}\cup\mathcal{E}^-} \lambda\mathcal{L}_{sup}(\hat{y}_{i,j}, a_{i,j}) + (1-\lambda)\mathcal{L}_{match}(\hat{y}_{i,j}, y_{i,j})$$

In the above equation λ is an hyper-parameter to weight the importance of each loss, $\hat{y}_{i,j}$ represent the MLP prediction whereas $y_{i,j}$ the teacher model output. The second proposed setup regarded as before, to minimize the error of the student model itself while "aligning" the GNN node embeddings, H, with the MLP, $\hat{H}$.

$$\mathcal{L}_{RM} = \sum_{(i,j)\in\mathcal{E}\cup\mathcal{E}^-} \lambda\mathcal{L}_{sup}(\hat{y}_{i,j}, a_{i,j}) + (1-\lambda)\sum_{i\in\mathcal{V}}\mathcal{L}_{match}(\hat{h}_i, h_i)$$

The last proposed approach combined a ranking loss, $\mathcal{L}_{LLP_R}$, with the goal of teaching the student model to extract relational knowledge around the anchor node v, with a distribution based loss. Such loss $\mathcal{L}_{LLP_D}$ was implemented as KL divergence, for each node i belonging to the anchor-extracted subset of nodes C_v.

$$\mathcal{L}_{LLP_R} = \sum_{v\in\mathcal{V}}\sum_{\{\hat{y}_{v,i},\hat{y}_{v,j}\}\in\hat{y}_v} \max\{0, -r\cdot(\hat{y}_{v,i}-\hat{y}_{v,j})+\delta\} \tag{2.4}$$

$$\tag{2.5}$$

$$r = \begin{cases} 1 & \text{if } \hat{y}_{v,i}-\hat{y}_{v,j} > \delta \\ -1 & \text{if } \hat{y}_{v,i}-\hat{y}_{v,j} < \delta \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

$$\mathcal{L}_{LLP_D} = \sum_{v\in\mathcal{V}}\sum_{i\in\mathcal{C}_v} \frac{\exp(y_{v,i}/\tau)}{\sum_{j\in\mathcal{C}_v}\exp(y_{v,j}/\tau)} \log\left(\frac{\exp(\hat{y}_{v,i}/\tau)}{\sum_{j\in\mathcal{C}_v}\exp(\hat{y}_{v,j}/\tau)}\right) \tag{2.7}$$

The hyper-parameter δ represents a margin to, reducing the strictness avoiding possible noise and deviation to be distilled into the student model. Finally the loss for the final set up was computed as a weighted sum between $\mathcal{L}_{sup}$, $\mathcal{L}_{LLP_R}$ and $\mathcal{L}_{LLP_D}$:

$$\mathcal{L} = \alpha\cdot\mathcal{L}_{sup} + \beta\cdot\mathcal{L}_{LLP_R} + \gamma\cdot\mathcal{L}_{LLP_D}$$

The weights, α, β, γ were hyper-parameters mediating the contribution of each loss.

2.6 Link Prediction for Flow-Driven Spatial Networks

The research [35] focused on LP tasks for flow-driven spatial networks; such structures are enclosed in an Euclidean space and follow physical constraints. Given the target nodes, the h-hop enclosing subgraph is extracted from a graph $G = (V, E)$, the line graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ was generated such that each edge belonging to the input subgraph was represented as a node. If two edges were incident in the original subgraph, their respective representation as nodes in the line-graph would have been adjacent. Each node embedding $n'_i \in \mathcal{V}$ in the line graph $\mathcal{G}$ representing the edge $(n_i, n_j) \in E$ was computed as the difference between the two nodes $n_j, n_i \in V$ connected by the edge (n_i, n_j). As Labeling trick, they assigned to the target nodes the value zero, whereas the target nodes' neighborhood nodes were assigned with either one, If a node belonged to the first target node, otherwise, two. All the remaining nodes that did not belong to the one-hop subgraph of target nodes were assigned with the value three. The line-graph node embeddings were then fed to an attention message passing mechanism called GAV layer, defined as linear projection $\phi_\theta^{(1)}$ followed by an MHA mechanism with res connection, as in the normal encoder-transformer set-up [30].

$$\tilde{n}'_i = \text{MultiHeadAttention}(\phi_\theta^{(1)}(n'_i), \phi_\theta^{(1)}(N_i)) \tag{2.8}$$

The final part of GAV consisted of another linear projection $\phi_\theta^{(2)}$ in which values were then tampered with the activation function tanh.

$$s_i = \tanh(\phi_\theta^{(2)}(\tilde{n}_i + \phi_\theta^{(1)}(n'_i))) \tag{2.9}$$

$$\hat{n}_i = s_i \cdot n'_i \tag{2.10}$$

The choice of the activation function played a major role in the preservation of the structural property of the original node $n_i \in \mathcal{V}$, geometrically speaking, the weights returned by the last projection of the GAV layer applied the input nodes resulted in simple scaling and/or rotation. The link decoder $\phi_\theta^{(3)}$, an MLP, received for each starting target node, $n_i, n_j \in V$ the set of refined vectors $\mathcal{C}_{N(n_i)}, \mathcal{C}_{N(n_j)}$, in the line graph, which were created from the edges incident to the actual target node $v_i, v_j \in V$.

$$\hat{y}^t_{ij} = \phi_\theta^{(3)}\left(\text{mean}\left(\mathcal{E}_{\mathcal{N}(n^t_i)}\right) \Big\| \text{mean}\left(\mathcal{E}_{\mathcal{N}(n^t_j)}\right)\right) \tag{2.11}$$

The two sets of refined vectors were then concatenated and fed to MLP, yielding the likelihood, the error was scored using binary cross-entropy.

$$\mathcal{L}_{\text{BCE}} = \frac{-1}{|\mathcal{E}|} \sum_{i,j \in \mathcal{E}} y^t_{ij} \log(\hat{y}^t_{ij}) + (1 - y^t_{ij}) \log(1 - \hat{y}^t_{ij}) \tag{2.12}$$

2.7 On the Impact of Feature Heterophily on Link Prediction with Graph Neural Networks

The study analyzed the impact of graph-heterophily, i.e., neighboring nodes whose class labels differ, on LP tasks. According to the research [47], the growth of the neighbor's similarities scores positively affects the rate of change for the LP scores. They computed the node features similaritiy ϕ as the mean-centered cosine similarity, such score is then summed with all the possible nodes sharing a link to measure the graph feature similarity

$$\bar{X} = \frac{1}{|V|} \sum_{v \in V} x_v \tag{2.13}$$

$$\bar{x}_u = x_u - \bar{X} \tag{2.14}$$

$$k(\bar{x}_u, \bar{x}_v) = \frac{\bar{x}_u^T \bar{x}_v}{||\bar{x}_u||_2 ||\bar{x}_v||_2} \tag{2.15}$$

$$K = \frac{1}{|\mathcal{E}|} \sum_{(u,v) \in \mathcal{E}} k(\bar{x}_u, \bar{x}_v) \tag{2.16}$$

In the research a setup (a graph) is defined as homophilic if exist a real value M which is limited by the infimum of the negative set of links scores $\tilde{K}_{neg}$ and upper-bounded by supremum of the positive set of links scores $\tilde{K}_{pos}$. The heterophilic case is defined instead as the exact opposite of the homphilic case, whereas in the gated setup, there must exist two real values M_1, M_2 such that M_2 is the lower bound for the positive supremum and the negative infimum, M_1 instead represent the upperbound.

$$\text{Heterophilic} : \exists M \in \mathbb{R} \mid \quad \sup(\tilde{K}_{pos}) \leq M < \inf(\tilde{K}_{neg}) \tag{2.17}$$

$$\text{Homophilic} : \exists M \in \mathbb{R} \mid \quad \sup(\tilde{K}_{neg}) < M \leq \inf(\tilde{K}_{pos}) \tag{2.18}$$

$$\text{Gated} : \exists M_1, M_2 \in \mathbb{R} \mid \quad M_1 < \inf(\tilde{K}_{pos}) \leq \sup(\tilde{K}_{pos}) \leq M_2 \tag{2.19}$$

The choice of the link decoder is also affected by the homophilic rate of the graph itself, the higher the rate, the simpler the model architecture required, such as choosing a dot-product operation. The study also shows how heuristic-based methods, such as **CN** (1.1) and **PPR** 1.1.2, performed at their peak in the homophilic setup. Furthermore, most tested link encoders performed best in the homophilic setup, followed by the heterophilic ones. The worst performance detected was at the gated setup, i.e., between the two configurations above, demonstrating a U-shaped trend.

