Hyperlink prediction with Modular Directed Hypergraph Neural Networks

Wientjes, Manon

Award date:
2021

Department of Mathematics and Computer Science
Software Engineering and Technology Group

Hyperlink prediction with Modular Directed Hypergraph Neural Networks

Master's Thesis

Manon Wientjes

Supervisors:
Dr. Yaping Luo (ING and TU/e)
Dr. Yulong Pei (TU/e)
Ir. Kevin van der Vlist (ING)

Eindhoven, July 2021

Abstract

Link prediction is a graph learning problem in which unobserved links are predicted based on the observed features and structure of the graph. In many applications, modelling of higher-order relationships among entities is needed. Link prediction on hypergraphs (hyperlink prediction) is the problem of predicting missing higher-order relationships in a hypergraph. Moreover, modelling directions between these relationships is also useful in many practical applications.

Hyperlink prediction as a field of research is still emerging. Meanwhile, directed hyperlink prediction is underexplored. A model that uses the directions as well as the hypergraph structure as information to make predictions has not been proposed yet. In this thesis, we propose Modular Directed Hypergraph Neural Network (MDHNN) as a model that can be used for directed hyperlink prediction. MDHNN uses the direction of the edges and the structure of the hypergraph as information to make predictions. Besides, we carry out an extensive comparison of existing (directed) hyperlink prediction methods based on three different evaluation metrics. The experiments run on real-world data show that MDHNN has better performance than most baseline models and on some dataset even outperforms all existing baseline methods. The results of the experiments show that using the structure of the graph is not well exploited in current methods. Besides, using the direction of the edges to make predictions can increase the performance by 5 percent.

Acknowledgments

I would like to thank my supervisor Yaping Luo for the supervision of my thesis and giving great feedback on conducting research and the report.
My grateful thanks are also extended to Yulong Pei for giving useful suggestions and insights in deep and graph learning.
I also would like to thank Kevin van der Vlist for giving me such an interesting problem and valuable practical feedback.
Finally, special thanks to my bunny Kiko for not destroying my power cable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As a major financial services provider, ING leverages a large number of IT systems. To interact with those systems, a sizeable portion of software is part of the class middleware components. Whilst these systems certainly apply specialized features that are not necessarily generalizable, there is also a major part that is more generic and reusable. Examples of this are data mapping and service orchestration. To achieve lower maintenance costs and improve the time to market, a project within ING proposed to use model-driven engineering techniques to automate these generic parts of software systems, whilst still allowing for manual work on the more specific parts in a software system. In order to automate this, large domain models need to be available.

Such a domain model can be represented as a directed (for now) unweighted hypergraph. In one such hypergraph, defined by $H = (X, E)$, each vertex $x_i \in X$ denotes type information and each hyperedge, also called hyperlink, $E_i \in E$ describes an operation that is possible with the types on the edge. This makes the hypergraph a reflection of a (partial) business domain. Any business has separate subdomains that are likely to be barely related or not related at all to other subdomains. This shows in the hypergraph by having multiple weakly connected components/clusters. In ING's context, business is the finance domain and two examples of subdomains are payments and mortgages.
We use this hypergraph to automatically discover paths given a desired start and end

requirement. If such a path exists, we can generate a middleware component. We are interested in the case where we cannot find a path, for example because there is no connection between two weakly connected components, since this poses a problem: software generation becomes impossible. Hence, we want to introduce hyperedges into the graph that would be sufficient to provide us with new paths and thus increase the connectivity of the graph. Such a hyperedge can only be introduced if it is accepted by a domain expert because it must comply with the business rules and logic. This problem of predicting missing hyperlinks is also known as hyperlink prediction.

Link prediction in simple graphs is a problem in which new links between vertices are predicted based on the observed features and structure of the graph. However, in many applications, there is need to model higher-order relationships among entities. Link prediction on hypergraphs (hyperlink prediction) is the problem of predicting missing higher-order relationships in a hypergraph. Hyperlink prediction has been especially popular in social networks [37, 5, 51]. Besides higher-order relationships, modelling directions between these relationships is also useful in many practical applications. However, hyperlink prediction as a field of research is still emerging. Moreover, directed hyperlink prediction is under explored. A model that uses the directions as well as the hypergraph structure as information to make predictions has not been proposed yet.

