Node concordance: a local homophily prediction task in graphs

Caio Lorenzetti Martinelli

Thesis presented to the
Institute of Mathematics and Statistics
of the University of São Paulo
in partial fulfillment
of the requirements
for the degree of
Master of Science

Program: Mestrado em Ciência da Computação

Advisor: Prof. Dr. Denis Deratani Mauá

São Paulo
July, 2023

Node concordance: a local homophily prediction task in graphs

Caio Lorenzetti Martinelli

This version of the thesis includes the corrections and modifications suggested by the Examining Committee during the defense of the original version of the work, which took place on July 31, 2023.

A copy of the original version is available at the Institute of Mathematics and Statistics of the University of São Paulo.

Examining Committee:

Prof. Dr. Denis Deratani Mauá (Advisor) – IME-USP
Prof. Dr. Bruno Ribeiro – Purdue University
Prof. Dr. Fabricio Murai Ferreira – Worcester Polytechnic Institute

Resumo

Martinelli, C. L. **Concordância entre nós: uma tarefa de predição de homofilia local**. Tese (Mestrado em Ciências) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

Homofilia é uma característica presente em muitos grafos do mundo real, esse trabalho propõe a tarefa de prever o manifestação local dela, a concordância entre nós. A tarefa é explorada em conjuntos de dados referência de predição de nós, usando os rótulos dos nós para criar o rótulo de concordância, e dois frameworks, um posicional e um estrutural, para uma versão semi-supervisionada da tarefa são propostos. Nesses conjuntos de dados, a tarefa pode ser vista como uma subtarefa da classificação de nós, nós queremos prever se dois nós são de mesma classe, sem levar em conta a quais classes eles pertencem. É mostrado que existe um vantangem de performance em atacar o problema de concordância de nós diretamente nesse caso. Os frameworks consistem em utilizar Graph Neural Networks (GNNs) e Node2Vec para gerar embeddings de nós que são informativos da concordância entre nós. O framework posicional é treinado de maneira não-supervisionada, tendo como objetivo, na verdade, a predição de arestas, usando apenas a topologia do grafo como recurso, e apresenta poder preditivo para concordância entre nós – apesar de que a relação entre os poderes preditivos para concordância entre nós e predição de arestas não é direta, como é mostrado nesse trabalho –. Os embeddings estruturais são treinados diretamente para concordância entre nós, usando as variáveis explicativas dos nós e mecanismos convolucionais das GNNs, e geralmente performam melhor do que o framework posicional, mas são mais sensíveis ao número de arestas rotuladas. Também é mostrado que os dois frameworks podem ser usados em combinação, uma vez que eles contém informações complementares um ao outro. Essa tarefa pode ter como fim ela própria se alguém quiser apurar exatamente a concordância entre nós de um grafo, ou pode servir como um passo de pré-processamento, para atribuir pesos às arestas ou alterar e fazer projeções de um grafo. O código desse trabalho é disponibilizado publicamente em https://github.com/caiolmart/node-concordance.

Palavras-chave: Graph Neural Networks, Tarefas em Grafos, Classificação de Nós, Predição de Arestas, Topologia de Grafos, Graph Representation Learning, Homofilia.

Abstract

Martinelli, C. L. **Node concordance: a local homophily prediction task in graphs**. Thesis (MSc) - Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

Homophily is a characteristic present in many real-world graphs, this work proposes a task to predict the local manifestation of it, the node concordance. The task is explored in benchmark datasets for node classification, using node labels to create the concordance label, and with two frameworks, one positional and one structured, for a semi-supervised version of the task are proposed. In those datasets, the task can be viewed as a subtask of the node classification, we want to predict if two nodes are same-class nodes, not taking into account which classes the nodes belong to. It is shown here that there is a performance advantage in tackling node concordance directly in this case. The frameworks consist of utilizing Graph Neural Networks (GNNs) and Node2Vec to generate node embeddings that are informative of the node concordance. The positional framework is trained in an unsupervised manner, actually targeting link prediction, using the graph topology as its only feature, and is shown to hold predictive power for node concordance – although the relation between the link prediction and node concordance predictive powers is not direct, as is shown in this work –. The structural embeddings are trained directly for node concordance, using node features and GNNs convolutional mechanisms, and generally perform better than the positional framework, but are more sensitive to the number of labeled edges. It is also shown that the two frameworks can be used in combination, in an ensemble, since they contain complementary information to each other. This task can be an end in itself if one desires exactly to assess the node concordance of the nodes, or can serve as a preprocessing step, to attribute edge weights or rewire and make projections of the graph. The code of this work is made publicly available on https://github.com/caiolmart/node-concordance.
Keywords: Graph Neural Networks, Tasks on Graphs, Node Classification, Link Prediction, Graph Topology, Graph Representation Learning, Homophily.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graphs are flexible data representations that are formed by sets of nodes (or vertices) and links (or edges, or arcs). The nodes are connected by those links, that can be directed or undirected. Both nodes and edges can be associated with additional information such as costs, weights, feature vectors, etc. They are vastly used to represent relational data, such as financial transactions ([XSS+21]), social networks ([WLX+20]), citation networks ([WSH+20], [SNB+08]), and web pages and their interlinked structure ([PBMW99]). Those are related to widespread aspects of nowadays society, which makes them a topic of interest for both industry and academia.

Being present in many areas, one may want to extract or predict relevant information about the graph or its components. Typical prediction tasks on graphs are categorized into node property prediction – e.g. in a financial network, we may be interested in predicting whether a node (person) is a fraudulent actor –, link property prediction – e.g. in a social network, we may be interested in predicting whether there is a missing link (friendship) in the network to suggest creating such a link –, and graph property prediction – e.g. in a graph representing a molecule, we may be interested in predicting whether the molecule inhibits the replication of a virus or interacts with some other molecule.

A property present in many graphs that is a key motivation to this work is homophily, the principle that edges happen more frequently between nodes that are similar [MLJ01]. Another form to view that is: nodes that are connected tend to share key properties. For example, people who buy from the same stores may have similar incomes, friends on social networks may share political views, and articles that cite each other may belong to the same subject.

This work proposes a novel task (and frameworks to tackle it), a link property prediction task that is closely related to the node property prediction, as will be argued. Given a graph, we want to predict whether two nodes belong to the same class, what we call here the concordance of nodes, a local prediction of the homophily of the graph.

This task is a link property prediction since the concordance labels and predictions are computed for node pairs, but is related to the node property prediction task because one could predict the class of each node and compare the predictions to achieve a concordance prediction. Predicting concordance directly is a more specific task than node classification – we cannot directly derive the node classes from concordance predictions; we want to distinguish same-class pairs, from inter-class pairs, not considering which class the nodes belong to. It is shown here that there is a gain in performance in directly tackling this task, which is in line with Vapnik's principle that if we have a restricted amount of information to solve a problem, we should solve it directly, never solve a more general problem as an intermediate step [Vap98].

A semi-supervised setting for this task is explored here. Given a graph with partially labeled nodes – and consequently partially labeled edges –, we model a function that can predict if any pair of nodes belong to the same class. Figure 1.1 exemplifies the task for two graphs, one homophilic and another heterophilic, scoring all the edges in the graph.

Homophily is a characteristic of many real-world graphs, being an assumption of the most popular Graph Neural Networks (GNNs) architectures. Predicting node concordance can be a desired

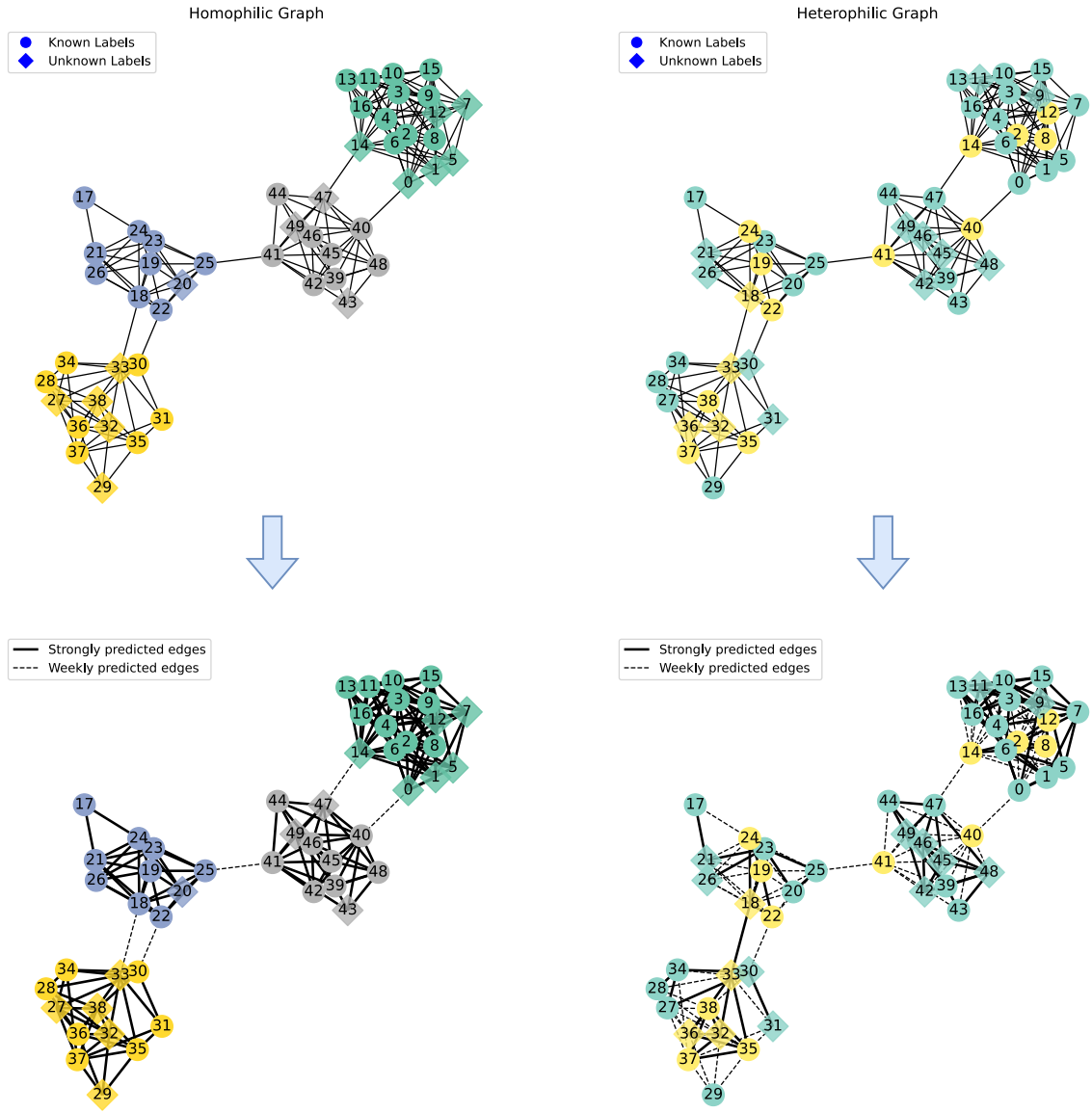

Figure 1.1: *Example of the proposed task predictions in a homophilic and a heterophilic graph. Node classes are represented by their colors. Nodes with known labels are displayed as circles, while nodes with unknown labels are as diamonds. The task consists of learning a function that generates concordance predictions for any pair of nodes, in this case, the edges of the graph are the pairs predicted and the function should return strong (high probability) predictions for the same-class edges, and weak (low probability) predictions for the inter-class edges.*

application for many use cases. This concordance measure could be used as a predictor of network dynamics - e.g. a user of a social network that is isolated, only having connections that are too different from him may exit the social network. In a recommender system, it can be used to suggest meaningful connections, or even balance the number of suggestions the recommender system is making (it can be the case that it only recommends concordant connections, which may not be of interest to a healthy network, encouraging polarization). Other questions may arise, and be answered with the help of such predictors, such as: Are concordant connections more long-lasting than discordant ones?

Since many GNN architectures assume homophily in their designs, it can also be the case that exploring node concordance predictors may serve as inspiration for new GNN architectures. Graph Attention Networks [VCC+17] have an approach to weighing neighbors in their convolutions. Could

a node concordance predictor be used to weigh edges? Or could it be used as a pre-processing of the graph? For example, one could rewire the graph using the predictor, making the graph more (or less) homophilic. Or could the concordance predictions be used in a neighborhood sampler of some novel GNN architecture – similar to what is done in [LAQ+21] –. Those are some possible applications that may be practically interesting, among others that the author cannot foresee.

Such as almost every machine learning problem in graphs, the node concordance prediction problem is one that has a particular setup. In graphs, it is not trivial to separate the data in train, validation, and test sets, since the samples are not independent and identically distributed (IID) in their essence. Frameworks to train and predict node concordance are presented in this work, with thorough discussions of their data splits, and their strengths and weaknesses.