2.8 LPFormers: An Adaptive Graph Transformer for Link Prediction

The study presents a new model for the LP task, underlying its ability to extract pairwise relations between nodes. According to study [27], MPNNs predict the likelihood of a link just by combining the node representations without including pairwise information of such nodes. LPFormers is a subgraph-based model 1.4.2; the extraction of such set around the target nodes u, v was computed via PPR, in which each nontarget node n is assigned with a score $\Gamma(u, v, n)$ correlated to the number of walks connecting n to the target nodes u, v. Formally:

$$\begin{aligned}
\Gamma(u, v, n) &= ppr(n, u) + ppr(n, v) && (2.20)\\
&= \sum_{k=0}^{\infty} \gamma^k (r_u^k(n) + r_v^k(n)) && (2.21)\\
&= \sum_{k=0}^{\infty} \gamma^k r_{uv}^k(n) && (2.22)
\end{aligned}$$

In the above equation, $\gamma = \alpha \cdot (1-\alpha)^k$ represents the dampening factor, i.e., the importance assigned to the longer walk. The hyper-parameter α is the teleportation probability of getting back to the source node. The function r_{uv}^k is a summation between the likelihoods of walking from u to n and v to n. They parametrized $\Gamma(\cdot)$ using three different MLPs, one when the nontarget node n is a common neighbor of u, v; one when n is either inside the one-hop subgraph of u or v and the last MLP otherwise, i.e., n is further than 1-hop from both target nodes.

$$\begin{aligned}
\mathrm{rpe}_{(u,v,n)} &= \mathrm{MLP}(ppr(u, n), ppr(v, n)) && (2.23)\\
\bar{\mathrm{rpe}}_{(u,v,n)} &= \mathrm{rpe}_{(u,v,n)} + \mathrm{rpe}_{(v,u,n)} && (2.24)
\end{aligned}$$

The enclosing subgraph was extracted via the application of a threshold η^π which changed for each of the setups described above, i.e., $\pi \in \{CN, 1, > 1\}$:

$$\hat{\mathcal{N}}^{\pi}_{(u,v)} = \{n \in \mathcal{N}^{\pi} | ppr(u, n) > \eta^\pi, ppr(v, n) > \eta^\pi\}$$

An MPNN was implemented as a link encoder yielding the embedding matrix H, and an attention mechanism was also implemented via a learnable function ϕ:

$$\begin{aligned}
h(u, v, n) &= W \left[h_n || rpe_{(u,v,n)}\right] && (2.25)\\
\tilde{w}(u, v, n) &= a^T \mathrm{LeakyReLU}\left(W h_u || W h_v || W h_n || h(u, v, n)\right) && (2.26)\\
w(u, v, n) &= \frac{\exp\left(\tilde{w}(u, v, n)\right)}{\sum_{l \in V(u,v)} \exp(\tilde{w}(u, v, l))} && (2.27)\\
s(u, v) &= \sum_{n \in V} w(u, v, n) \cdot h(u, v, n) && (2.28)
\end{aligned}$$

The parameter $s(u, v)$ represents the pairwise encoding of the extracted subgraph given the target nodes u and v. On the other hand, $h(u, v, n)$ represents the embedding of the node n given its topological relations with the target nodes. Lastly, an MLP was fed with Hadamard product between the target nodes embeddings concatenated the pairwise encoding $s(\cdot)$ and the different Neighbors $\hat{\mathcal{N}}^{\pi}_{(u,v)}$ for each scenario described above.

$$p(u, v) = \sigma \left[MLP\left(h_u \cdot h_v || s(u, v) || \hat{\mathcal{N}}^{CN}_{(u,v)} || \hat{\mathcal{N}}^{1}_{(u,v)} || \hat{\mathcal{N}}^{>1}_{(u,v)}\right)\right]$$

2.9 Neural common neighbor with completion for link prediction

Structural features (SF), are commonly used in LP models; the order in which such features are integrated into the framework often conditions the quality of the embeddings themselves. According to the research [33], LP methods that revolve around the integration of some type of SF before the node-encoder tend to be more expressive, but this also forces the model to compute the node representations once for each target node pair. On the other hand, integrating SF posterior to the node encoding phase, or decoupled from the encoder, allows the method to be more efficient, i.e., nodes are embedded just one time, even though the resulting embeddings themselves usually lose expressiveness. In this study, a new MPNN-then-SF method was implemented; the graph nodes were embedded just once using an MPNN. A pooling operation onto the subset S related to the target nodes was applied to reduce the neighborhood matrix to a vector:

$$\text{Pool}(\{\text{MPNN}(u, A, X) | u \in S\}) \tag{2.29}$$

The final vector representation for the first setup NCN (neural common neighbor) was computed as the Hadamard product between the target nodes, i, j, embeddings concatenated to the contextual information gathered from the node in S.

$$\text{NCN}(i, j, A, X) = \text{MPNN}(i, A, X) \odot \text{MPNN}(j, A, X) || \sum_{u \in N(i) \cap N(j)} \text{MPNN}(u, A, X) \tag{2.30}$$

In the presence of target nodes not sharing any CN, the NCN resulting embedding would reduce to a simple multiplication. The second proposed model tampered with the issue by implementing a form of graph augmentation, not directly on the starting graph but on the extracted neighborhood. Given the target nodes i, j, the probability vector P_{uij} representing the likelihood for a node u to be a common neighbor of the target nodes were defined as:

$$P_{uij} = \begin{cases} 1 & \text{if } u \in N(i, A) \cap N(j, A) \\ \hat{A}_{iu} & \text{if } u \in N(j, A) - N(i, A) \\ \hat{A}_{ju} & \text{if } u \in N(i, A) - N(j, A) \\ 0 & \text{otherwise} \end{cases} \tag{2.31}$$

Thus, in the case of a common neighbor u, NCNC does not differ from NCN; on the other hand, if u belonged to either just i or j neighborhood, a link decoder fed with the NCN embedding would have yielded such probability. NCN prediction was then computed as follows:

$$\text{NCNC}(i, j, A, X) = \text{MPNN}(i, A, X) \odot \text{MPNN}(j, A, X) || \sum_{u \in N(i) \cup N(j)} P_{uij} \text{MPNN}(u, A, X) \tag{2.32}$$

2.10 Mixture of Link Predictors

The study [20] presented an ensemble approach for LP, utilizing different expert models to achieve better metrics. Both heuristic and GNN methods lack differentiation for the execution of the different target node pairs. As shown in this study, no expert models or heuristics outperform others across all groups. The study also shows that each model result has a low overlapping ratio among the others, suggesting that each expert predicts a specific set of links.

$$Y_{ij} = \sigma\left(\sum_{o=1}^{m} G(h_{ij})_o E_o(A, X)_{ij}\right) \tag{2.33}$$

The final prediction was computed as a linear combination between the expert models, each model output was weighted by a gating model G, composed by three MLPs. G was fed with heuristic features s_{ij}, of which CN (1.1), AA (1.3), RA (1.4) as local, and PPR 1.1.2, Shortest path and Katz index (1.5) as global one. The gating model also received the Hadamard product of the target nodes features $x_i \cdot x_j$; both heuristic and node features were fed to two different MLPs concatenated and processed by one final MP, yielding the expert weights.

$$G(x_{ij}, s_{ij}) = \operatorname{softmax}\left(f_3(f_1(x_{ij})||f_2(s_{ij}))\right) \tag{2.34}$$

Each expert model was trained individually, then freezed to adjust the gating model weights with binary cross entropy as loss function.