To address all of these challenges, in this thesis Modular Directed Hypergraph Neural Network (MDHNN) for directed hyperlink prediction is proposed and the applicability of directed hyperlink prediction in the field of software generation is tested. The output of MDHNN reflects the probability that the edge will be accepted by a domain expert. MDHNN can take the information in the vertices (and therefore implicit business concepts) into account. This drives the following research question: *How can we design an algorithm for directed hyperlink prediction?*

The sub-questions that are supporting this research question are:

1. *What are the state-of-the-art (hyper)link prediction models and can they be used on ING's use case?*
2. *How can the direction of the edges be captured in a hyperlink prediction model?*
3. *Which datasets can be used for evaluation?*
4. *How does the performance of the proposed model compare with the different baseline models?*

In the remaining part of the thesis, we will first discuss the related work in Chapter 2. Thereafter, in Chapter 3 the preliminaries are introduced and MDHNN is proposed. Before conducting experiments as explained in Chapter 5, we have to select evaluation datasets. The selection of the evaluation datasets have been elaborated on in Chapter 4. Finally, in Chapter 6 multiple baseline models are compared and performance of MDHNN is given. The final conclusions and suggestions for future work are given in Chapter 7.

Chapter 2

Related work

Link prediction is a domain in machine learning and can be performed in simple graphs or hypergraphs. In this chapter, we will discuss four type of related work in details: graph learning, graph neural networks, hypergraph learning and hypergraph neural networks. The first exploration of learning over graphs have been done over simple graphs. Therefore, we first discuss related work on graph learning and especially graph neural networks. Graphs neural networks are the basis for hypergraph neural networks. After this, we discuss the related work on hypergraph learning of which hypergraph neural networks are a part.

2.1 Graph learning

Graphs are widely used as a representation of the network structure of connected data. Over recent years, graph learning has gained attention from researchers and practitioners. Graph learning is used to perform many tasks, such as clustering, classification and link prediction. Here, graph learning refers to applying machine learning on graphs.

A (supervised) machine learning algorithm requires a set of informative features as input. Hence, to make predictions on either nodes or edges, a feature vector representations is needed for the nodes and edges, respectively. Such feature vectors can be

created manually using expert knowledge. However, those domain-specific features often do not generalize between different prediction tasks.

Recent research in the field of representation learning has led to a less work-intensive way of generating feature vectors: using machine learning to solve an optimization problem [7]. One such feature vectors, also known as embeddings, could be learned in an supervised or unsupervised way. In the supervised task, the embeddings can be learned such that performance of a downstream prediction task is optimized, e.g. a pipeline. This will have good predictive accuracy as the embeddings are optimal for the specific task, but at the cost of computational efficiency as the number of parameters that need to be estimated will grow. On the other hand, if the embeddings are learned in an unsupervised manner, the objective function will be defined independent of the downstream task. This leads to more computational efficiency and results in task-independent embeddings.

Network embedding methods aim to obtain a low-dimensional representation of nodes in a network. In this low-dimension space the structural and inherent properties of the network are preserved such that the nodes from the network can be represented by the low-dimensional vectors.

There are several methods to generate such node embeddings. Early graph learning approaches use a matrix factorization approach to generate node embeddings [56]. More recently, node embeddings are generated by using a random walk procedure. One such random walk procedure is Deepwalk [46]. This method is inspired by skip-gram [41]. skip-gram learns context-based embeddings for tokens given sentences of a corpus of text. The context of each token consists of the tokens in a window of size c around the token. Deepwalk uses skip-gram, but treats random walks on a graph as sentences and vertices as tokens. The context is defined for each node of the graph by generating a neighborhood as a set of nodes within a window of size k in each random walk. Grover and Leskovec [23] proposed node2vec for generating node embeddings using a random walk. node2vec is similar to Deepwalk, only the definition of the context is different. The context is now defined for each node of the

graph by generating a neighborhood by performing a second-order random walk. This second-order random walk is a biased random walk which balances the breadth-first and depth-first exploration in the network.

The matrix factorization and random walk based methods are not suitable for directed hyperlink prediction. As later discussed in Section 5.2, both methods have their limitations and therefore cannot be extended to directed hyperlink prediction.