Here are presented two frameworks, one using structural and the other using positional embeddings (see section 2.6). The two frameworks are trained in different manners, with different data splits and loss functions. The structural embeddings used here are trained in a supervised manner, directly targeting the node concordance, using edges that connect labeled nodes, while the positional embeddings are trained in an unsupervised manner, they are trained to predict the existence of edges, which is shown to be a good predictor of node concordance, although the optimal link predictor may not be the optimal concordance predictor, as will be discussed. The two frameworks can be complementary, and they use GNNs and Node2Vec. Before diving into the frameworks we quickly summarize these algorithms.

GNNs are special-purpose neural networks that can tackle the before-mentioned tasks in graphs, in which they have played a central role. There are a vast number of techniques derived mainly from Graph Convolutional Networks (GCNs) [KW17] and GraphSage [HYL17] - which can be seen as generalizations of Convolutional Neural Networks (CNNs, [LBBH98]) -, that exploit the model architecture, such as Graph Attention Networks (GATs) [VCC+17] – weighing the neighborhood of the nodes –, simplified GCN [WZdS+19], and DeeperGCN [LXTG20], among others.

GNNs extract features from the graphs, that are informative for the task they are being trained to solve. For each node, each of their layers contains a relatively low-dimensional array, an embedding, that characterizes it. The node embeddings can be positional or structural (see section 2.4), depending on the setting of the GNN, which is generally associated with the location or role of the nodes, respectively. Other forms of extracting positional embeddings include the frameworks DeepWalk ([PARS14]), Node2Vec ([GL16]), and TransE ([BUGD+13]).

These models are used here as building blocks of the two mentioned frameworks, one supervised and another unsupervised.

The here proposed supervised framework relies on building structural embeddings from node features, using GNNs. The GNNs are responsible for weighting and combining node features (from each node of the pair that we want to assess the concordance), and the final prediction is given by a comparison of the embeddings of the two nodes of interest. The training procedure directly targets the concordance label. The basic idea is that we can extract from the GNNs embeddings that when compared are informative of the equality of the node classes – or if we focus on single nodes (without GNN aggregations), we can compare the features to predict if the nodes belong to the same classes. This approach can be used in both transductive and inductive settings.

Whilst the unsupervised framework relies on utilizing positional embeddings to assess the node concordance. The positional embeddings are obtained using solely the graph structure, with no node features. For a typical positional embedding, it will explore the homophily of the graph to generate concordance predictions, and this approach, therefore should only work on homophilic graphs. Take figure 1.1 for example, the positions of the nodes in the plot can be seen as positional embeddings, the approach consists in generating predictions from the comparison of two node embeddings: close embeddings will generate high probabilities and distant embeddings, low probabilities. In a homophilic graph, these predictions will be accurate to node concordance – in the figure, to answer if two nodes are the same color –, but in heterophilic graphs, a typical positional embedding – one that is related to the shortest path distance between nodes – will not work. Nevertheless, positional embeddings can be informative of other aspects of the node, e.g. Node2Vec can capture node roles

in the graph (see sections 2.5.2 and 5.2.2), and this framework could also work on heterophilic graphs. This approach is, in essence, transductive, since for generating informative positional node embeddings, we need to include test/validation nodes in the training procedure.

The approaches are compared in quantitative - what is the predictive power of each approach and how are they correlated (in different datasets) - and qualitative - discussion of use cases, where we could prefer one approach over the other - manners. We show that edge homophily predictive power can be extracted from both approaches, that they can be used in a complementary manner, and that the unsupervised approach may be preferable in settings with fewer labeled nodes, although generally being outperformed by the supervised approach.

This work is organized as follows: The second chapter contains key concepts that are used through the experiments. It contains an introduction to the prediction tasks on graphs, with a deeper explanation of the tasks that will be used by the author. Then, it is given an introduction to GNNs, their assumptions, usage, and architectures. We discuss the differences between structural and positional embeddings and go through other approaches to generate positional embeddings in graphs. And, finally, explain and discuss the differences between inductive and transductive learning. The third chapter describes the proposed frameworks, passing through their target function, architecture, training procedure, and data splits. The fourth chapter describes the experimental design, it presents the datasets used and the evaluation format of the proposed frameworks. The fifth chapter contains the results and discussion. It is separated into research questions, each one containing the relevant results to answer them. And the sixth and final chapter contains the conclusions and future work.

Chapter 2

Concepts

This chapter explains briefly the key concepts used in this research.

2.1 Basic concepts

A graph is a collection of nodes and edges that can have a series of attributes associated with them. The information contained in a graph can be as simple as the existence of nodes and the edges that connect them, and hence the existence of the graph, but it also can have intrinsic information: The nodes and edges can have features, for example, in a social network a person (node) may have a name, age, address, among other features, a friendship (edge) may store the number of interactions in the social network, the age of the connection, the date of the last interaction, etc.

The graph can also have its properties. Considering various graphs, each representing a molecule, where the atoms are the nodes and the chemical bonds are the edges. Each graph may have properties such as molecular mass, reactivity to a molecule, the ability to inhibit the replication of a virus, and others.

More formally, a graph G is necessarily described by its nodes (V) and connections (E), which are represented here using an adjacency matrix $A : |V| \times |V|$, with $|E|$ non-zero elements. If node features are present, they are described by a matrix $X : |V| \times k$ where k is the number of features. And finally, if edge features are present they can be described in several manners: if the features are one-dimensional, such as edge weights, they can be stored directly in the adjacency matrix A, whereas if there are more features to be stored (l features), there are some possibilities, such as adding another dimension to the adjacency matrix $\hat{A} : |V| \times |V| \times l$, or creating a map that stores the edge features using the position in the matrix as the map key.

2.2 Prediction Tasks

The prediction tasks in graphs typically try to infer one of the properties described in section 2.1. In the two subsections are explained the two tasks used in this work: node and link properties prediction tasks.

2.2.1 Node Property Prediction

The node property prediction task consists of using the graph's available information to infer a property attributed to the node. For example, in a financial network, where nodes are people or companies and edges are financial transactions, we may want to predict if a node is a fraudulent actor.

If available, we may want to use the node features (X), such as age, income, past financial activity, to classify it as a risky user. It also could be useful to use the same information of its neighbors (nodes that are connected to it - nodes that transacted money with it), which we could even weigh by the value of the using the value of the transactions (edge property). We could also

iterate over the neighborhood and get the neighbors of the neighbors, using possibly the information of the whole graph to predict the label of a single node. So here is exemplified the usage of the node and edge properties to make a node property prediction. We could use graph properties to make the node prediction as well. For example, if the graph is disconnected, we could use the number of past frauds that occurred in the connected component in which the node is, or even the number of past frauds in the k-neighborhood, k being the maximum distance in financial transactions. Although, in our case, this information may be redundant with the usage of the node and edge properties.

In other words, we want to learn a function f that approximates the desired property of the node, Y_i, receiving the graph G (with all the information of the nodes and edges) and the node i:

$$f(G, i) \approx Y_i. \tag{2.1}$$

2.2.2 Link Property Prediction

The link property prediction task consists in using the available information of the graph G to infer a property that is attributed to the node. For example, in a social network (as exemplified at the beginning of the section), we may want to predict the number of interactions two users will have in the next month. We can use the properties of the two users, such as past usage metrics of the social network, and the properties of the edge, such as metrics related to common friends, such as the Jaccard or Adamic Adar coefficients of the set of neighborhoods of the two nodes, or even a past number of interactions between the two nodes.

In other words, we want to learn a function f that approximates the desired property of the edge Y_{ij}, receiving the graph G (with all the information of the nodes and edges) and the two nodes (i, j) that are connected by the edge:

$$f(G, i, j) \approx Y_{ij}. \tag{2.2}$$

The main difference in the modeling of this problem is in the task of predicting the existence of future edges. In this task, we typically have a graph G_0 in an initial state and must predict which edges will be added in the future (when the graph will be G_t). Our target function then is

$$Y_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in G_t, \\ 0 & \text{if } e_{ij} \notin G_t. \end{cases} \tag{2.3}$$

So, we will try to approximate a function that can construct the future edges based on the initial graph:

$$f(G_0, i, j) \approx Y_{ij}. \tag{2.4}$$

Since graphs, especially large ones, are usually sparse, the probability of an edge between two random nodes to exist is close to zero. What leads us to some particularities of this task:

- Evaluation metrics: Accuracy can mislead the performance of our functions since the dominant class (non-existence of the edge) is very probable. Additionally, it gets computationally expensive to calculate a metric such as accuracy over the whole graph. In a directed graph with N nodes, there are $N(N-1)$ possible edges. Therefore, to compute a metric over the whole graph, the computational cost grows with $\Theta(N^2)$. The solution to this is typically subsampling the negative class. Two metrics that are widely used are:

  Hits@K: Which evaluates the percentage of positive edges in the test set T that are within the top K predictions when compared to a large set of negative edges,

$$\text{Hits@}K = \frac{1}{|T|} \sum_{e_{ij} \in T} 1 \text{ if } \text{rank}(e_{ij}) \leq K. \tag{2.5}$$

MRR: Mean Reciprocal Rank, which evaluates the rank of every positive edge in the test set T compared to a large set of negative edges,

$$\text{MRR} = \frac{1}{|T|} \sum_{e_{ij} \in T} \frac{1}{\text{rank}(e_{ij})}. \tag{2.6}$$

- Training: Similar to the metrics computation, it is very expensive to train our function over all the negative edges. A widely used solution to this is to randomly sample negative edges over training epochs.

2.3 Graph Neural Networks

As mentioned in the introduction, there is a vast number of techniques in Graph Neural Networks, derived mainly from Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017 [KW17]) and GraphSAGE (Hamilton et al., 2017 [HYL17]). Both can be understood to make spatial-based convolutions over nodes, and can be seen as generalizations of Convolutional Neural Networks (CNNs) [LBBH98] that have for long been present in computer vision research.

The premises of such techniques are that the node features and labels vary smoothly over the graph, and that there is a gain in predictive power in using not only the node features but also the neighborhood features. The idea is to aggregate the information of the node with the information of the neighbors, learning the influence the neighbors can exert over the node or link to be classified.

The convolutions are over the immediate neighborhood of the nodes, and each convolutional layer can be understood to aggregate information of nodes that are more distant from the node to be classified. Let us first consider the convolutions for node classification. Figure 2.1 exemplifies the calculation of the final node representation in a 2-layer graph convolutional network. Here, the layer (l) convolutions appear in the form of a function $f^{(l)}$ that receives as node states at the previous layer $(h_{(i)}^{(l-1)})$ of both the node itself as well as the representation of the neighbors. In the node classification task, we typically have node features (X_i), and the first node representation is defined to be the node features, $h_i^0 = X_i$. Note that in a graph the neighborhood size is variable. So our function must deal with any neighborhood size, there will be examples of such functions. As mentioned, the more layers are added to the network, the more distant information can be used for the node classification. In the example, we can see that the representation of the node 0 in the first layer aggregates information of the immediate neighborhood and the representation of the node in the second layer aggregate information of the whole graph.

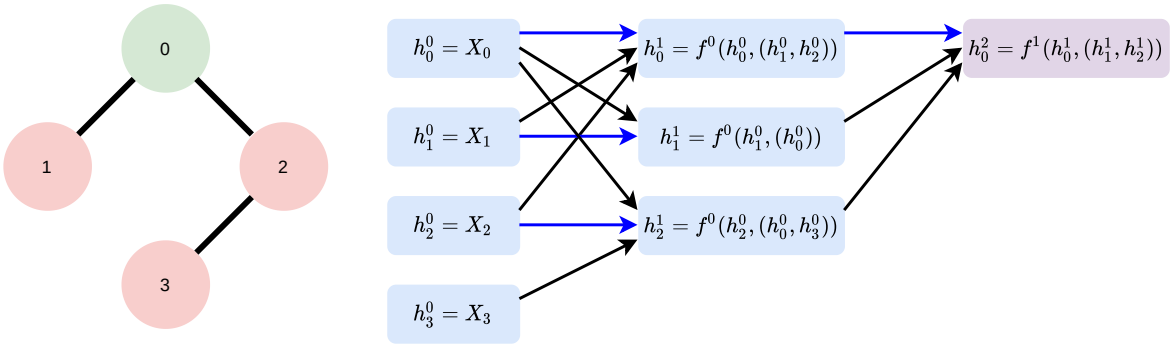

Figure 2.1: *Example of a two-layer convolutional network calculation of the representation of a node in a graph.*

These convolutions can be used also for link prediction tasks. The differences are that typically the node features are not used, typically the first layer representation is an embedding of the nodes, $H^0 = Z$ that are initialized randomly and learned through the training process, and the

training process uses the node representations at the last layer (L) and tries to learn a function that approximates the target in equation 2.3. Our function (from equation 2.4), then, is

$$f(G_0, i, j) = \Lambda(h_i^L, h_j^L) \approx \begin{cases} 1 & \text{if } e_{ij} \in G_t, \\ 0 & \text{if } e_{ij} \notin G_t. \end{cases} \tag{2.7}$$

In the following subsections, we will understand better the two techniques the Graph Neural Networks are mainly derived from.