2.11 Pure Message Passing Can Estimate Common Neighbor for Link Prediction

The research [6] aimed to study the prowess of message-passing mechanisms for gathering link structural features, such as approximated common neighbor cardinality. Such computation could result in a complex task; according to the study, MPNNs are not capable of gathering structural information such as CN. In the study, they sampled from an F-dimensional hypercube with unit vector norms a random vector $h_v^{(0)}$ for each node k. The sampled embedding was multiplied with the concatenation of the node encoder output and the degree of the nodes itself after the application of an MLP $f : \mathbb{R}^{F_x+1} \rightarrow \mathbb{R}$.

$$\tilde{h}_v^0 = f(\text{GNN}(v)||[d_v]) \cdot h_v^{(0)} \tag{2.35}$$

$$\tilde{h}_v^l = \sum_{u \in \mathcal{N}(v)} \tilde{h}_u^{l-1} \quad \forall l > 0 \tag{2.36}$$

The node encoder implemented the message-passing mechanism as a simple sum of the neighbors, with the goal of preserving the structural similarity between nodes. To conclude, an MLP computed the link likelihood with the Hadamard product of the target nodes concatenated with the Distance Encoding embedding.

$$\#(p, q) = \mathbb{E}(\tilde{h}_u^p \cdot \tilde{h}_v^p) \tag{2.37}$$

$$h_{(u,v)} = (\text{GNN}(u) \odot \text{GNN}(v)) \; || \; [\#(1, 1), \dots, \#(r, r)] \tag{2.38}$$

The parameter r represents the maximum number of hops, which in this research was set to two, whereas the function $\#(p, q)$ denotes a proximity-aware aggregation metric based on shortest path distance.

2.12 Revisiting link prediction: A data perspective

The study [21] highlighted the incompatibility between structural features, i.e. low and high order heuristics, and node features. Pair of Nodes with elevated level of structural features often indicates an higher likelihood of them being connected. On the other hand similarity between two nodes, i.e. nodes features, also indicates an higher probability of link presence. The issue revolves around pair of nodes which do not share enough structural information but they have similar node features themselves. Opposite scenario is when the structural proximity are meaningful but the node features themselves are not highly indicative for link presence. In this research they presented a new setup for different models, by decoupling structural and node features using two different link encoders-decoders. The resulting output for both GNNs were fed to two different Readout module which yeilded the probabilities. Then, such likelihoods were weighted with a learnable function, MLP, returning a coefficient $\alpha \in (0, 1)$.

$$H^{feat} = \text{GNN}(X_{feat}, A) \tag{2.39}$$

$$H^{drnl} = \text{GNN}(X_{drnl}, A) \tag{2.40}$$

$$\tilde{p}_{feat}(e_{uv}) = \textbf{Readout1}(H_u^{feat}, H_v^{feat}) \tag{2.41}$$

$$\tilde{p}_{drnl}(e_{uv}) = \textbf{Readout2}(H_u^{drnl}, H_v^{drnl}) \tag{2.42}$$

$$\tilde{p}(e_{uv}) = \alpha \tilde{p}_{feat}(e_{uv}) + (1 - \alpha)\tilde{p}_{drnl}(e_{uv}) \tag{2.43}$$

In which the structural features representation are generated applying **DRNL** 1.4.4 labeling system.

2.13 Preserving node similarity adversarial learning graph representation with graph neural network

The research [39] presented an adversarial setup PNS-AGNN, in which both the generator and the discriminator were implemented as GNNs. They first generate the distribution matrix R starting from the normalized RA (1.4) for each pair of nodes in the graph; the parameter R represents the neighbor similarity distribution matrix.

$$\bar{S}_{uv} = \begin{cases} \sum_{z \in N(u) \cap N(v)} \frac{1}{d(z)} & \text{if } u \neq v, \\ 0 & \text{otherwise} \end{cases} \tag{2.44}$$

$$M_{uv} = \begin{cases} 0 & \text{if } u \neq v, \\ \sum_{v'} \bar{S}_{uv} & \text{otherwise} \end{cases} \tag{2.45}$$

$$R = M^{-1}\bar{S} \tag{2.46}$$

The generator received the BFS tree around the target node u and generated the negative pairs of nodes. Such pairs were generated with the goal of following the same distribution as positive ones, creating, in fact, the so-called hard negatives. The negative pairs were sampled from nonneighboring nodes, i.e., further than one hop. On the other hand, for each target node, u, the discriminator was fed with the set of nodes that share common neighbors with u, $\{v|R_{uv} > 0\}$. Finally, the discriminator GNN aggregated the positive and negative nodes sampled to generate the final embedding.

$$\begin{aligned} \min_{g} \max_{\mathbf{W}} O(G, D) = \sum_{u=1}^{|\mathcal{P}|} \Big(& \mathbb{E}_{\nu \sim \mathrm{PR}(\cdot|u)} \big[\log \sigma(\mathbf{H}_v^T \mathbf{H}_u; \mathbf{W}) \big] \\ & + \mathbb{E}_{\nu \sim Q(\cdot|u;g)} \big[\log(1 - \sigma(\mathbf{H}_v^T \mathbf{H}_u; \mathbf{W})) \big] \Big). \end{aligned} \tag{2.47}$$

Where the parameter g represents the generator, whereas $\mathbf{W}$ the discriminator, the distribution $\mathrm{PR}(\cdot|u)$ denotes the neighbor similarity distribution and $G(\cdot|y;g)$ represents the negative neighbor distribution generated by g.

Chapter 3

Proposed Model

Inspired by the versatility of **SEAL** [42] we utilize a subgraph-based setup, the link encoder is implemented as a stack of **Transformer Encoders** [30] whereas the link decoder is a learnable function ϕ. The encoder receives the extracted nodes features together with the precomputed distance embeddings and the labeling trick vector, and generate the encoded representation of the nodes $\mathrm{H}^{(N)}$. The learnable function ϕ is fed with the aggregated representation of the target nodes $\tilde{h}_{uv}$ and return the link likelihood $p(e_{uv})$.