2.2 Graph neural networks

The previous two methods to learn node embeddings, i.e. matrix factorization and random walk based methods, are unsupervised. Another method is to use deep learning, which can be both unsupervised and supervised methods. Recently, the generalization of deep learning to graphs has shown good performance across different domains, such as social networks [24, 47], recommendation systems [60], knowledge graphs [55], chemistry [20], etc. Especially, there is a growing interest in graph neural networks because they can efficiently and automatically extract relevant features from a graph. The first notions of graph neural networks were introduced by Gori et al. [22] and Scarselli et al. [49]. They apply recurrent neural networks to deal with graph structures.

Some current state-of-the-art methods are: Graph Convolutional Networks (GCNs) [34], Graph Sample and Aggregate (GraphSAGE) [25], Graph Attention Networks (GATs) [54] and Variational Graph Auto-Encoders (VGAEs) [32]. GCN is a method which is based on Convolution Neural Networks (CNNs) [3]. GCNs operate CCNs directly on graphs. CNNs have proven to be extremely powerful on grid-like structures in various research domains, such as image classification/segmentation [28], speech recognition [1] etc. However, because of the strict assumption that the input data should have regular and grid-like structure, they could not directly be applied on graphs. This problem has being overcome by the introduction of GCNs.

GCNs repeatedly aggregate feature vectors from neighbors to learn node embeddings. Those embeddings encode both local graph structure and node features. Kipf and Welling [34] have demonstrated that GCNs are superior over baseline models by a significant margin.

GraphSAGE is an extension to GCNs. In GraphSAGE, a function is learned that generates embeddings by sampling and aggregating features from a node's local neighborhood. The sampling procedure makes batching possible. GAT is also an extension to GCNs. The extra component of GAT is that nodes are able to attend over their neighborhoods' features. This is done by learning attention coefficients for each node in a neighborhood. Graph convolutions have also been used in VGAE. VGAE uses a GCN encoder and an inner product decoder. After the VGAE has been trained, node embeddings can be generated by applying the encoder to network-structured data. Unlike the graph neural network methods mentioned, VGAE is an unsupervised method.

We refer the interested readers to two comprehensive surveys on the topic of graph neural networks [66, 68].

Graph neural networks can be extended to work for directed hyperlink prediction. However, as introduced in Section 2.4, hypergraph neural networks are generalizations of graph neural networks. Therefore, the application of graph neural networks to hypergraphs is covered by hypergraph neural networks.

2.3 Hypergraph learning

Sometimes, unlike a graph, interactions happen between multiple entities, i.e. higher-order interactions. Hypergraphs, as proposed by Berge [9, 10], were argued by Estrada & Rodriguez-Velazquez [15] as a model to represent complex networks with higher-order relations between sets of entities across a variety of domains. Higher-order relations could also be represented with, for example, simplicial complexes [42, 8] or sets [62, 26, 48]. However, hypergraphs are more naturally represented. Hypergraphs

have been used to model higher-order relations in different fields, such as biology [35], social networks [5], databases [16], etc.

However, despite the fact that hypergraphs are expressive objects, hypergraph learning is lesser studied than simple graph learning. Most existing models assume that the interactions between entities are pairwise. A reason for this is that pairwise relationships remain ubiquitous, because higher-order interactions are often not recorded explicitly in network data. Another reason is that hypergraph learning is more difficult, because the space of higher-order relationships is large. Nonetheless, some approaches for hypergraph learning have been proposed.

Hypergraph learning is first introduced by Zhou et al. [67]. They generalized spectral clustering to hypergraphs and proposed Spectral Hypergraph Clustering (SHC) which aims to minimize the label difference among vertices with stronger connections on hypergraph.
Matrix factorization methods have also been exploited for hypergraphs. However, these methods do not learn node embeddings but are directly applied to hyperlink prediction since a probability is generated for every hyperedge. Coordinated matrix maximization (CMM) [63] predicts hyperlinks in the adjacency space, of size $\mathcal{O}(n^2)$, using an expectation-maximization algorithm. CMM has been extended to Clique-Closure based Coordinated Matrix Minimization [52]. A drawback of these methods is that they rely on the fact that there is a set of candidate hyperedges which is smaller than the set of unobserved hyperedges. Besides, they cannot handle information in the nodes. A more suitable method is hyperedge2vec [50]. hyperedge2vec's Hypergraph Tensor Decomposition performs tensor decompositions over the hypergraph (dual) to generate node and hyperedge embeddings.