2.3.1 Graph Convolutional Networks

The GCN algorithm adds self-loops to the graph $G = (A, X)$: $\tilde{A} = A + I_N$ where A is the adjacency matrix and I_N the identity matrix, in a way that the following layer loop consider features of the node, as well as the neighborhood features:

$$h_i^{(l+1)} = \sigma(b^{(l)} + \sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ji}} h_j^{(l)} W^{(l)}) \tag{2.8}$$

Where σ is an activation function, $b^{(l)}$ is a bias term, c_{ji} is the product of the square root of the node degrees (i.e., $c_{ji} = \sqrt{|\mathcal{N}(j)|}\sqrt{|\mathcal{N}(i)|}$), and $W^{(l)}$ is a weight matrix to be learned through the training process.

We can understand this as a weighted average of the node representations in the previous layer over the neighborhood, transformed linearly (added a bias term), and transformed by an activation function such as RELU. Here we can note that this function can be applied to nodes with neighborhoods of any size, a property we need to be satisfied.

2.3.2 GraphSAGE

The GraphSAGE algorithm treats separately the node's representation and its neighborhood. It learns weights that transform the node representation and the neighbors representations separately, and its architecture enables it to use other forms of aggregation than the weighted average. The layer loop is

$$\begin{aligned} h_{\mathcal{N}(i)}^{(l+1)} &\leftarrow \text{aggregate}(\{h_j^l, \forall j \in \mathcal{N}(i)\}), \\ h_i^{(l+1)} &\leftarrow \sigma(W^{(l)} \cdot \text{concat}(h_i^l, h_{\mathcal{N}(i)}^{(l+1)})), \\ h_i^{(l+1)} &\leftarrow \text{norm}(h_i^{(l+1)}). \end{aligned} \tag{2.9}$$

Here, the aggregation step consists in gathering information of the previous layer in the neighborhood. This step must be able to deal with any neighborhood size. Plus, since in a graph nodes have no natural ordering, the aggregator must operate over an unordered set of representations of the previous layer, and ideally would be invariant to permutations of these representations.

The article presents three candidates:

- Mean aggregator: Takes the mean of the vector representations of the neighbors. It is the aggregator that most resembles the GCN one. The main difference is that the mean is calculated only over the neighbors, and then concatenated with the self-node representation in the previous layer. For instance, if d^l is the dimension of the representation in the layer l, the concatenated vector, $\text{concat}(h_i^l, h_{\mathcal{N}(i)}^{(l+1)})$ would have dimension $2d^l$, and then, W would be of dimension $d^{l+1} \times 2d^l$. The mean aggregator respects the invariance property and does not add any learnable parameters to the process.

- Pooling aggregator: A trainable aggregator which allows nodes to contribute partially to the $h_{\mathcal{N}(i)}^{(l+1)}$ vector:

$$\text{aggregate}_l^{\text{pool}} = \max(\{\sigma(W_{\text{pool}}^{(l)} h_j^{(l)} + b^{(l)}), \forall j \in \mathcal{N}(i)\}). \tag{2.10}$$

Where max denotes the element-wise max operator, σ is an activation function and $b^{(l)}$ is a bias term. Here, the vector $h^{(l+1)}_{\mathcal{N}(i)}$ can take information from only one node up to the dimension of the layer, d^l, nodes. This aggregator has also the permutation invariance property.

- LSTM aggregator: A more complex aggregator, which satisfies our needed property of being able to deal with any neighborhood size, and has greater expressive capability, but it is not permutation invariant. Here is needed to apply the LSTM to a random permutation of the neighbor node's representations.

2.3.3 Graph Attention Networks

Graph attention networks [VCC+17] introduce a means of weighting the different neighbors of a node, according to the importance of their representations to the new aggregated representation.

Each layer will compute a weighted mean of the linearly transformed - by the learnable W^l matrix - representations of the previous layer of the node and its neighbors:

$$h_i^{(l+1)} = \sigma\Big(\sum_{j\in\mathcal{N}(i)} \alpha_{ij} W^{(l)} h_j^{(l)}\Big). \tag{2.11}$$

The attention resides on the computation of the α_{ij} parameters, the neighbor weights. The neural network architecture must be able to compute such parameters for any neighborhood size, and also compute the self-importance, determined by the α_{ii}.

For that, the authors chose to use the same $W^{(l)}$ matrix that will eventually be used in equation 2.11 to pre-process the representation of the previous layer, concatenate these transformed representations of nodes i and j, and calculate the importance signal (d_{ij}) using the attention vector $a^{(l)}$ and a LeakyReLU:

$$d_{ij} = \text{LeakyReLU}(a^{(l)T}\text{concat}(W^{(l)}h_i^{(l)}, W^{(l)}h_j^{(l)})). \tag{2.12}$$

Note that if $h_i^{(l)} \in \mathbb{R}^{F^{(l)}}$ and $h_i^{(l+1)} \in \mathbb{R}^{F^{(l+1)}}$, then $W^{(l)} \in \mathbb{R}^{F^{(l+1)}\times F^{(l)}}$ and $a^{(l)} \in \mathbb{R}^{2F^{(l+1)}}$.

Finally, the neighbor weights can be calculated by a softmax using the importance signal:

$$\alpha_{ij} = \text{softmax}_j(d_{ij}) = \frac{\exp(d_{ij})}{\sum_{k\in\mathcal{N}(i)}\exp(d_{ik})}. \tag{2.13}$$

2.4 Positional vs. Structural Embeddings

The typical views regarding positional and structural embeddings are the following: Positional embeddings are associated with the notion of closeness, close nodes - nodes that are connected by short paths - should have similar node embeddings. Whereas structural representations say more about the role of the node in the graph - is it a node that connects two clusters (communities), is it a central node, etc. Isomorphic nodes should always have the same structural embedding [SR20].

For example, in the graph in figure 2.2, nodes 1 and 7 should have different positional embeddings, but the same structural representation: They are far apart, but they are isomorphic - both are green, are connected to two nodes (one yellow and one green), the green connection is connected to another yellow node, and so on.

GNNs can generate both positional and structural embeddings. The most common case is when they are used for node classification, where the node features are given, and it iterates over the features of some neighborhood of the node of interest. This is an example of structural embedding: if we have two similar nodes that are very far apart in the graph with similar neighborhoods, they will have similar embeddings given by this common GNN. But, as mentioned in section 2.3, for any graph we can initialize randomly the set of node embeddings (instead of using the node features), and update them (together with the GNN weights) to predict closeness. And this will be a case of positional embeddings generated by a GNN.

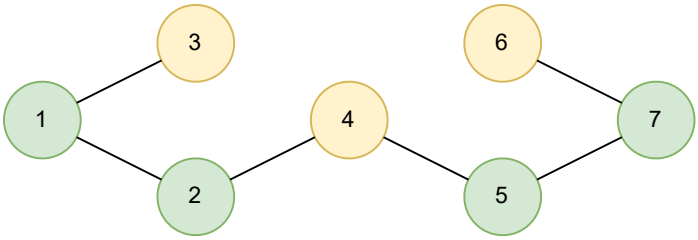

Figure 2.2: *Example of an undirected graph with binary node features represented by the node colors.*

2.5 Positional Embeddings in Graphs

Each hidden layer of a node in GNNs can be viewed as an embedding of that node. There are other approaches that generate those embeddings, typically used for the link prediction task, that have appeared before GNNs in the graph representation learning field.

In this section, I present three techniques regarding embedding generation in graphs, two that resemble word embedding used in language modeling: DeepWalk (Perozzi et. al., 2014 [PARS14]) and Node2Vec (Grover et. al., 2016 [GL16]), and one that is native to graphs (particularly, Knowledge Bases, which are essentially multi-graphs): TransE (Bordes et. al., 2013 [BUGD+13]).

2.5.1 DeepWalk

In language modeling, the required inputs are the vocabulary ($\mathcal{V}$) and the corpus. Meanwhile, DeepWalk considers the set of graph vertices its vocabulary ($\mathcal{V} = V$) and the corpus is generated by short truncated random walks.

The algorithm consists of a random walk generator (exemplified in figure 2.3) and an update procedure for the representation matrix, $\Phi \in \mathbb{R}^{|V| \times d}$, which is initialized randomly. The nodes are sampled uniformly, as the root of the walk, and the walks sample uniformly the neighbors of the last node. The update procedure is the SkipGram (Mikolov et. al, 2013 [MCCD13]), a language model that maximizes the co-occurrence probability of words within a window w in a sentence.

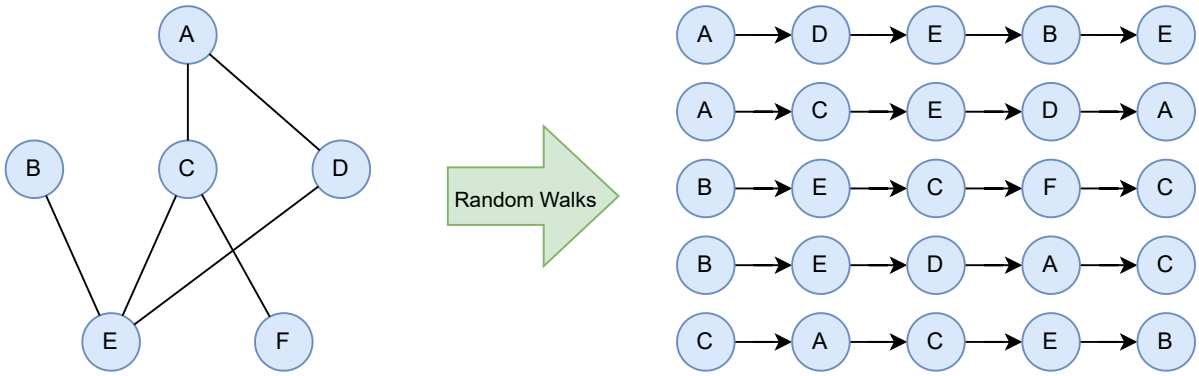

Figure 2.3: *Illustration of a random walk generation procedure.*

In the link prediction task, those learned representations are input to a model that returns a score for the link between two nodes. This score can be as simple as the cosine similarity between the representations of the two nodes (which can be scaled to satisfy the $[0, 1]$ range), or it can be a model with learnable parameters.

The DeepWalk algorithm appears at the Open Graph Benchmark (OGB - Hu et al., 2020 [HFZ+20], see section 4.1) link property prediction leaderboards. The code provided on OGB shows that a typical usage is to train a Multi-Layer Perceptron (MLP) with the element-wise multiplication ($*$) of the representation of the nodes. What can be written as:

$$f(G_0, i, j) = \text{MLP}(\Phi(i) * \Phi(j)). \tag{2.14}$$

2.5.2 Node2Vec

Node2Vec (Grover et. al., 2016 [GL16]) is a very similar framework to DeepWalk (Perozzi et. al., 2014 [PARS14]). It generates node embeddings through the SkipGram algorithm (Mikolov et. al., 2013 [MCCD13]), where the vocabulary is the set of nodes and the corpus is generated by random walks. The framework can be viewed as a generalization of the DeepWalk, as the latter poses as a special case of the former.

The framework introduces biased random walks, with parameters to the walk generation that can make the embeddings to be more representative of the local or global structure aspects of the graph.

The probability of x be the next node in the walk, in which the last node is v is given by

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v,x) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{2.15}$$

Where π_{vx} is the transition probability between nodes v and x and Z is a normalizing factor. The transition probability depends on the weight of the edge, w_{vx}, and also on the node previous to the last node in the walk, $\pi_{vx} = w_{vx} \cdot \alpha_{pq}(t,x)$, i.e., if the walk is currently $[\ldots, t, v]$, the probability of the next visited node to be x depends on both the weight of the edge that connects it to v and a short term memory of the walked path, where α_{pq} is the search bias, defined as

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0, \\ 1 & \text{if } d_{tx} = 1, \\ \frac{1}{q} & \text{if } d_{tx} = 2. \end{cases} \tag{2.16}$$

Where p and q are parameters that introduce the bias to the walk, and d_{tx} is the shortest path distance from t to x. The parameter p influences the return probability to t, low p values increase the chance of the walk to return to t, while high values diminish this chance. And q regulates the "inward" versus "outward" nodes preference on the search, a high q increases the probability of the walk to stay closer to the previous node t, or even form triangles during the walk, while a low q introduces a bias that repels the walk from the previously visited nodes, exploring the graph further away. Compared with the DeepWalk, presented in the previous subsection, the Node2Vec generalizes its walk sampling strategy, and they become the same strategy for $p = 1, q = 1$.

2.5.3 TransE

TransE is a framework designed to deal with Knowledge Bases (KBs), introduced by Bordes et. al., 2013 [BUGD+13]. Similar to the two previous frameworks, it learns low dimension representations of nodes (or entities). The approach turns out to be different when it tries also to learn the relationships (edges) as translations on the entities embedding space. While the two previous frameworks do not discern between distinct types of edges - and there are many graphs with a single type of edge -, this framework encodes different edge types and different translations in the embedding space.