3.1 Pre-processing

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the starting graph, the **h-hop** enclosing subgraph around the target nodes is computed by extracting all the nodes with at most distance h from either u or v:

$$\mathscr{S}^h_{uv} = \{n \in \mathcal{V} \mid d(n,u) \leq h \vee d(n,v) \leq h\} \tag{3.1}$$

$$l = |\mathscr{S}^h_{uv}| \tag{3.2}$$

For each extracted node $n \in \mathscr{S}^h_{uv}$ its distance-based positional encoding is defined as the vector $d_n \in [0,\infty]^l$ of shortest-path-length between n and all the remaining nodes in $\mathscr{S}^h_{uv}$.

$$D = \begin{bmatrix} 0 & d(n_1,n_2) & d(n_1,n_3) & \cdots & d(n_1,n_l) \\ d(n_2,n_1) & 0 & d(n_2,n_3) & \cdots & d(n_2,n_l) \\ d(n_3,n_1) & d(n_3,n_2) & 0 & \cdots & d(n_3,n_l) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d(n_l,n_2) & d(n_l,n_2) & d(n_l,n_3) & \cdots & 0 \end{bmatrix} \in [0,\infty]^{l \times l} \tag{3.3}$$

If a path between two nodes does not exist the related cell is set to ∞, the final positional encoding are computed as the element-wise inverse of the matrix D, in other word for each distance vector d_n a new encoding is defined as: $\tilde{d}_n = 1/(d_n+1)$. Given the domain of the matrix D its edge value are treated accordingly, by setting directly to one the diagonal values, while zero-ing the cell containing ∞.

$$\tilde{D} = (D+1)^{\circ -1} \in (0,1]^{l \times l} \tag{3.4}$$

The labeling-trick 1.4.4 vector $t \in [0,1]^l$ is generated once for the subgraph $\mathscr{S}^h_{uv}$, each entry of t represents the label assigned to a node $n \in \mathscr{S}^h_{uv}$. In this research two similar approaches were tested, as mentioned in [32] a simple zero-one labeling is employed as first choice for the labeling function $g_1^{(u,v)} : \mathscr{S}_{uv}^h \to \{0,1\}$, whereas as second choice a distance based labeling function $g_2^{(u,v)} : \mathscr{S}_{uv}^h \to [0,1]$. The function $g_2^{(u,v)}$ assign labels to non target nodes as the average between the inverse of their distance given the target nodes.

$$g_1^{(u,v)}(n) = \begin{cases} 1 & \text{if } n \in \{u,v\} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

$$g_2^{(u,v)}(n) = \begin{cases} 1 & \text{if } n \in \{u,v\} \\ \frac{1}{2} \cdot (\tilde{d}_{n,u} + \tilde{d}_{n,v}) & \text{otherwise} \end{cases} \tag{3.6}$$

The final labeling vector can then be computed using one of the two proposed functions $g_1^{(u,v)}, g_2^{(u,v)}$.

$$t = \begin{bmatrix} g_s^{(u,v)}(n_1) \\ g_s^{(u,v)}(n_2) \\ \vdots \\ g_s^{(u,v)}(n_l) \end{bmatrix} \in [0,1]^{l\times 1} \quad \forall s = 1,2 \tag{3.7}$$

3.2 Model Architecture

The model is composed of three main blocks, the backbone of the net containing two linear layers $W_{feat} \in \mathbb{R}^{p_1 \times g}, W_{pos} \in \mathbb{R}^{p_2 \times l}$ and the related biases $b_{feat} \in \mathbb{R}^{p_1}, b_{pos} \in \mathbb{R}^{p_2}$.

The node encoder is implemented with a stack of N Transformer-encoder layers $\textbf{tr-block}_i : \mathbb{R}^k \to \mathbb{R}^k \ \forall i = 1, \ldots, N$. The link decoder is an MLP $\phi : \mathbb{R}^k \to (0,1)$ where $k = p_1 + p_2 + 1$, such model can either be just a sequence of linear layer, or it can be enriched by other modules as layer normalization and/or dropout.

3.2.1 Forward flow

Given the matrix $\tilde{D}$ and the node features matrix $X \in \mathbb{R}^{l\times g}$, the backbone feeds X to the linear layer (W_{feat}, b_{feat}) while $\tilde{D}$ is fed to (W_{pos}, b_{pos}). For sake of simplicity we assume the bias vectors b_{feat}, b_{pos} are automatically broadcasted from $\mathbb{R}^{p_1}, \mathbb{R}^{p_2}$ to $\mathbb{R}^{l\times p_1}, \mathbb{R}^{l\times p_2}$.

$$\text{pos}_{out} \leftarrow \tilde{D} W_{pos}^T + b_{pos} \in \mathbb{R}^{l\times p_1} \tag{3.8}$$

$$\text{feat}_{out} \leftarrow X W_{feat}^T + b_{feat} \in \mathbb{R}^{l\times p_2} \tag{3.9}$$

The resulting embedding pos_{out} and feat_{out} are concatenated together with the labeling trick tensor t and then fed to the transformers stack.

$$\mathrm{H}^{(0)} \leftarrow \text{concat}\{\text{feat}_{out} || \text{pos}_{out} || t\}\big|_2 \in \mathbb{R}^{l\times k} \tag{3.10}$$

$$\mathrm{H}^{(i)} \leftarrow \textbf{tr-block}_i(\mathrm{H}^{(i-1)}) \in \mathbb{R}^{l\times k} \quad \forall i = 1, \ldots, N \tag{3.11}$$

Where the subscript $|_2$ refers to the dimension on which the concatenation operation is cast upon. From the transformer output $\mathrm{H}^{(N)}$ the target nodes embeddings

$h_u^{(N)}, h_v^{(N)}$ are extracted and aggregated with a tunable function $f_{\text{head}} : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}^k$. In this experiment f_{head} is constrained to the set $\{f^{\oplus}, f^{\odot}\ f^{\text{avg}}\}$ representing respectively: sum, element-wise multiplication and average. The link decoder ϕ receives the aggregated representation of the target nodes embedding and yields the likelihood of the link e_{uv}.

$$\tilde{h}_{uv} \leftarrow f_{\text{head}}(h_u^{(N)}, h_v^{(N)}) \in \mathbb{R}^k \qquad \forall f_{\text{head}} \in \{f^{\oplus}, f^{\odot}, f^{\text{avg}}\} \tag{3.12}$$

$$p(e_{uv}) \leftarrow \phi(\tilde{h}_{uv}) \in (0, 1) \tag{3.13}$$

The model loss is computed as the binary cross entropy between the predicted link likelihood $p(e)$ and the discrete ground truth label y_e, $\forall e \in \mathcal{E}^+ \cup \mathcal{E}^-$.

$$\mathscr{L} = \sum_{e \in \mathcal{E}^+ \cup \mathcal{E}^-} -y_e \log(p(e)) - (1 - y_e) \log(1 - p(e)) \tag{3.14}$$

Where the sets $\mathcal{E}^+, \mathcal{E}^-$ represent the positive and negative set of links to predict. The model can also include additional positional encodings which are integrated with the actual distance-based positional encoding $\tilde{D}$ using an aggregator function $f_{\text{tail}} \in \{f^{\oplus}, f^{\odot}, f^{\text{avg}}\}$. This operation is performed before the application of the backbone linear layer (W_{pos}, b_{pos}).