Similarly as for graphs, random walks on hypergraphs have been established. hyper2vec [29] is a generalization of node2vec to hypergraphs. Deep Hyperedges [44] performs specialized random walks on the hypergraph.

Similarly as with graph learning, random walk based methods cannot be applied to directed hyperlink prediction. Furthermore, the matrix factorization methods also cannot be applied since they are designed for undirected hypergraphs.

2.4 Hypergraph neural networks

Recently, a lot of methods have been introduced that use deep learning to get d-dimensional embeddings for the nodes in a hypergraph. These embeddings can be aggregated over the hyperedges to predict the missing links.

Hypergraph convolution and attention approaches have been proposed which define a hypergraph Laplacian matrix and perform convolutions on this matrix. Some of those supervised node embedding models are:

- HyperSAGE [4] is a model that learns static embeddings. The model is a generalization of both Graph Convolutional Networks (GCNs) [34] and GraphSAGE [25]. HyperSAGE learns embeddings by aggregating information of intra-edge, i.e. within a hyperedge, neighbors as well as inter-edges, i.e. across hyperedges, neighbors.

- Feng et al. [17] proposed Hypergraph Neural Networks (HGNNs). HGNNs use the clique expansion [2] of a hypergraph over which a GCN is applied. Hence, they extend GCNs for hypergraphs.

- Hypergraph Convolution and Attention [6] also uses the clique expansion to apply GCNs to hypergraphs to define hypergraph convolution. Hypergraph attention learns a dynamic incidence matrix that can better reveal the dynamic connections of hyperedges.

- HyperGCN [57] generalizes GCNs to hypergraphs. HyperGCN performs convolutions over the hypergraph Laplacian with mediators. In contrary to HGNNs

which use a quadratic number of edges for each hyperedge, HyperGCN only uses at most a linear number edges.

- Inspired by GraphSAGE [25] and GATs [54], Zhang et al. [65] proposed HyperSAGNN which is a model to learn node embeddings for hypergraphs. The model learns static (the embedding is the same for every node) and dynamic (the embedding depends on the hyperedge considered) embeddings using multi-head self-attention and combines these to get final node embeddings.

- hyperedge2vec [50] learns hyperedge embeddings by using an autoencoder. The autoencoder is trained to reconstruct the original hyperedge from p noisy hyperedges. The p noisy hyperedges are selected by a random walk over the Hasse lattice.

Above models all work for undirected hypergraphs. Directed hypergraphs have been introduced by Gallo et al. [19]. Only one related work has been found that explicitly solves directed hyperlink prediction. Yadati et al. [58] proposed NHP which is a pipeline that consists of a hyperlink-aware GCN layer which takes the clique expansion of the hypergraph and pass it through a GCN to get vertex embeddings then each directed hyperedge gets a score by the hyperlink and direction scoring layers. However, it should be noted that the method does not use the direction of the edges of the graph in the model but only in the training procedure.

All hypergraph neural networks discussed can be extended for directed hyperlink prediction since they generate node embeddings. In Section 5.2, we will discuss which baseline models are used.

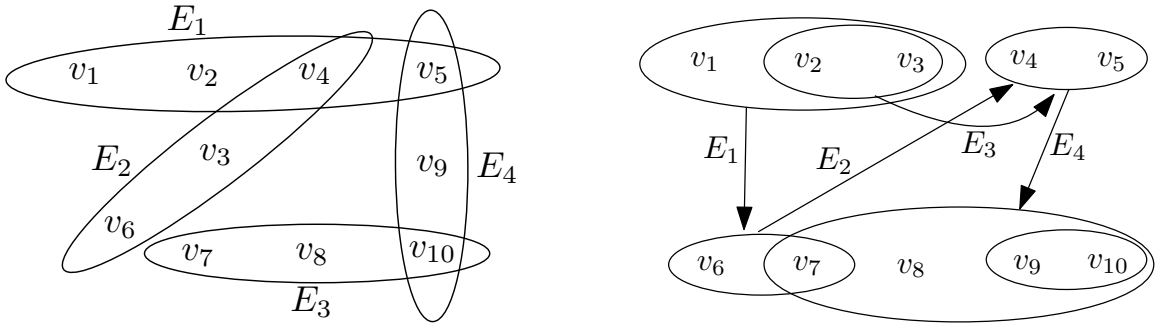
Chapter 3

Methodology

In this chapter, we first present the definitions needed to understand the methodology. Thereafter, the details of methodology will be presented.