The framework has as input a set S of triplets (h, l, t) (head, label, tail) that represent the edges of the graph, h and t are entities within the set of nodes, V, and l are the relationships, within the set of edges, E. The node embeddings take values in $\mathbb{R}^k$, k being the dimensionality of the embedding, and the relationships are translations in this space, which can be viewed as an edge embedding. The model tries to learn the node and edge embeddings such that $h + l \approx t$, or in more objective form, the distance $d(h + l, t)$ must be smaller than the distance $d(h + l, t')$ if the triplet (h, l, t') does not exist, and the same holds for some inexisting triplet (h', l, t), with distance $d(h' + l, t)$.

To achieve that, the authors propose a learning method that consists on sampling the existing triplets from S, negative sampling "corrupted triplets" for each sampled triplet:

$$S'_{(h,l,t)} = \{(h', l, t) | h' \in V\} \cup \{(h, l, t') | t' \in V\}. \tag{2.17}$$

And updating the embeddings using the stochastic gradient descent according to the loss function

$$\mathcal{L} = \sum_{(h,l,t)\in S} \sum_{(h',l,t')\in S'_{(h,l,t)}} [\gamma + d(h+l,t) - d(h'+l,t')]_+, \tag{2.18}$$

where $[x]_+$ denotes the positive part of x and $\gamma > 0$ is a margin hyperparameter.

2.6 Inductive and Transductive Learning

In graphs, we often have a tradeoff between our learning methods. We want our models to generalize to new observations, quickly classify them, and possibly be used in other graphs (inductive learning features), but at the same time, we want also our models to reach better performance, what can be obtained through transductive learning (Vapnik, 2013 [Vla13]).

There are approaches to gain performance that can be used only for the same graph that our model is trained onto. These approaches are vastly used in the literature, from classical models such as Label Propagation Algorithm ([Zhu05]), to more modern techniques that use those ideas to enhance GNN performance ([WL20], [CXH+20]), or even that simply adapt the algorithm to receive base probabilities and propagate them ([HHS+20]).

Consider we have a citation graph, where each node is a paper, with its text features, and each edge is a citation, and we want to classify the category of each paper. If we have our test nodes connected to our train nodes, we can use the labels of our train nodes to propagate the labels to our test nodes. But, if we have a new set of nodes, disconnected from our train nodes (no paths between train and test nodes), we are not able to make predictions using our train labels. The Label Propagation Algorithm is an example of transductive learning. Although it may be useful to classify our test nodes (if they are in the same graph as our train nodes), we are not able to generalize to another graph, even if the nodes in the other graph have features that come from the same distribution $P(x)$ from the train nodes, and the graph has similar properties to the train graph.

In other words, inductive learning will try to learn general rules from training cases, we must be able to apply those rules to any node that has features and edges that come from the same or similar distributions, whereas transductive learning will try to learn specific rules to the problem using all the data presented to the model.

Chapter 3

Embeddings for Node Concordance

This chapter describes the framework to generate and use embedding for node concordance prediction.

3.1 Structural Embeddings for Node Concordance Prediction

The structural embeddings in this work are trained directly to predict node concordance. All the graphs studied here contain node features (X), and the edges have binary node concordance labels, but no features.

3.1.1 Target function

To train our structural embedding we define our target in equation 3.1. The node concordance Υ, is the link property we want to assess, a binary measure of similarity between the two nodes it connects. In this framework, it is possible to use concordance directly as the target function for training our structural embeddings (Θ_s):

$$\Theta_s(i,j) = \Upsilon(i,j). \tag{3.1}$$

3.1.2 Architecture

The training process consists of learning a function (Ω_s) that is informative of the concordance of any pair of nodes. With some similarity to equation 2.7, our function is trained as

$$\Omega_s(i,j) = f(G,i,j) \approx \Theta_s(i,j). \tag{3.2}$$

This function utilizes structural node embeddings (h^L) of the pair of nodes to calculate its predictions:

$$\Omega_s(i,j) = \rho(h_i^L, h_j^L). \tag{3.3}$$

Here, it is particularly chosen to generate the predictions from the cosine similarity of the two node embeddings:

$$\rho(h_i^L, h_j^L) = \sigma(a + b\cos(h_i^L, h_j^L)), \tag{3.4}$$

where σ is the sigmoid function and a and b are two learnable parameters.

The node embeddings are calculated using standard GNNs (GraphSage, GCN, and GAT), where the first layer embeddings are the node features ($h^0 = X$).

This architecture is said structural because different (that can be distant) nodes of the graph will have the exact same embedding if they are isomorphic, or even locally isomorphic - if they have the same features, the same neighborhood structure (with the same neighborhood features), where the neighborhood size depends on the depth of the GNN.

As a baseline model, it is trained an MLP using only the node features (h^0 or X). Since comparing the features using solely the cosine similarity could be excessively simple, the node features are combined using the element-wise multiplication ($*$), and it is trained a concordance predictor MLP:

$$\Omega_s^{\text{baseline}}(i,j) = \rho^{\text{baseline}}(h_i^0, h_j^0) = \text{MLP}(h_i^0 * h_j^0). \tag{3.5}$$

3.1.3 Training

All the parameters of our Ω_s function are trained together. The parameters of the GNN are initialized randomly, according to the default implementation in Pytorch Geometric. As for the embedding comparison, we hope that similar embeddings generate high concordance predictions - the reason cosine similarity is used to compare them - what is further enforced in the initialization of its parameters: a is initialized as 0 and b as 1.

The loss function considered is the binary cross entropy between the predictions and the target Θ_s.

3.1.4 Data Split

The node concordance is a link property, but it is essentially a comparison metric between the characteristics of two nodes. Therefore, is constructed an edge split (in train, validation, and test sets) that is the result of a node split. That means that train edges connect two train nodes, test edges have at least one end in a test node, and the rest of the edges are validation edges.

Since this approach is structural, it is easier to create an inductive setting, we can use only train edges to train our embeddings and a test node that is isomorphic to a train node will have an identical node embedding, and so a pair of train nodes that are isomorphic to a pair of test nodes will have the same concordance prediction score. We can, therefore, use only train edges to create our model, and this model should be able to predict the concordance of validation/test edges.

To make a perfect inductive setting, we should also split our graph into train, validation, and test graphs, where the test graph would be the whole graph, the validation graph would exclude the test edges/nodes and the train graph would exclude both validation and test edges/nodes. For simplicity we do not make this separation, we follow the transductive setting, as do all the experiments in OGB leaderboards whose code the author inspected, of considering the whole graph but evaluating our losses and hence doing the backpropagation only on train edges. It is worth noting that in this case the features of validation/test nodes will be used in the convolutions of the GNNs.

3.2 Positional Embeddings for Node Concordance Prediction

The positional embeddings in this work are trained for link prediction. The Ω_p function that utilizes it must also be able to receive any pair of nodes, and it is trained to predict the existence of an edge between them. It is expected here that due to homophily, the link prediction is informative of the concordance. But, as will be further discussed in the results chapter, the best predictor of node concordance in this setting is not the best link predictor. The best link predictor would be able to perfectly answer if a link exists or not, giving no additional information to the graph itself. We, therefore, want an imperfect link predictor that maximizes the distinction between same-class and inter-class connections.

This approach is unsupervised, it does not utilize the node concordance label during its training. It also ignores the node features, using a randomly initialized embedding as its node's first layer representation.

3.2.1 Target function

To train our positional embeddings using GNNs, we use the target function that is implicitly described in equation 2.7, but specified here as

$$\Theta_p^{\text{GNN}}(i,j) = \begin{cases} 1 & \text{if } e_{ij} \in G, \\ 0 & \text{if } e_{ij} \notin G. \end{cases} \tag{3.6}$$

The target function of the unsupervised approach is a simple link prediction target, and does not directly optimize the node concordance prediction power.

It is specified that this target function serves our GNNs. Node2Vec is another framework for generating positional embeddings used here, where the target function does not distinguish between pairs that are or are not in the graph, but between pairs that are in the context window of the random walk or are in the false random walk generated during the sampling. This target function is particularly similar when we set the context size to 2, where all the positive optimizations will regard edges, but it is not guaranteed that all the graph edges will be used, and graph edges may be used at different proportions.

3.2.2 Architecture

The training process of our GNNs consists of learning a function (Ω_p^{GNN}) that is informative of the existence of edges between nodes. Our function is trained according to equation 2.7, but the whole graph is used in the process, leaving us to

$$\Omega_p^{\text{GNN}}(i,j) = f(G,i,j) \approx \Theta_p^{\text{GNN}}(i,j). \tag{3.7}$$

This function utilizes positional node embeddings (h^L) of the pair of nodes to calculate its predictions:

$$\Omega_p^{\text{GNN}}(i,j) = \rho(h_i^L, h_j^L). \tag{3.8}$$

Here is also chosen to generate the predictions from the cosine similarity of the two node embeddings, according to equation 3.4.

The node embeddings are calculated using standard GNNs (GraphSage, GCN, and GAT), where the first layer embeddings are the raw node embeddings ($h^0 = E$).

This architecture is said positional because distant nodes of the graph will tend to have different embeddings, regardless they are isomorphic, the optimization process will lead to embeddings that are informative of the distance between nodes, not informative of their quality or role in their local graphs.

The Node2Vec approach is very similar to the GNN approach, but the target function is proper from the model, and its architecture does not include a link function (ρ in the GNN framework), that will be used to compare node embeddings, only generating node embeddings (h^{Node2Vec}). Therefore, we must include a comparison between the node embeddings for this framework, which is chosen to be a sigmoid applied to the cosine similarity measure between the embeddings, according to equation 3.9.

$$\rho^{\text{Node2Vec}}(h_i^{\text{Node2Vec}}, h_j^{\text{Node2Vec}}) = \sigma(\cos(h_i^{\text{Node2Vec}}, h_j^{\text{Node2Vec}})) \tag{3.9}$$

We remove the parameters a and b from the equation since the ρ^{Node2Vec} will not participate in the optimization process, and hence the parameters cannot be optimized.

3.2.3 Training

All the parameters of our Ω_p functions are trained together.

The embeddings of both GNN and Node2Vec approaches are initialized randomly, according to the default implementation in Pytorch Geometric, i.e. normal distribution initialization. And in the GNN case, also the layer parameters are initialized randomly, according to the default implementation in Pytorch Geometric, a is initialized as 0 and b as 1.

Each training epoch consists of a whole pass on the training edges, with a same-sized negative sampling of edges, randomly extracted, in batches whose size (B) varies for each experiment.

The loss function ($\mathcal{L}_b$) for each batch b of the GNNs, with its positive (E_b), and negative ($\hat{E}_b$) edges is described as

$$\mathcal{L}_b = \frac{1}{B}(\sum_{e_{ij} \in E_b} - \log(\Omega_p^{\text{GNN}}(i,j)) + \sum_{\hat{e_{ij}} \in \hat{E}_b} - \log(1 - \Omega_p^{\text{GNN}}(i,j))). \tag{3.10}$$

3.2.4 Data Split

Similar to the structural embeddings, the framework will be evaluated by splitting the edges from a node split. But since this approach is positional, our setting must be transductive, all nodes must be contained in the training process, as the positional embeddings are calculated in the training process. If the nodes do not participate in the training processes, they would only have their randomly initialized embeddings as first-layer representations, which would not be informative at all.

Therefore we randomly split the edges between train, validation, and test, in sizes that vary according to each dataset. As the node labels (or even the features) are not used in this framework, this data split poses only as a performance report, that will not be used in the evaluation of the frameworks, which will be discussed in the results section.

Chapter 4

Experimental Design

This chapter presents the experimental design for evaluating the performance of the proposed frameworks in the node concordance prediction task.

4.1 Datasets

4.1.1 Description

It is interesting to evaluate our framework performance on realistic datasets, which can be a proxy for usability in real problems. In the absence of node concordance benchmark datasets, here are used benchmark datasets for node label prediction, from which is created our node concordance label.

The chosen datasets are the citation networks Cora, CiteSeer, and Pubmed from [SNB+08] and `ogbn-arxiv` from the Open Graph Benchmark (OGB - [HFZ+20]). All those datasets are graphs that contain node features (X) and labels (Y). The features are extracted from text, and the labels are categories of the articles the node represents. In the case of `ogbn-arxiv`, the node features are the average of the word embeddings in the title and abstract, and in the case of the Cora, CiteSeer, and Pubmed datasets, bag-of-words representations of the documents.

Table 4.1 shows descriptive statistics of the datasets. There are shown the dimension of the node feature vectors ($|X_i|$), the number of classes ($|\{Y\}|$), the homophily (h) – which is computed as the proportion of concordant edges in the graph – and the number of nodes ($|V|$) and edges ($|E|$) of their data splits.