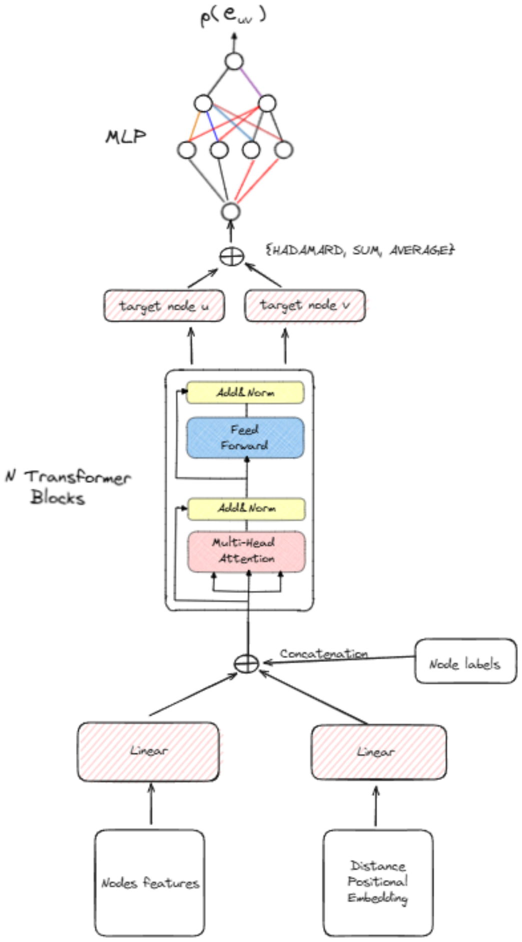


Figure 3.1: Image representing the full architecture of the proposed model.

Chapter 4

Experiments

In this work, the experiments were conducted with three widely used citation network datasets: **CORA**, **CiteSeer** and **PubMed**. These datasets consist of academic papers as nodes and citation links as edges, forming directed graphs. CORA (10'556 edges, 2'708 nodes) contains machine learning papers categorized into seven classes, CiteSeer (9'104 edges, 3'327 nodes) contains research papers categorized into six areas, and PubMed (88'648 edges, 19'717 nodes) contains biomedical papers classified by disease type. Each node is represented by a bag-of-words feature vector derived from the paper's content. If an extracted sub-graph size was higher than the chosen threshold, the exceeding nodes were randomly chosen and removed from the sub-graph. We applied a random 70-10-20 splits (train-val-test), and the negative links were sampled randomly from the processed graph. The model generated the right set of hyper-parameters for each dataset using the given validation set. The validation session was executed with early stopping on the loss value for seven epochs. The number of epochs was incremented to twenty-five with early stopping on the test set loss. **OneCycleLR** [28] was used as a scheduler together with **AdamW** [19] as optimizer; lastly, the decoder activation was fixed to the sigmoid function. Below is the list of hyper-parameters:

- The hop $h \in [1, 2, 3]$
- The max subgraph size $l \in \{200, 350, 500\}$
- The labeling trick function $g \in \{g_1, g_2\}$
- The backbone positional encoding out dimension $p_1 \in \{99, 199, 299\}$
- The backbone feats embedding out dimension $p_2 \in \{600, 700, 800\}$
- The aggregator function $f_{head} \in \{f^{\oplus}, f^{\odot}, f^{\text{avg}}\}$
- The number of transformer encoder layers $N \in [1, 2, 3]$, number of head n-head $\in \{2, 4, 10\}$, dropout presence and activation function $s \in \{\text{ReLU}, \text{GeLU}\}$. The hidden size of the feed forward dimensionality dim-ff $\in \{1000, 1400, 1600\}$.
- The depth of the MLP decoder num-layer-decoder $\in [1, 2, 3, 4]$, and the presence or absence of dropout and layer-norm layers.
- The parameter for the OneCycleLR:
    - Maximum learning rate max-lr $\in [8e^{-10}, 1e^{-2}]$

  - Percentage of cycle for increasing the learning rate max-pct $\in [0.05, 0.3]$
- The weight decay wgt-decay $\in [1e^{-6}, 1e^{-2}]$ for the optimizer

4.1 Evaluation Protocols

The model performances against the competitors were measured using the following metrics:

- Accuracy
- F1-score
- ROC AUC
- HITS@K with $K \in \{1, 5, 10, 20, 50, 100\}$

The **HITS@K** metric was computed from the test-set positive samples. Five hundred negative links were randomly sampled for each positive link using the target nodes as part of the negative pairs.

4.2 Competitors

We compare our methods with several state-of-the-art models, ranging from adversarial learning techniques and topological-based methods to traditional link prediction algorithms. Specifically, we compare with:

- **Mixture of Link Predictors** [20] ensemble approach combining different state-of-the-art LP models by weighting their prediction with an MLP fed with a set of heuristics extracted from the subgraph around the target nodes.
- **Neural Common Neighbor with Completion for Link Prediction** [33] MPNN-then-SF technique implementing the scoring model to augment the number of considered embedding after the MPNN embedding process.
- **A Topological Perspective on Demystifying GNN-Based Link Prediction Performance** [34] leverages the topological information of the graph through the implementation of a new heuristic to improve the message-passing mechanism.
- **Linkless Link Prediction via Relational Distillation** [9] distillates the knowledge of a GNN teacher model to an MLP architecture with different techniques.
- **Learning from Counterfactual Links for Link Prediction** [44] propose a graph augmentation mechanism by implementing a GNN generating both negative and positive links to aggregate more contextual information.
- **Preserving Node Similarity: Adversarial Learning Graph Representation with Graph Neural Network** [39] analyzing the effect of an adversarial training set up by also leveraging the graph structural information via Resource Allocation heuristic.

- **Revisiting Link Prediction: A data perspective** [21] is a subgraph-based method that decouples the structural information from the node features to compute the predictions. The resulting likelihoods are weighted with an additional MLP yielding the weight.
- **Link Prediction with Persistent Homology** [38] implements a GNN-then-SF technique where the structural global features are aggregated from the graph set of simplices are concatenated to the GCN nodes representation.

4.3 Reproducibility

The code is available at: `https://github.com/skippa1da2flippa/last_1629.git`

Chapter 5

Results

5.1 Competitors comparison

5.1.1 Cora

Table 5.1: Performance Comparison (ROC AUC, Accuracy, F1) in Cora

Model	**ROC AUC**	**Accuracy**	**F1**
Counterfactual	93.05	NA	NA
Linkless	95.23	NA	NA
PSNGNN	NA	89.3	88.9
Our approach	94.	87.2	87.2

Table 5.2: Performance Comparison (Hits@K) in Cora

Model	**Hits@100**	**Hits@50**	**Hits@20**	**Hits@10**	**Hits@1**
Mix of LP	96.26	NA	NA	75.84	32.12
NCNC	89.05	81.36	67.1	53.78	10.9
Topological	87.95	NA	NA	NA	NA
Counterfactual	NA	75.49	65.57	NA	NA
Linkless	NA	78.82	27.87	NA	NA
Revisiting	NA	NA	NA	75.07	NA
Our approach	88.34	79.43	65.88	53.84	20.9

As shown in the first table 5.1, the proposed method is comparable to the presented state-of-the-art, such outcome is not demonstrated also in the hits table 5.2, which shows a clear gap from the **Mixtures of Link predictors** [20], in the HITS@K $\forall K \in \{1, 10, 100\}$. This behavior can be explained by the presence of multiple expert models, each gathering different aspects of the received graph, making the ensemble more precise. The proposed model shows a slight improvement in the remaining hits HITS@K $\forall K \in \{20, 50\}$, in the same fashion for the metric **ROC AUC**.