3.1 Definitions

Definition 1 (Simple graph)**.** *A simple graph is defined as a pair $G = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the finite set of vertices and $E = \{e_1, e_2, \ldots, e_m\}$ is the finite set of edges. Each edge is denoted by a pair of vertices.*



(a) Undirected hypergraph.

(b) Directed hypergraph.

Figure 3-1: Example of an undirected and directed hypergraph.

Definition 2 (Hypergraph)**.** *A hypergraph [10], an example is shown in Figure 3-1a, is defined as $H = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is the finite set of vertices and*

$E = \{E_1, E_2, \ldots, E_m\}$ is the finite set of hyperedges. Each hyperedge E_i is a subset of V such that

- $E_i \neq \emptyset \qquad \forall i = 1, 2, \ldots, m,$
- $\cup_{i=1}^{m} E_i = V.$

If the cardinality of each hyperedge is two, then you have a simple graph. Hence, hypergraphs are generalizations of simple graphs.

Definition 3 (Directed hypergraph)**.** *A directed hypergraph [19], an example is shown in Figure 3-1b, is a hypergraph with directed hyperedges, also called hyperarcs. Each directed hyperedge is an ordered pair of disjoint subsets of vertices $E_i = (X_i, Y_i)$ such that*

- $T(E_i) = X_i \qquad \forall i = 1, 2, \ldots, m,$ *also called the tail of the hypearc,*
- $H(E_i) = Y_i \qquad \forall i = 1, 2, \ldots, m,$ *also called the head of the hypearc,*
- $\cup_{i=1}^{m} T(E_i) \cup H(E_i) = V.$

It must be noted that $T(E_i)$ and $H(E_i)$ can be empty.

Definition 4 (Static and dynamic networks)**.** *A network is a collection of connected objects and hence is often represented as a graph. We distinguish two types of networks: static and dynamic networks. Dynamic networks change over time, i.e. edges and nodes can be added and removed. Static networks remain unchanged over time.*

Definition 5 (Link prediction on directed hypergraphs)**.** *Link prediction is a binary classification problem where we predict missing links (static network) or links that are likely to occur in the future (dynamic network). Formally, this can be defined as the problem of learning a function f such that*

$$f(E_i) = \begin{cases} 1, & \text{if } E_i \in E \\ 0, & \text{if } E_i \notin E, \end{cases}$$

where $E_i \subseteq X$.

Definition 6 (Message passing)**.** *Message passing is the principle that (hyper)graph neural networks, like GCNs, use. Message passing algorithms learn the node embeddings by aggregating information from immediate neighbors, where the initial features of neighbors are represented in the feature matrix* $\mathbf{X}$*. In one message passing step, each node sends its current feature vector to its neighbors and receives the aggregated feature representation of its neighbors.*
Multiple message passing steps can be stacked. This is also called l *hops because you include node features at a distance* l *away in the representations.*

Definition 7 (Graph convolutional layer)**.** *GCNs apply convolutions over graphs. GCNs repeatedly aggregate feature vectors from neighbors to learn node embeddings. The graph convulational layer is formulated as:*

$$\mathbf{X}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}\mathbf{X}^{(l)}\mathbf{P}^{l}\right),$$

where σ *is a non-linear activation function,* $\mathbf{A}$ *is the adjacency matrix,* $\tilde{\mathbf{A}} = \mathbf{A} + I_n$ *is the adjacency matrix with self connections,* $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ *and* $\mathbf{P}^{l}$ *is a trainable weight matrix. The rows of* $\mathbf{X}^{(l+1)}$ *contain the final representations of the nodes in layer* $l + 1$*.*
A GCN consists of multiple graph convolutional layers and can be used both in the supervised and unsupervised setting. If a GCN is used for a supervised task, then the last layer is a softmax layer.