4.1.2 Node concordance label

We extract our node concordance label (Υ) from the node labels. Edges that connect same-class nodes are concordant edges and otherwise are discordant edges:

$$\Upsilon(i,j) = \begin{cases} 1 & \text{if } Y_i = Y_j, \\ 0 & \text{if } Y_i \neq Y_j. \end{cases} \tag{4.1}$$

				$\lvert V\rvert$				$\lvert E\rvert$			
Dataset	$\lvert X_i\rvert$	$\lvert\{Y\}\rvert$	h	Total	Train	Validation	Test	Total	Train	Validation	Test
`ogbn-arxiv`	128	40	0.655	169,343	90,941	29,799	48,603	1,166,243	374,839	247,627	543,777
Cora	1433	7	0.810	2,708	1,208	500	1,000	10,556	2,308	2,130	6,118
Citeseer	3703	6	0.736	3,327	1,827	500	1,000	9,104	2,642	1,712	4,750
Pubmed	500	3	0.802	19,717	18,217	500	1,000	88,648	75,472	4,140	9,036

Table 4.1: *Descriptive statistics of the datasets used. Here are shown the dimension of the node feature vectors ($|X_i|$), the number of classes ($|\{Y\}|$), the homophily (h), and the number of nodes ($|V|$) and edges ($|E|$) of their data splits.*

It is worth noting that the labeling of the nodes in these datasets is manually made by either the authors or moderators of these portals. It could be the case that a node concordance label would be easier to achieve, or even more precise for some purposes. Categorizing papers can be rather arbitrary, there is always the possibility of papers that encompass multiple categories, and within a category, there can be papers that are rather distinct.

4.1.3 Data Splits

The Cora, CiteSeer, and Pubmed datasets have more than one data split proposed in the literature. Here is used the node split proposed by [CMX18], one to align with its semi-supervised learning scenario, that is shared here. The `ogbn-arxiv` dataset has one node split proposed, which is based on the time of the paper publication: papers published until 2017 are training nodes, those published in 2018 are validation nodes, and papers since 2019 are test nodes. This dataset provides the publication year of each paper, from where we are able to create new node splits using the same methodology.

Note that the proposed splits are based on nodes, not on edges, as we want - given concordance is a link property. To solve this issue, we consider any edge that has at least one end in a test node as a test edge, any edge that has an end in a validation node (but no test node) as a validation edge, and the rest as train edges. This split will be used for the evaluation of the node concordance prediction frameworks.

4.2 Evaluation

The evaluation of our node concordance predictor is done by comparing the predictions of our learned functions Ω against the node concordance itself (Υ). We want that our predictor is able to distinguish between same-class and inter-class edges, the same-class edges should have higher scores than inter-class ones. It is also desired that the scores do not necessarily reflect probabilities, since our unsupervised setup does not use the node concordance metric in the training process.

The ROC-AUC (Area Under the Receiver Operating Characteristic Curve) is the metric of choice for the evaluation. It measures the capability of the predictions to order same-class vs. inter-class edges. A perfect predictor (Ω^*) would be one where any inter-class edge would have a lower score than any same-class edge: $\Omega^*(i,j) > \Omega^*(k,l)$ if $\Upsilon(i,j) = 1$ and $\Upsilon(k,l) = 0$, with a ROC-AUC of 1. A predictor that gives the same score for every pair would have a ROC-AUC of 0.5 - which will be the approximate metric for a random predictor. And predictors that perform worse than that would have ROC-AUCs between 0 and 0.5.

The ROC-AUC also permits us to compare the performance of the predictors across datasets, since it is not sensitive to class imbalance, leaving us with the possibility of assessing the absolute and comparative power of our predictors in different settings.

Chapter 5

Results and Discussion

In this chapter, are presented the results of the discussed experimental setup. The results are presented to answer the following questions:

- Do GNNs increase the predictive power of structural embeddings?
- Can positional embeddings have node concordance predictive power?
- Is there an advantage in tackling node concordance directly?
- What are the strengths and weaknesses of each framework?

5.1 Do GNNs increase the predictive power of structural embeddings?

For this investigation, are shown in detail the results for one dataset, the `ogbn-arxiv` graph.

The structural embeddings have a natural baseline to the GNN approach, a model that considers solely the features of each pair of nodes.

As mentioned in the experimental design, we train an MLP using the combined features of the node pair (see equation 3.5) varying the number of layers from 1 to 5, with a fixed number of hidden features in each layer, 128, the same as the number of input features - the dimension of the averaged embedding of each word in the title and abstract of each node.

Figure 5.1 shows the evaluation metrics of the 3-layer MLP model along the training epochs. The model is run 30 times along 5000 epochs, and the evaluation metrics are computed every 100 epochs. The figure shows the average and standard deviation of the metrics - loss (binary cross entropy) and ROC-AUC - for each epoch of these runs.

We can see that the loss and ROC-AUC in the train set are monotonically decreasing and increasing, respectively. The loss on the validation and test sets have minimums around epoch 1000 and sharply rise afterward. Although less clear, their ROC-AUCs have maximums around the same period but decrease with much lower intensity. We can also see that the validation metrics are better than the test metrics reflecting the nature of the dataset splits, the more recent edges in the test set are more distinct from the train edges and, thus, are exposed to greater performance decay.

Figure 5.2 shows a comparison of the validation ROC-AUC - the metric in which we are most interested - performance along epochs for the different numbers of MLP layers. For visualization purposes, are only shown the first 2000 epochs.

We can see that the ROC-AUC increases faster as MLP layers are added, but the maximum of the metric appears to be stable after the addition of the third layer, what secures us that a sufficiently complex baseline will be used for comparison against the GNN functions.

To study the performance along epochs of the GNN functions, it is shown the metrics of a 2-layer GraphSage function ($\Omega_s^{\text{2-GraphSage}}$) during training in figure 5.3. The function consists of two

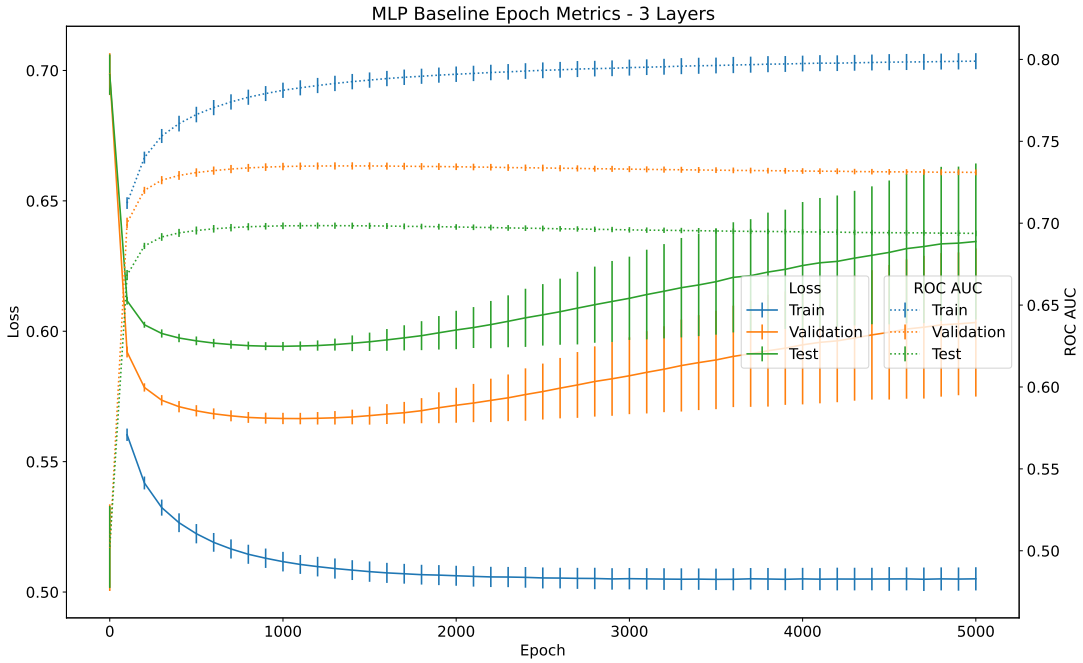

Figure 5.1: *Average loss and ROC-AUC of the $\Omega_s^{3\text{-}MLP}$ function for 30 runs along 5000 epochs. Continuous lines connect the binary cross entropy data points and the dotted lines connect ROC-AUC data points. The standard deviation of the metrics along the runs is shown in the error bars.*

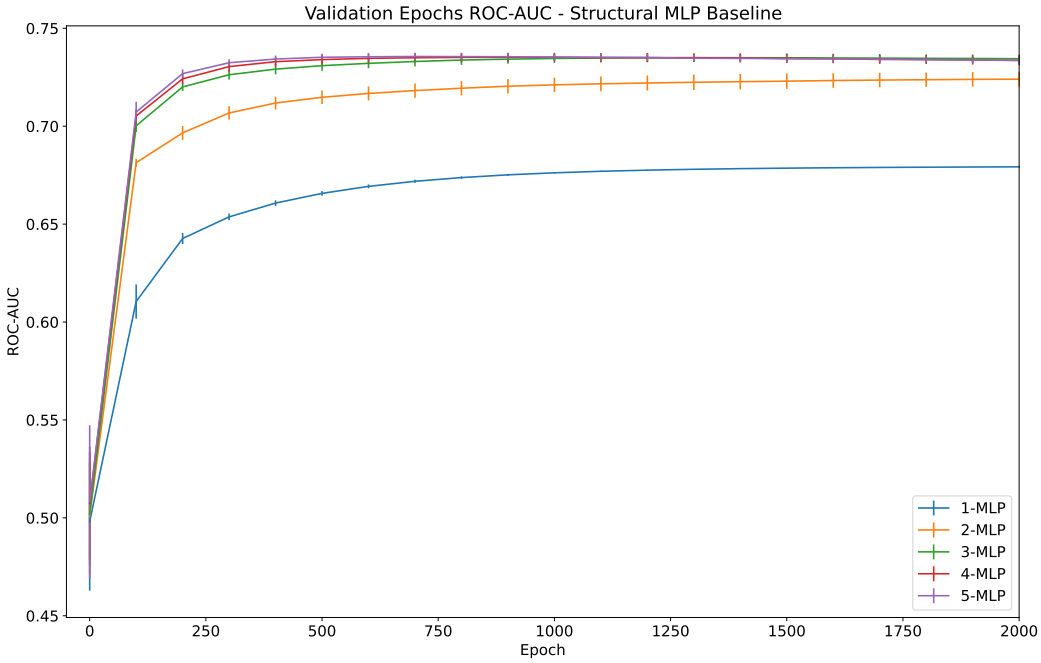

Figure 5.2: *Average validation ROC-AUC of the Ω_s^{MLP} functions for 30 runs along 2000 epochs. Each line contains the ROC-AUC data points for a layer number. The standard deviation of the metric along the runs is shown in the error bars.*

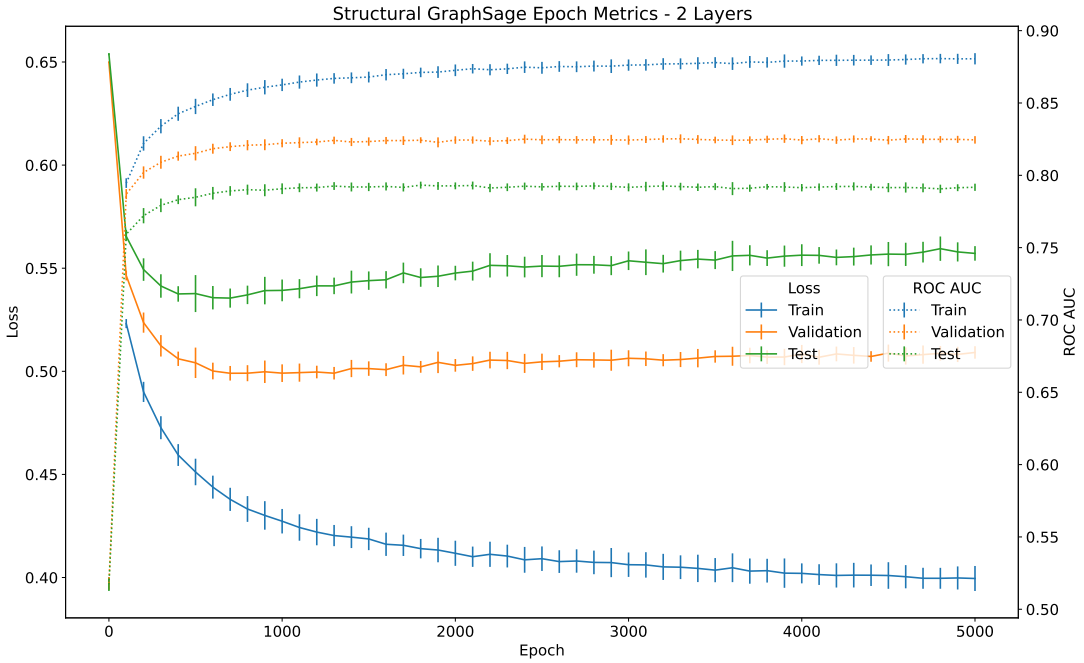

Figure 5.3: *Average loss and ROC-AUC of the $\Omega_s^{2\text{-}GraphSage}$ function for 30 runs along 5000 epochs. Continuous lines connect the binary cross entropy data points and the dotted lines connect ROC-AUC data points. The standard deviation of the metrics along the runs is shown in the error bars.*

consecutive GraphSage convolutional layers, each outputting 128 features, calculated for each node, whose outputs are used to compute concordance predictions according to equation 3.4.