5.1.2 Citeseer

Table 5.3: Performance Comparison (ROC AUC, Accuracy, F1) in Citeseer

Model	**ROC AUC**	**Accuracy**	**F1**
Counterfactual	92.12	NA	NA
Linkless	95.32	NA	NA
PSNGNN	NA	91.9	91.8
Our approach	95.3	89.3	89.3

Table 5.4: Performance Comparison (Hits@K) in Citeseer

Model	**Hits@100**	**Hits@50**	**Hits@20**	**Hits@10**	**Hits@1**
Mix of LP	96.44	NA	NA	82.77	58.50
NCNC	93.13	88.6	79.05	69.59	32.45
Topological	91.86	NA	NA	NA	NA
Counterfactual	NA	77.01	68.09	NA	NA
Linkless	NA	NA	77.32	34.75	NA
Revisiting	NA	NA	NA	82.64	NA
Our approach	91.76	84.4	72.09	62.09	28.68

The above tables 5.3, 5.4 show an aligned behavior with the result in Cora dataset, demonstrating a mild improvement in the metric **ROC AUC**, while being surclassed in the remnant hits by Mixtures of Link Predictors and Neural Common Neighbor (NCNC). The **Accuracy** and **F1** metrics in both datasets show a slight distance from the actual state-of-the-art **Preserving node similarity adversarial learning graph representation with graph neural network**. Such behavior is probably due to the GAN's ability to discern hard negative pairs gained through the adversarial training process, which integrates negative pairs with richer structural interaction.

5.1.3 PubMed

The result shown in the above table 5.6 aligns with the previous analysis, showing a mild improvement with respect to **NCNC**; this can either be caused by the less amount of data employed in the metric calculation or due to the proposed method ability to generalize better wider subgraph thanks to the attention mechanism.

Table 5.5: Performance Comparison (ROC AUC, Accuracy, F1) in PubMed

Model	**ROC AUC**	**Accuracy**	**F1**
Counterfactual	97.53	NA	NA
Linkless	97.9	NA	NA
Homology	97.03	NA	NA
Our approach	94.9	88.3	88.3

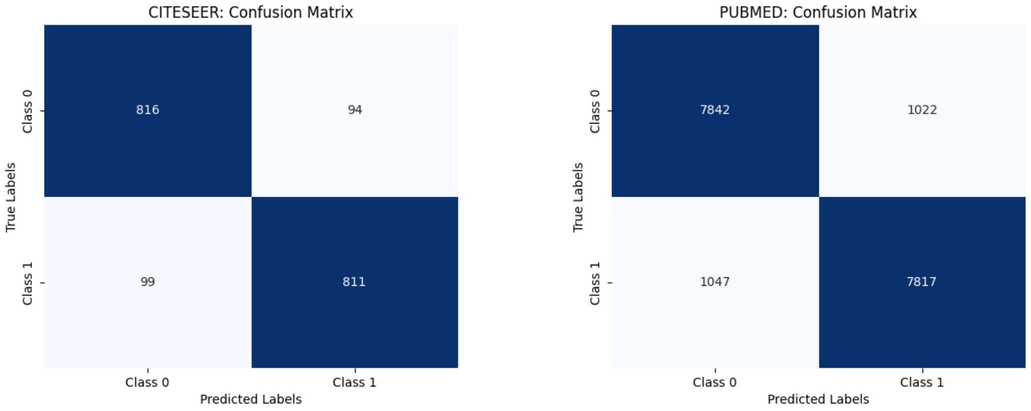
Table 5.6: Performance Comparison (Hits@K) in PubMed

Model	**Hits@100**	**Hits@50**	**Hits@20**	**Hits@10**	**Hits@1**
Mix of LP	90.38	NA	NA	61.11	45.13
NCNC	81.8	69.25	51.42	34.61	8.57
Topological	76.51	NA	NA	NA	NA
Counterfactual	NA	62.80	44.9	NA	NA
Linkless	NA	57.33	53.48	NA	NA
Revisiting	NA	NA	NA	61.89	NA
Our approach	88.	69.3	52.6	40.8	10.5

Note: Only 20% of the data was used for computing hits@k.

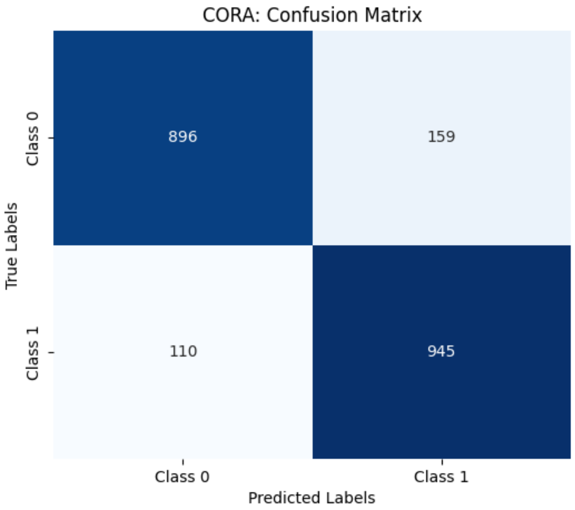
5.2 Confusion matrices

In the result below 5.2, the **Class 0** represents the negative samples, and **Class 1** represents the positive respectively. The different confusion matrices show a uniform behavior in the error rate distribution between the employed datasets. In the dataset **Citeseer** and **PubMed**, negative links are mildly less likely to be misclassified as positive, whereas in **Cora**, positive link tends to be misclassified more since there is a mild pattern in the data this is probably due to the applied random split.



(a) Plot 1: Citeseer confusion matrix

(b) Plot 2: PubMed confusion matrix



(c) Plot 3: Cora confusion matrix

Figure 5.1: Dataset comparison: confusion matrices

5.3 Positional Encoding and Labeling trick analysis

Three additional versions of the proposed model were implemented to analyze the effectiveness of the distance-based positional encoding and the labeling trick vectors. The first and the second variant replaced the backbone linear layer (W_{pos}, b_{pos}) with a simple embedding matrix, similar to the approach used in **BERT** [5]. These two new configurations differ from each other in terms of how the positional embedding was generated. The first model generated the local positional encoding from an embedding matrix E_1 of size (max-nodes $\times\, p_1$), where max-nodes, represent the max number of tokens, i.e., nodes, which Transformer-Encoder can take in one time for each link. In the second setup, the embedding matrix E_2 was instead of size (num-nodes-graph $\times\, p_1$) where num-nodes-graph refers to the total number of nodes in the graph. Thus, in the second configuration, the matrix E_2 learned an embedding for each node in the graph. In contrast, in the first setup, the embedding matrix E_1 refined the positional vectors for the limited window of nodes. The third variant instead retained the original architecture but excluded the labeling trick vector t from the model's input.

Table 5.7: Performance Comparison (ROC AUC, Accuracy, F1) in Cora between different configurations

Model	**ROC AUC**	**Accuracy**	**F1**
local emb	73.1	67.5	67.3
global emb	74.2	67.9	67.2
no trick	92	85.5	85.5
base	94	87.3	87.3

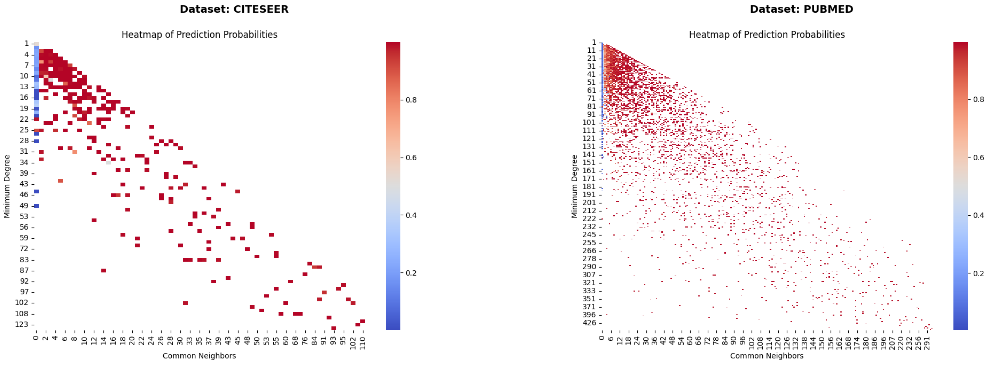
Table 5.8: Performance Comparison (ROC AUC, Accuracy, F1) in Citeseer between different configurations

Model	**ROC AUC**	**Accuracy**	**F1**
local emb	71	66	65.8
global emb	71.7	65.8	65.8
no trick	88.7	79.8	79.6
base	95.3	89.3	89.3

The results in the above tables 5.7, 5.8 show the strong impact distance-based positional encoding has on the model performance. On the other hand, although the improvement in the labeling trick is lower, with respect to the positional representation, it still has relevance in the model performance. Interestingly the Cora model is less affected by the absence of the labeling trick.