Similar behavior to the baseline model can be seen, with a less sharp rise in the BCE loss after its minimum. What is most relevant, in the author's view, is the difference between the maximum (and the plateau) of the validation and test ROC-AUCs with those in the baseline model, showing the performance gain in utilizing the neighbors (in the case, the 2-neighborhood) features to predict node concordance.

5.2 Can positional embeddings have node concordance predictive power?

The question for positional embeddings is introduced with more skepticism since the predictor is trained without receiving the concordance label at any point. Furthermore, it does not receive any of the node features, which is the main source of information to predict the class of the nodes for the node classification task for what these datasets are typically used. The unsupervised design and underutilization of available information justify this skepticism, which will be overcome with the following results.

As discussed in the experimental design section, there are diverging data splits between training and evaluation, as the raw node embeddings must be trained for every node (including the validation/test nodes given by the dataset data split). One percent of the graph edges are randomly separated for validation and another one percent for the test, these separations are useful for measuring the link predictive capacity of our function, rather than the node concordance predictive power. We keep the same data split as for the structural embeddings for evaluating the latter. Therefore, after each epoch, the loss in the random splits and the ROC-AUC in the evaluation edge splits (which come from the datasets node splits) are measured.

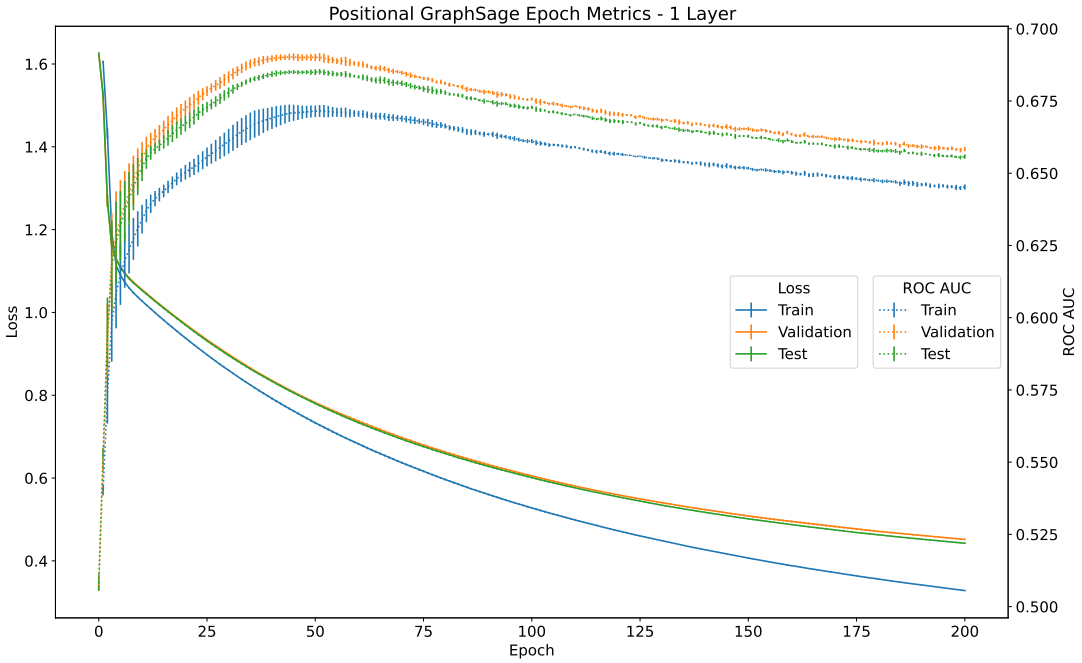

Figure 5.4: *Average loss and ROC-AUC of the $\Omega_p^{1\text{-}GraphSage}$ function for 30 runs along 200 epochs. Continuous lines connect the link-prediction loss data points and the dotted lines connect node concordance ROC-AUC data points. The standard deviation of the metrics along the runs is shown in the error bars. Loss and ROC-AUC are computed in different splits.*

5.2.1 Positional Embeddings - GNNs

Figure 5.4 shows the loss and ROC-AUC of the $\Omega_p^{1\text{-GraphSage}}$ function for the respective data splits. One GraphSage convolutional layer, with raw embeddings (E) of dimension 50, as input generates each node embeddings, also of dimension 50, which are used to compute the edge existence probability as described in equation 3.8, that is shown to be discriminant of node concordance.

It can be seen that while the performance for link prediction increases (the loss decreases) monotonically during the trained epochs - even on the unseen validation and test edge sets -, while the node concordance predictive power of the function peaks around epoch 50.

This result puts evidence to an obvious conclusion, that we may not benefit from the best link-prediction function, but from the function that better learns the homophilic graph structure. It is obvious because the best link predictive function would be able to predict exactly the graph edges, and what we are trying to do here is to distinguish the obvious (homophilic) edges from the less obvious (heterophilic) ones.

Here is further show the assumption of this approach: the homophily characteristic is a generative factor of the graph, edges are created from this characteristic, but other factors also create edges. The best link predictor would learn all the factors in the edge creation mechanism, but during training, first are learned the predominant factors of edge generation, and if homophily is a major factor in the graph generation, we can benefit from it using the link predictor. This result is illustrative of the assumptions and also shows a limitation of this approach: if homophily does not play a (central) role in the graph generation, this framework would not work.

5.2.2 Positional Embeddings - Node2Vec

There are, though, other ways to generate positional node embeddings, that instead of capturing the edge creation mechanism, are able to capture node properties regarding the role of a node in

the graph. Node2Vec is a framework that is able to construct embeddings that are informative of a range of characteristics of the node role in the graph. With its biased random walk and context size parameters, it is able to generate embeddings that can be used for identifying network communities (what is similar to link prediction), but also for identifying structural roles in the graph. Both embedding types can be predictive of node concordance but in a heterophilic graph it is expected that only the latter would be predictive of this property.

Although Node2Vec embedding can be informative of structural roles, here the framework is classified as positional because there is no guarantee that isomorphic nodes will have the same embeddings. The biased random walks are generated from the node neighborhood and without sufficiently large context windows, far-apart nodes should have distinct node embeddings.

Here, the Node2Vec embeddings are trained with fixed walk length (20), context size (10), and embedding dimension (50). Each epoch generates one positive and one negative walk starting from each node of the graph and updates the node embeddings from it. The parameters p and q from equation 2.16 are varied in a log space from 0.1 to 10, with 5 samples, resulting in a total of 25 combinations. The data split is the same as for the positional embeddings using GNNs.

Figure 5.5 shows the loss and ROC-AUC of the $\Omega_p^{\text{N2V p=0.1, q=10}}$ function for the respective data splits along 500 epochs. Node2Vec embeddings (E) of dimension 50 are trained and compared as described in equation 3.9. It is computed the same loss as in figure 5.4 for comparison purposes - it could be shown the Node2Vec loss for the train set, which measures the predictive power for the whole context window, instead of for the link prediction, which would result in a monotonic decrease in the loss for the train set, but the loss for validation and test is not defined, and would not be compared to the previous result.

It can be seen that the link prediction predictive power has a peak close to epoch 100, as does the node concordance predictive power. The train loss increases afterward, showing evidence that the Node2Vec is not optimizing the link prediction directly. This also puts evidence to the

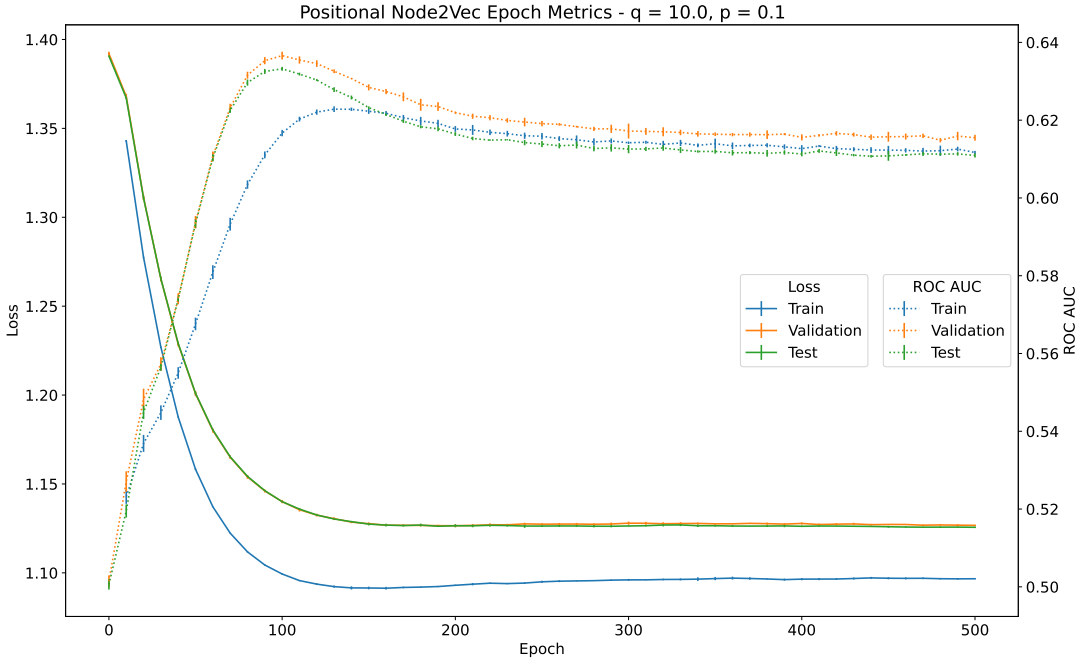

Figure 5.5: *Average loss and ROC-AUC of the $\Omega_p^{N2V\ p=0.1,\ q=10}$ function for 5 runs along 500 epochs. Continuous lines connect the link-prediction loss data points and the dotted lines connect node concordance ROC-AUC data points. The standard deviation of the metrics along the runs is shown in the error bars. Loss and ROC-AUC are computed in different splits.*

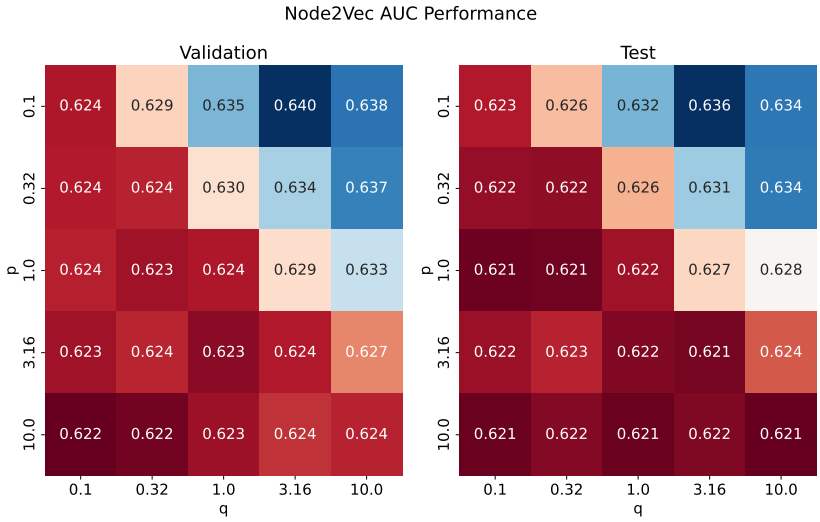

Figure 5.6: *Best ROC-AUC metrics of the Ω_p^{N2V} for validation and test sets of* `ogbn-arxiv`.

possibility that Node2Vec can optimize its expressive abilities to something that is not related to link prediction, which is particularly interesting here, since the goal is to predict node concordance. Nonetheless, for this graph, the GNN approach performs better than Node2Vec.

Figure 5.6 shows a heatmap of the concordance predictive power (measured in ROC-AUC) varying the p and q parameters of the Ω_p^{N2V} function. It can be seen that the function has a better performance with a smaller return parameter (p) and a higher in-out parameter (q). A smaller return parameter means a higher probability of immediately revisiting a node and a higher in-out parameter means a higher probability of visiting a node that is a neighbor of the previous node in the walk. Both of them mean a higher probability that the walk stays within a community, and close nodes will have similar embeddings. Therefore, the Node2Vec that best predicts concordance, in this case, is a Node2Vec that predicts communities well, what is intuitive in a homophilic graph.

5.3 Is there an advantage in tackling node concordance directly?

To investigate if there are performance gains in tackling node concordance directly, instead of modeling node classifiers and comparing their predictions to compute node concordance predictions, we use the `ogbn-arxiv` graph, and the GraphSage model.

The GraphSage model for node classification computes node embeddings in a very similar fashion to the node embeddings computed in the $\Omega_s^{N\text{-GraphSage}}$ functions, with the exception that the last layer must output an array of dimension equal to the number of node classes – in this case 40 –. In the optimization process is computed the negative log-likelihood of the predictions to the actual node classes, which is used to update the model parameters – the weights of the GraphSage layers –. The node concordance predictions are obtained by comparing the node classification predictions of the two nodes of interest, similarly to the Node2Vec embeddings comparison in equation 3.9.

Figure 5.7 shows the node classification accuracy – the evaluation metric proposed by OGB [HFZ+20] for the task in this dataset – and the node concordance ROC-AUC along training epochs. The nodes validation set average accuracy peaks in epoch 3200, while the edges validation node concordance ROC-AUC peaks in epoch 200, what puts evidence to the fact that these tasks are quite different regarding optimal node embeddings.

The figure directly compares with the figure 5.3, but were also tested the models with 1 and 3 GraphSage layers. The result is shown in figure 5.8. It can be seen that the $\Omega_s^{N\text{-GraphSage}}$ outperforms the node classifier for every number of layers. There is an average gain of 0.087 ROC-AUC points, a 12.5% increase.

This result puts evidence to the fact that it is suboptimal to train a more general model – a

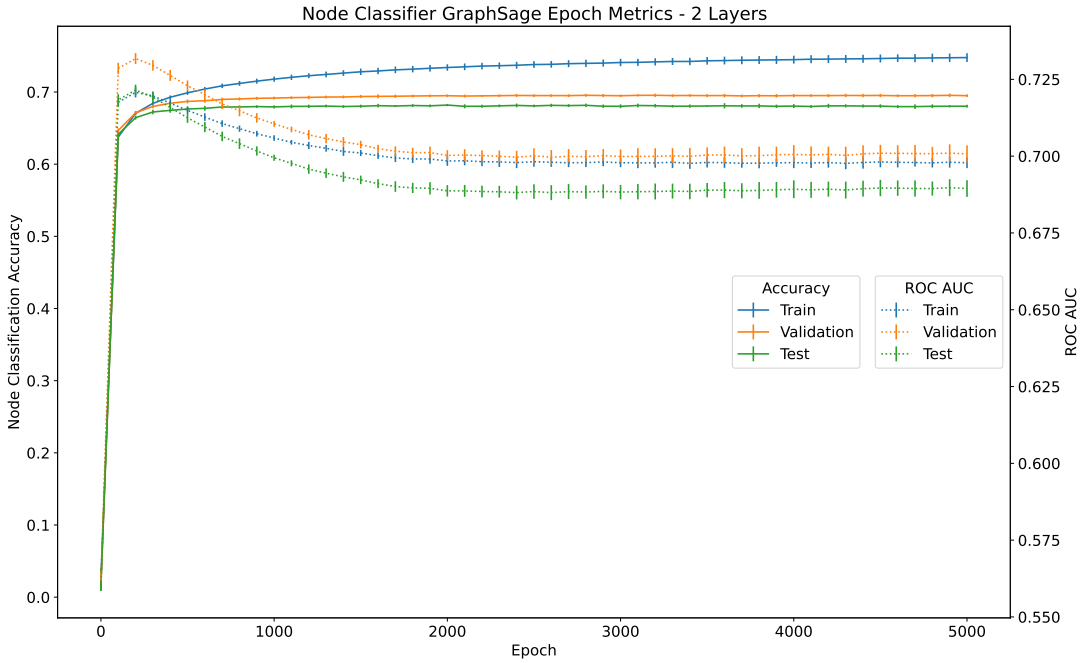

Figure 5.7: *Node classification accuracy and node concordance ROC-AUC of a two-layer GraphSage node classifier used to predict node concordance for 5 runs along 5000 epochs. Continuous lines connect the node classification accuracy data points and the dotted lines connect node concordance ROC-AUC data points. The standard deviation of the metrics along the runs is shown in the error bars.*

node classifier – to obtain node concordance predictions. what is in line with Vapnik's principle that if we have a restricted amount of information to solve a problem, we should solve it directly, never solve a more general problem as an intermediate step [Vap98].

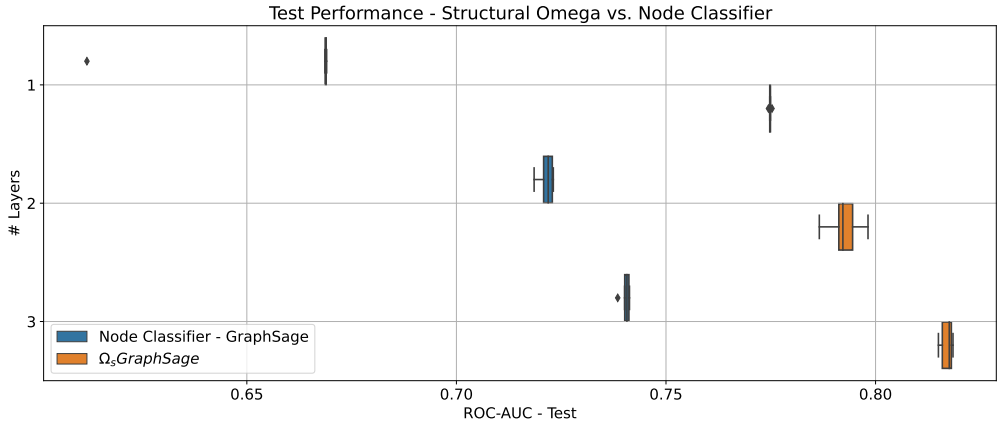

Figure 5.8: *Boxplot comparing the test performance of $\Omega_s^{N\text{-}GraphSage}$ and the GraphSage Node Classifier for 1 to 3 convolutional layers. For each run, it is chosen the best model epoch from the validation performance and displayed the performance on the test set.*

5.4 What are the strengths and weaknesses of each framework?

The previous sections show that there is a gain in predictive power from using GNNs in structural embeddings, and that it is possible to extract features that are informative of node concordance from positional embeddings.

Table 5.1 shows the performance of each Ω function at all the evaluated datasets. For each model run it is chosen the epoch with the greatest validation performance, and with the best model of each run is computed the mean and standard deviation of the ROC-AUC metric.

It is possible to see that:

- Structural embeddings generated by the GNNs outperform the baseline performance in all cases.
- The GAT, GCN, and GraphSage GNNs have similar performance in general, with an exception in the CiteSeer dataset, where GraphSage outperforms the other GNNs.
- The structural approach outperforms the positional in all cases.
- The positional framework only outperforms the structural baseline with $\Omega_p^{GraphSage}$ function in the Cora dataset.

		ROC-AUC		
Dataset	Function	Train	Validation	Test
ogbn-arxiv	$\Omega_p^{\text{GraphSage}}$	0.675 ± 0.003	0.698 ± 0.004	0.690 ± 0.004
	$\Omega_p^{\text{Node2Vec}}$	0.622 ± 0.000	0.638 ± 0.002	0.634 ± 0.001
	Ω_s^{GAT}	0.866 ± 0.002	$\mathbf{0.849 \pm 0.002}$	0.820 ± 0.003
	Ω_s^{GCN}	0.868 ± 0.002	0.847 ± 0.001	$\mathbf{0.823 \pm 0.002}$
	$\Omega_s^{\text{GraphSage}}$	0.890 ± 0.002	$\mathbf{0.849 \pm 0.002}$	0.817 ± 0.001
	Ω_s^{MLP}	0.788 ± 0.004	0.736 ± 0.002	0.700 ± 0.002
CiteSeer	$\Omega_p^{\text{GraphSage}}$	0.582 ± 0.017	0.627 ± 0.013	0.583 ± 0.013
	$\Omega_p^{\text{Node2Vec}}$	0.477 ± 0.031	0.535 ± 0.017	0.483 ± 0.011
	Ω_s^{GAT}	0.924 ± 0.030	0.697 ± 0.010	0.631 ± 0.018
	Ω_s^{GCN}	0.917 ± 0.003	0.708 ± 0.002	0.683 ± 0.002
	$\Omega_s^{\text{GraphSage}}$	1.000 ± 0.000	$\mathbf{0.786 \pm 0.002}$	$\mathbf{0.751 \pm 0.005}$
	Ω_s^{MLP}	0.989 ± 0.006	0.676 ± 0.002	0.663 ± 0.005
Cora	$\Omega_p^{\text{GraphSage}}$	0.713 ± 0.008	0.791 ± 0.005	0.730 ± 0.005
	$\Omega_p^{\text{Node2Vec}}$	0.617 ± 0.011	0.635 ± 0.012	0.596 ± 0.011
	Ω_s^{GAT}	0.945 ± 0.009	0.853 ± 0.012	0.789 ± 0.015
	Ω_s^{GCN}	0.995 ± 0.000	$\mathbf{0.878 \pm 0.002}$	0.819 ± 0.002
	$\Omega_s^{\text{GraphSage}}$	1.000 ± 0.000	0.871 ± 0.003	$\mathbf{0.822 \pm 0.002}$
	Ω_s^{MLP}	0.926 ± 0.002	0.711 ± 0.001	0.685 ± 0.000
Pubmed	$\Omega_p^{\text{GraphSage}}$	0.543 ± 0.018	0.535 ± 0.013	0.560 ± 0.025
	$\Omega_p^{\text{Node2Vec}}$	0.514 ± 0.012	0.547 ± 0.025	0.526 ± 0.029
	Ω_s^{GAT}	0.971 ± 0.005	0.863 ± 0.013	0.852 ± 0.013
	Ω_s^{GCN}	0.983 ± 0.004	0.883 ± 0.011	0.874 ± 0.009
	$\Omega_s^{\text{GraphSage}}$	0.999 ± 0.001	$\mathbf{0.903 \pm 0.010}$	$\mathbf{0.881 \pm 0.021}$
	Ω_s^{MLP}	0.870 ± 0.017	0.757 ± 0.004	0.754 ± 0.003

Table 5.1: *Summary of ROC-AUC metric for multiple runs across datasets. At each run are chosen the models with the best evaluation metrics. The averages and standard deviations are computed for each model and displayed in the table.*

These results appear to disqualify the positional approach in comparison to the structural one. But the positional approach has its strengths: Since it is an unsupervised approach, using the labels only for model selection, it should have less performance decay in datasets with smaller proportions of labeled train nodes. Also, since the approaches are significantly different - one utilizes node features, the other only uses the graph structure, and the target functions are distinct (although correlated in homophilic graphs) - the two approaches could be complementary. Both these suspicions are investigated in the following subsections.

5.4.1 Is the positional approach less sensitive to the number of labeled nodes?

To investigate if the performance decay of the structural approach is greater than the one of the positional approach, we utilize the $\Omega_p^{1-\text{GraphSage}}$ and the $\Omega_s^{1-\text{GraphSage}}$ – the positional and structural functions with one GraphSage convolutional layer – and the `ogbn-arxiv` dataset. The reason for that is we want comparable models – and hence we use the same GNNs –, we want simpler models, with less variance, and more probability of achieving generalization, since we will train some models with few training edges – and hence we use only one convolutional layer – and we want to be able to generate realistic data splits – and the `ogbn-arxiv` dataset provides this possibility, as will be discussed next.

The `ogbn-arxiv` dataset provides the year of publication of each node (paper), and its node set is split into train, validation, and test sets based on the year of publication: the train nodes are the nodes published before the threshold year t, the validation nodes are the ones published in year t, and the test nodes after the year t. In particular, the dataset uses $t = 2018$ for its proposed node split. We can vary this year t, generating multiple node splits, with varying train, validation, and test node sets size, and with these node splits, we can generate our edge splits, as we did before. Figure 5.9 shows the sizes of the node and edge sets size for each threshold year (t). The number of training edges (nodes) goes from 9,215 (6,911) for $t = 2009$, to 622,466 (120,740) for $t = 2019$.

The positional framework does not take into account the labels in its training process, only in the model selection, therefore we run the model once, storing the predictions of each training epoch. The union of train and validation edge sets for each threshold year (t) is used to choose the model, based on the ROC-AUC computed for this union set, and the model is evaluated in the test set. Meanwhile, the structural framework needs to be trained for each t, and the model epoch is chosen using the ROC-AUC computed only in the validation set.

Figure 5.10 shows the test performance of the two functions. We can see that the $\Omega_p^{1-\text{GraphSage}}$ performance is almost constant along the various t, while the $\Omega_s^{1-\text{GraphSage}}$ increases a lot as training edges are added to the model. The positional function performs better for $t \leq 2015$, which puts evidence to the strength of the positional framework in settings with few labeled nodes.