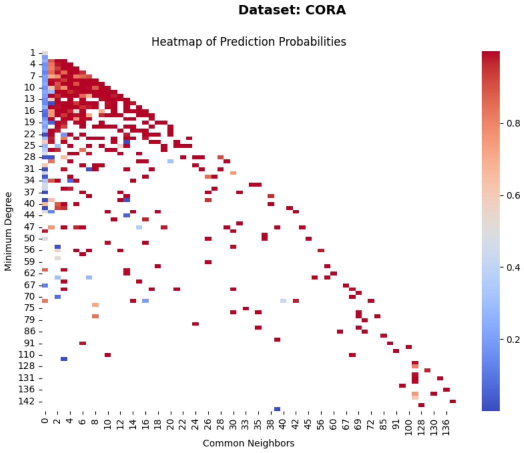
5.4 Dataset results analysis

The plot below 5.4 shows the relation between the min degree of the target pairs and their common neighbor cardinality; the different gradation of colors represents the likelihood assigned to any given point by the model. The plot shows that as the min degree rises and the common neighbor remains, the likelihood remains low; on the other hand, when both node degree and common neighbor are high, the models tend to assign larger probability.
In the tables 5.4 and 5.3, the dotted horizontal line represents the best decision boundary given the predictions assigned by the model; the different color defines the real labels instead. In both plots **PubMed** dataset tends to have the majority of negative samples grouped under the threshold $t = 0.2$, which is in line with the higher homophily rate reported in the paper [47], which assigns to the PubMed dataset the score 0.22 and 0.05 and 0.12 to Citeseer and Cora respectively. A higher homophily rate implies that nodes sharing similar features are more likely to connect; thus, when a negative pair of nodes is randomly generated, the structural relation is often weak, leading to a lower decision boundary. On the other hand, the other two datasets show a more balanced threshold, indicating a higher level of uncertainty, which is due to the dataset's tendency to be more likely to have two different nodes connected.



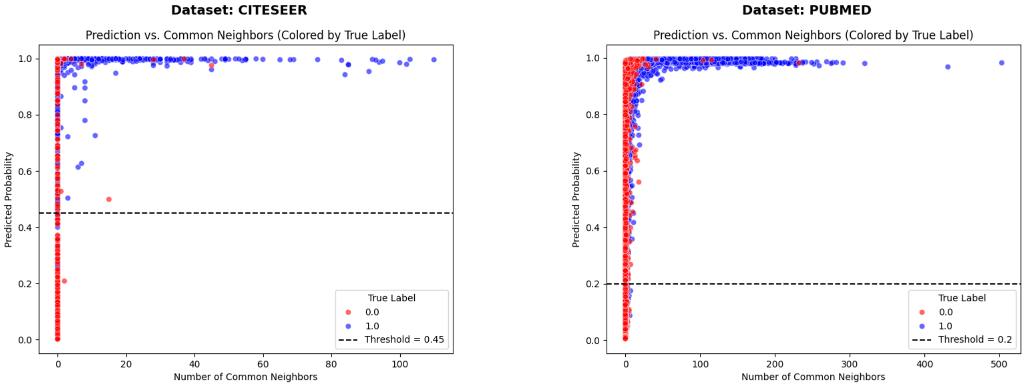
(a) Plot 1: Citeseer heatmap

(b) Plot 2: PubMed heatmap



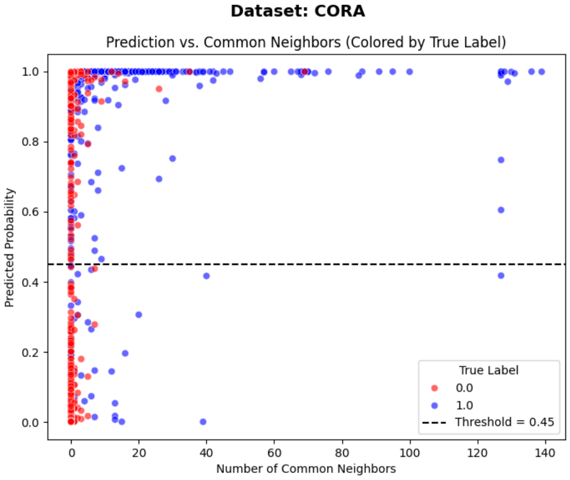
(c) Plot 3: Cora heatmap

Figure 5.2: Dataset comparison: probability heatmaps given the common neighbors metric and min node degree



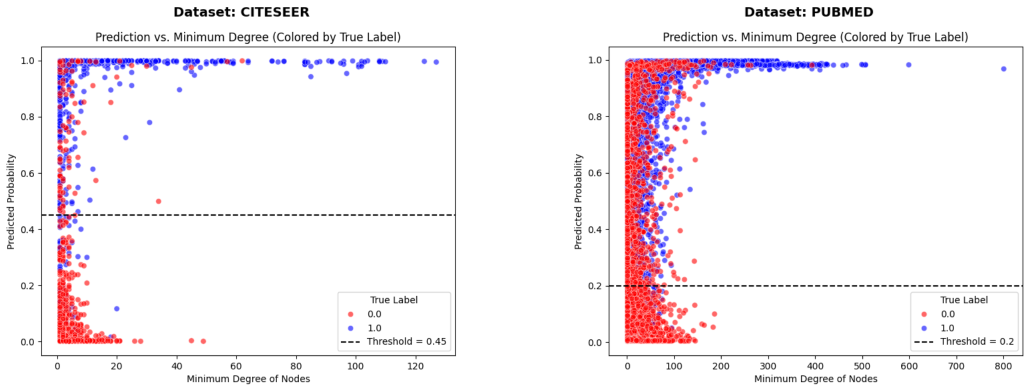
(a) Plot 1: Citeseer prob. vs CN

(b) Plot 2: PubMed prob. vs CN



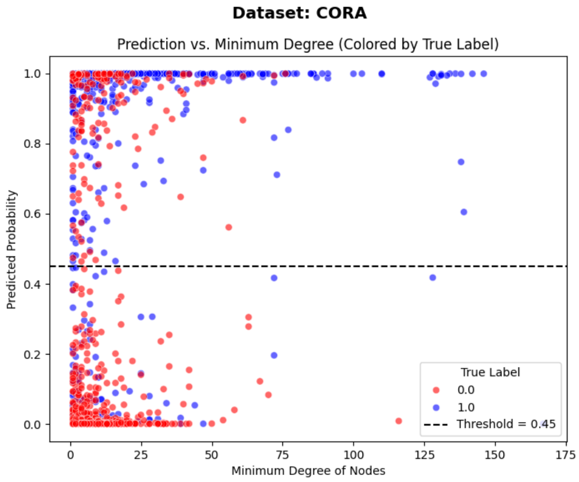
(c) Plot 3: Cora prob. vs CN

Figure 5.3: Dataset comparison: relationships between CN and likelihood



(a) Plot 1: Citeseer prob. vs deg

(b) Plot 2: PubMed prob. vs deg



(c) Plot 3: Cora prob. vs deg

Figure 5.4: Dataset comparison: relationships between degree and likelihood

Chapter 6

Conclusions

The research presented an attention-based link prediction algorithm, leveraging the expressivity of the Transformer block encoder [30] while introducing structural information via the implementation of distance-based encoding. Such structure is enriched with an interchangeable labeling trick technique [43], which aimed, together with the distance encoding system, to enhance the expressive power of the Transformer architecture for graph-based computation. The research shows that, for small subgraphs, plain attention mechanism can be an alternative to the normal message-passing (MP) paradigm even though its computational complexity is more significant. Furthermore, the positional embedding analysis described in 5.3 aligns with the research [25] underlying the importance of structural features inclusion for an improved spatial understanding in attention-based models. The developed labeling trick system has a marginal role with respect to positional embedding, which still results in making the proposed model comparable with the present techniques.