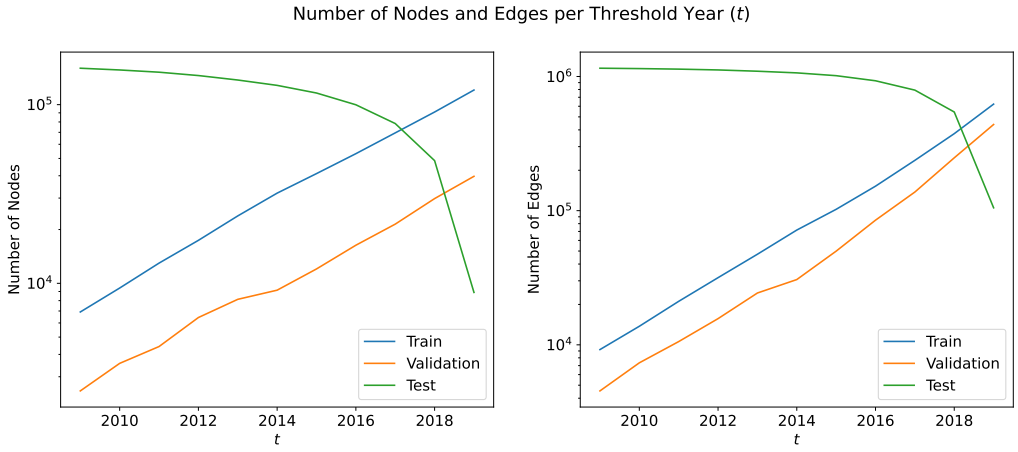

Figure 5.9: *Number of nodes and edges for each threshold year (t) in the train, validation, and test sets for* $t \in [2009, 2019]$. *The number of nodes and edges are shown in a logarithmic scale for visualization purposes.*

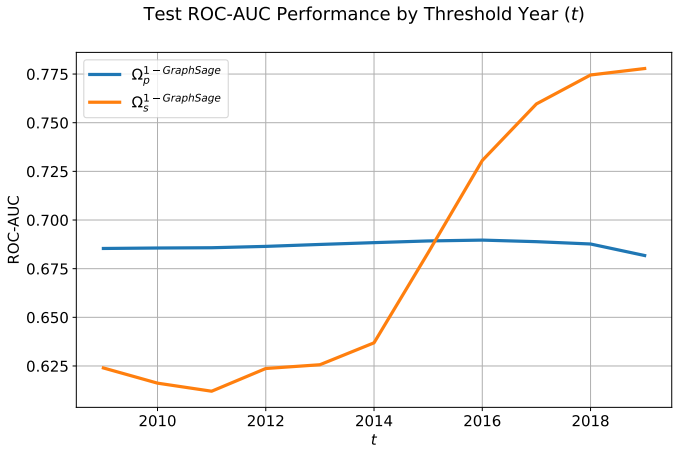

Figure 5.10: *ROC-AUC scores in the test set for various threshold years (t) of the* $\Omega_p^{1-GraphSage}$ *– positional – and the* $\Omega_s^{1-GraphSage}$ *– structural – functions.*

5.4.2 Can positional and structural embeddings be used in combination?

To investigate if positional and structural embeddings can be used in combination we need to understand if one adds information to the other. Figure 5.11 shows the correlations between model predictions and ensemble performance between $\Omega_p^{\text{GraphSage}}$ and Ω_s^{GCN} functions. The model with the best validation ROC-AUC score (with the default 2018 threshold year) is selected, and the predictions of this model for the test edges are computed, and finally are calculated the correlation matrix and a simple ensemble (varying the weights of each model prediction) performance.

It can be seen that the two positional approaches have the lowest correlations with the Ω_s^{MLP} function predictions and that the GNN structural functions all have high (> 0.75) correlations between them, and low (< 0.45) between them and the positional functions. This is expected since the features and target function are different among the approaches. This indicates that the positional approach can be complementary to the structural one, different node concordance predictive factors possibly are being extracted from each approach. Another way to investigate if this hypothesis is true is to check if we can enhance the predictive power of the structural approach by creating a new Ω function that is an ensemble between a structural and a positional function. On the right of figure 5.11, it shows the performance of a simple ensembled function, a weighted average

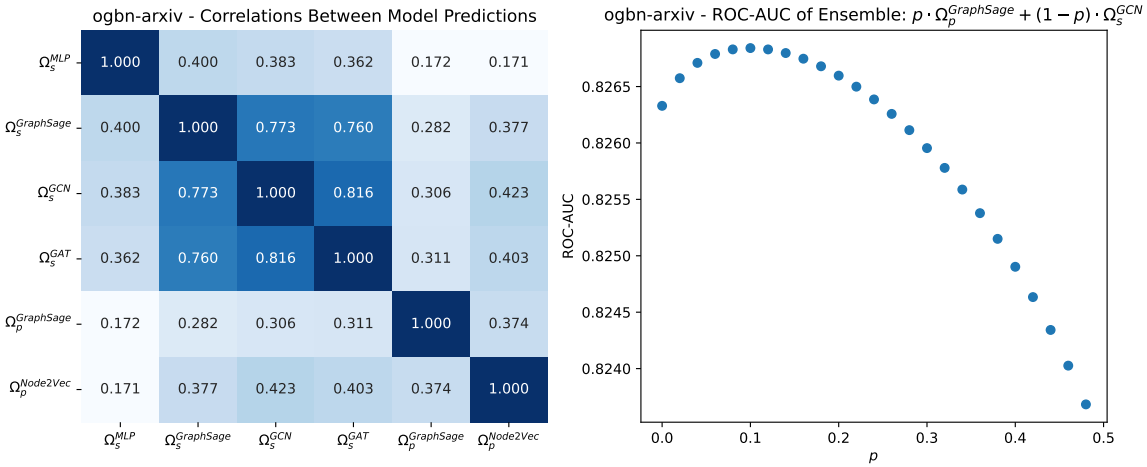

Figure 5.11: *On the left, it is shown the correlation matrix (computed in the test set of edges) between the best model (selected by validation ROC-AUC) of each architecture/framework. On the right, it is shown the test performance of a simple ensemble between the* $\Omega_p^{GraphSage}$ *and* Ω_s^{GCN} *functions.*

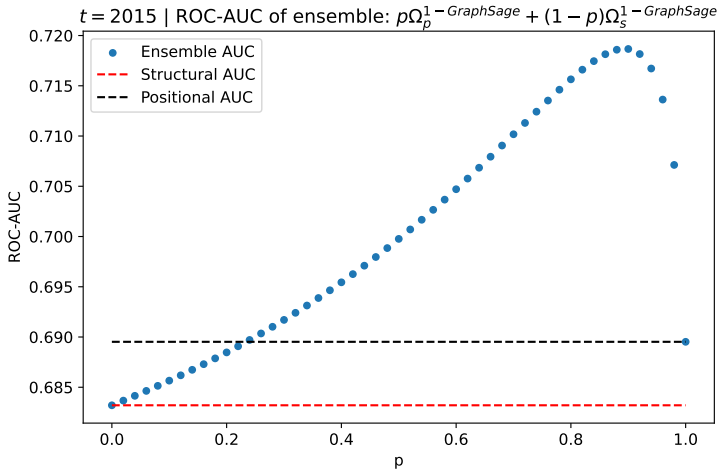

Figure 5.12: *The test performance of a simple ensemble between the* $\Omega_p^{1-GraphSage}$ *and* $\Omega_s^{1-GraphSage}$ *functions in the* `ogbn-arxiv` *dataset with threshold year* $t = 2015$.

between the predictions of the two functions. It can be seen that despite the great performance difference between the structural and positional approaches - which have test ROC-AUC scores of 0.8263 and 0.6831, respectively - for weights $p \in [0.02, 0.24]$, there is an enhancement in the node concordance predictive power when compared to the plain Ω_s^{GCN} predictions.

A more fair ensemble study can be done using structural and positional functions that have more similar performance. The previous subsection (5.4.1), presents a good candidate for this study: The dataset with threshold year $t = 2015$, the structural function $\Omega_s^{1-\text{GraphSage}}$, and the positional function $\Omega_p^{1-\text{GraphSage}}$. The two functions generate predictions that have a low correlation of 0.317 in the test set – which is in line with the correlations found with the default threshold year $t = 2018$ and the best models selected across more options for the number of layers in figure 5.11 –, and have very similar concordance predictive power, with a ROC-AUC of 0.683 for the structural, and 0.690 for the positional. The same simple ensemble methodology is displayed in figure 5.12.

We can see that the range of outperformance of the ensemble against the best model (in this case the $\Omega_p^{1-\text{GraphSage}}$) is enlarged – the ensemble outperforms for $p \in [0.24, 0.98]$ –, and the peak enhancement is greater – 0.03 ROC-AUC points against 0.0005 of the previous study.

These results put evidence to the complementarity of the two approaches, which are learning considerably distinct predictive factors from the graph.

Chapter 6

Conclusions and Future Work

This work introduces a novel link property prediction task on graphs and explores predictors to the local manifestation of homophily, the node concordance. Homophily is a widespread characteristic of many real-world graphs and is a building block for many graph algorithms, from Label Propagation (LPA) to Graph Neural Networks (GNNs). Predicting its local manifestation can be an end in itself, in cases where we have only partially labeled edges (or nodes if construct edge labels from the node ones) or can be a preprocessing step, in rewiring the graph or attributing weights to its edges.

This task is related to node classification – when the nodes have labels, and the concordance label is obtained from it – and can be seen as a more specific subtask of it, this work shows that there is performance gain in tackling directly this task, i.e. using node classifications as a means to predict node concordance leads to suboptimal performance. The node concordance labels can be generated without the necessity of node labels and the results shown here encourage a reader that may want to assess node concordance to tackle it directly, or if the reader needs to label a dataset for this task, to label the edges, not the nodes, which may be easier in some cases, leaving out the necessity of creating a taxonomy of node classes. This is the case, for example, in many clustering tasks, where we want to know which nodes are of the same cluster, without necessarily knowing what the clusters mean.

Here are explored two frameworks to predict node concordance, one structural and one positional. The structural and supervised framework generally performs better but is more dependent on the number of training points. The positional and unsupervised framework can be used to extract reasonable predictive power and is particularly recommended in cases in which node features are not present or not informative of the node concordance, or cases in which the representativeness of labeled nodes is rather small.

Further research on this topic includes testing the frameworks in other graph benchmarks in general, and particularly in heterophilic ones, to assess the strengths of the various models tested here in heterophilic settings, along with other GNNs designed to tackle this kind of graph. Also, to explore direct applications of the node concordance predictions, such as its impact on network dynamics and recommender systems, and indirect applications, such as a preprocessing step of graphs for node or graph classification, weighing edges or rewiring the graph to become more (or less) homophilic.

Bibliography

[BUGD+13] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In: *Advances in Neural Information Processing Systems*, 2013. 3, 10, 11

[CMX18] Jie Chen, Tengfei Ma and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. 1 2018. 18

[CXH+20] Hao Chen, Yue Xu, Feiran Huang, Zengde Deng, Wenbing Huang, Senzhang Wang, Peng He and Zhoujun Li. Label-Aware Graph Convolutional Networks. *International Conference on Information and Knowledge Management, Proceedings*, pages 1977–1980, 2020. 12

[GL16] Aditya Grover and Jure Leskovec. Node2Vec. pages 855–864, 2016. 3, 10, 11

[HFZ+20] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems*, 2020-December(NeurIPS):1–34, 2020. 10, 17, 24

[HHS+20] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim and Austin R. Benson. Combining Label Propagation and Simple Models Out-performs Graph Neural Networks. (Figure 1), 2020. 12

[HYL17] William L. Hamilton, Rex Ying and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):1025–1035, 2017. 3, 7

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–14, 2017. 3, 7

[LAQ+21] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang and Qing He. Pick and choose: A gnn-based imbalanced learning approach for fraud detection. pages 3168–3177. Association for Computing Machinery, Inc, 4 2021. 3

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 3, 7

[LXTG20] Guohao Li, Chenxin Xiong, Ali Thabet and Bernard Ghanem. DeeperGCN: All You Need to Train Deeper GCNs. 2020. 3

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, pages 1–12, 2013. 10, 11

[MLJ01] McPherson Miller, Smith-Lovin Lynn and M Cook James. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27:415–444, 2001. 1

[PARS14] Bryan Perozzi, Rami Al-Rfou and Steven Skiena. DeepWalk. pages 701–710, 2014. 3, 10, 11

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. 1

[SNB+08] Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29:93–106, 2008. 1, 17

[SR20] Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. 2020. 9

[Vap98] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998. 1, 25

[VCC+17] Petar Velicković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò and Yoshua Bengio. Graph attention networks. *arXiv*, pages 1–12, 2017. 2, 3, 9

[Vla13] Vapnik Vladimir. Transductive Inference and Semi-Supervised Learning. *Semi-Supervised Learning*, pages 452–472, 2013. 12

[WL20] Hongwei Wang and Jure Leskovec. Unifying Graph Convolutional Neural Networks and Label Propagation. 2020. 12

[WLX+20] Yongji Wu, Defu Lian, Yiheng Xu, Le Wu and Enhong Chen. Graph convolutional networks with markov random field reasoning for social spammer detection, 2020. 1

[WSH+20] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh Han Wu, Yuxiao Dong and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1:396–413, 2 2020. 1

[WZdS+19] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza, Christopher Fifty, Tao Yu and Kilian Q. Weinberger. Simplifying graph convolutional networks. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:11884–11894, 2019. 3

[XSS+21] Bingbing Xu, Huawei Shen, Bingjie Sun, Rong An, Qi Cao and Xueqi Cheng. Towards consumer loan fraud detection: Graph neural networks with role-constrained conditional random field, 2021. 1

[Zhu05] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. 2005. 12