On the other hand, the presented method still lacks the ability to completely discern the hard negative from the positive samples, which can be fixed by introducing a hard negative pair, i.e., a pair enclosed in a rich structural context, during the training process. Moreover, the model could benefit from the inclusion of additional structural features via different heuristics, as seen in [20]. Such inclusion can be performed by computing the subgraph extraction with RPR 1.1.2 as in [27]. By inducing the subgraph with a random-walk-based algorithm, the latent long-range dependencies, which are not captured by the distance-based encoding due to its short range of action given the hop number h, could be gathered.

Bibliography

[1] Khushnood Abbas, Alireza Abbasi, Shi Dong, Ling Niu, Laihang Yu, Bolun Chen, Shi-Min Cai, and Qambar Hasan. Application of network link prediction in drug discovery. *BMC bioinformatics*, 22:1–21, 2021.

[2] Adar Adamic. Adamic la, adar e. *Friends and neighbors on the web, Social Networks*, 25 (3):211–230, 2003.

[3] Dzmitry Bahdanau. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[5] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Kaiwen Dong, Zhichun Guo, and Nitesh V. Chawla. Pure message passing can estimate common neighbor for link prediction. *arXiv preprint arXiv:2309.00976*, 2023. doi: 10.48550/arXiv.2309.00976.

[7] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017. doi: 10.48550/arXiv.1704.01212.

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

[9] Zhichun Guo, William Shiao, Shichang Zhang, Yozen Liu, Nitesh V Chawla, Neil Shah, and Tong Zhao. Linkless link prediction via relational distillation. In *International Conference on Machine Learning*, pages 12012–12033. PMLR, 2023.

[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[11] Md Shamim Hussain, Mohammed J Zaki, and Dharmashankar Subramanian. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 655–665, 2022.

[12] Paul Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50, 1912. doi: 10.1111/j.1469-8137.1912.tb05611.x.

[13] Leo Katz. A new index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953. doi: 10.1007/BF02289026.

[14] Junghun Kim, Ka Hyun Park, Hoyoung Yoon, and U Kang. Pull: Pu-learning-based accurate link prediction. *arXiv preprint arXiv:2405.11911*, 2024.

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

[16] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[18] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007. doi: 10.1002/asi.20591.

[19] I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[20] Li Ma, Haoyu Han, Juanhui Li, Harry Shomer, Hui Liu, Xiaofeng Gao, and Jiliang Tang. Mixture of link predictors on graphs. *arXiv preprint arXiv:2402.08583*, 2024. doi: 10.48550/arXiv.2402.08583.

[21] Haitao Mao, Juanhui Li, Harry Shomer, Bingheng Li, Wenqi Fan, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. Revisiting link prediction: A data perspective. *arXiv preprint arXiv:2310.00793*, 2023.

[22] Hebatallah A Mohamed, Diego Pilutti, Stuart James, Alessio Del Bue, Marcello Pelillo, and Sebastiano Vascon. Locality-aware subgraphs for inductive link prediction in knowledge graphs. *Pattern Recognition Letters*, 167:90–97, 2023.

[23] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14, 2001.

[24] Wonpyo Park, Woonggi Chang, Donggeon Lee, Juntae Kim, and Seung-won Hwang. Grpe: Relative positional encoding for graph transformer. *arXiv preprint arXiv:2201.12787*, 2022.

[25] Ahsan Shehzad, Feng Xia, Shagufta Abid, Ciyuan Peng, Shuo Yu, Dongyu Zhang, and Karin Verspoor. Graph transformers: A survey. *arXiv preprint arXiv:2407.09777*, 2024.

[26] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[27] Harry Shomer, Yao Ma, Haitao Mao, Juanhui Li, Bo Wu, and Jiliang Tang. Lpformer: An adaptive graph transformer for link prediction. *arXiv preprint arXiv:2310.11009*, 2023. doi: 10.48550/arXiv.2310.11009.

[28] Leslie N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[29] Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1): D607–D613, 2019.

[30] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. doi: 10.48550/arXiv.1710.10903.

[32] Xiyuan Wang, Pan Li, and Muhan Zhang. Improving graph neural networks on multi-node tasks with labeling tricks. *arXiv preprint arXiv:2304.10074*, 2023.

[33] Xiyuan Wang, Haotong Yang, and Muhan Zhang. Neural common neighbor with completion for link prediction. *arXiv preprint arXiv:2302.00890*, 2023. doi: 10.48550/arXiv.2302.00890.

[34] Yu Wang, Tong Zhao, Yuying Zhao, Yunchao Liu, Xueqi Cheng, Neil Shah, and Tyler Derr. A topological perspective on demystifying gnn-based link prediction performance. *arXiv preprint arXiv:2310.04612*, 2023.

[35] Bastian Wittmann, Johannes C Paetzold, Chinmay Prabhakar, Daniel Rueckert, and Bjoern Menze. Link prediction for flow-driven spatial networks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2472–2481, 2024.

[36] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 346–353, 2019.

[37] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020.

[38] Zuoyu Yan, Tengfei Ma, Liangcai Gao, Zhi Tang, and Chao Chen. Link prediction with persistent homology: An interactive view. In *International conference on machine learning*, pages 11659–11669. PMLR, 2021.

[39] Shangying Yang, Yinglong Zhang, Jiawei E, Xuewen Xia, and Xing Xu. Preserving node similarity adversarial learning graph representation with graph neural network. *Engineering Reports*, page e12854, 2024.

[40] Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10737–10745, 2021.

[41] Muhan Zhang. Graph neural networks: Link prediction. In Lingfei Wu, Peng Cui, Jian Pei, and Liang Zhao, editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 195–223. Springer Singapore, Singapore, 2022.

[42] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.

[43] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073, 2021.

[44] Tong Zhao, Gang Liu, Daheng Wang, Wenhao Yu, and Meng Jiang. Learning from counterfactual links for link prediction. In *International Conference on Machine Learning*, pages 26911–26926. PMLR, 2022.

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

[46] Hao Zhu, Mahashweta Das, Mangesh Bendre, Fei Wang, Hao Yang, and Soha Hassoun. Few-shot graph link prediction with domain adaptation.

[47] Jiong Zhu, Gaotang Li, Yao-An Yang, Jing Zhu, Xuehao Cui, and Danai Koutra. On the impact of feature heterophily on link prediction with graph neural networks. *arXiv preprint arXiv:2409.17475*, 2024.

[48] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021)*, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/f6a673f09493afcd8b129a0bcf1cd5bc-Abstract.html`.