Definition 8 (Hypergraph neural networks)**.** *Hypergraph neural networks apply a GCN over the clique expansion of a hypergraph. In a hypergraph neural network, the embedding of the* i*-th vertex in the* (l)*-th layer* $x_i^{(l)}$ *is formulated as:*

$$x_i^{(l)} = \sigma\left(\sum_{j=1}^{n}\sum_{e=1}^{m} H_{ie}H_{je}W_{ee}x_j^{(l)}\mathbf{P}\right).$$

The vector representation of each layer in the network is:

$$\mathbf{X}^{(l+1)} = \sigma\left(\mathbf{H}\mathbf{W}\mathbf{H}^{T}\mathbf{X}^{(l)}\mathbf{P}^{l}\right),$$

where

$$\sigma = \textit{non-linear activation function,}$$

$$\mathbf{P}^l = d_{in} \times d_{out} \textit{ weight neural network matrix,}$$

$$\mathbf{H} = (H_{ie}) \textit{ is a } n \times m \textit{ incidence matrix,}$$

$$\mathbf{W} = (W_{ee}) \textit{ is a } m \times m \textit{ hyperedge weight matrix.}$$

For an undirected hypergraph H_{ie} is defined by:

$$H_{ie} = \begin{cases} 1, & \textit{if } v_i \in E_e \\ 0, & \textit{if } v_i \notin E_e. \end{cases}$$

Normalizing this gives:

$$\mathbf{X}^{(l+1)} = \sigma\left(\mathbf{D}^{-1/2}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^T\mathbf{D}^{-1/2}\mathbf{X}^{(l)}\mathbf{P}^l\right),$$

where

$$\mathbf{D} = (D_{ii}), \textit{ where } D_{ii} = \sum_{e=1}^{m} W_{ee}H_{ie}, \textit{ is a } n \times n \textit{ vertex degree matrix,}$$

$$\mathbf{B} = (B_{ee}), \textit{ where } B_{ee} = \sum_{i=1}^{n} H_{ie}, \textit{ is a } m \times m \textit{ hyperedge degree matrix.}$$

Which is the vector form of

$$x_i^{(l)} = \sigma\left(\sum_{j=1}^{n}\sum_{e=1}^{m} \frac{H_{ie}H_{je}W_{ee}}{B_{ee}\sqrt{D_{ii}D_{jj}}} x_j^{(l)}\mathbf{P}\right).$$

This formulation makes sure that information is shared within each hyperedge and between hyperedges. The number of layers controls how deep the information of each node flows.

3.2 Modular Directed Hypergraph Neural Network

In this thesis, we design a directed hyperlink prediction model based on a graph neural network approach since recent developments have shown the potential of graph neural networks for making predictions over graphs [25, 34, 54]. Besides, all state-of-the-art (hyper)graph learning models are graph neural networks.
In most graph neural network models, an embedding for a hyperedge is generated by aggregating its node embeddings. We follow this principle for our directed hyperlink predictor.

HGNN [17] is designed for undirected hypergraphs and propagates messages within each hyperedge and between hyperedges. Therefore, HGNN cannot be directly applied on directed hypergraphs since it does not consider the direction of a hyperedge. For directed hypergraphs we also want a model where information is shared within each hyperedge and between hyperedges. Hence, we propose MDHNN which is a model consisting of three modules named induced tail hypergraph, induced head hypergraph and directed hypergraph.

First of all, we will introduce the induced tail hypergraph module. In this module, we want to propogate information within and between the tails of the hyperedges. We do this by “inducing" an undirected hypergraph H_{tail} from H, i.e. create an undirected edge from every tail in H. Formally, H_{tail} is defined by

$$V_{tail} = \{v_i \mid v_i \in T(E_e) \ and \ E_e \in H\},$$

$$E_{tail} = \{\{v_i \mid v_i \in T(E_e)\} \mid E_e \in H\}.$$

Now we can share the information by applying the undirected hypergrah convolution

$$\sigma\left(\mathbf{D_{tail}}^{-1/2}\mathbf{H_{tail}}\mathbf{W}\mathbf{B_{tail}}^{-1}\mathbf{H_{tail}}^{T}\mathbf{D_{tail}}^{-1/2}\mathbf{X}^{(l)}\mathbf{P}^{l}\right).$$

The same principle can be applied for sharing information within and between the heads. This module is called induced head hypergraph.

Lastly, we also want to diffuse information between the tail and the head of a hyperedge. We do this in the directed graph module by applying GCN to the directed clique expansion DCE. DCE is defined by

$$E_{DCE} = \{(v_i, v_j) \mid v_i \in T(E_e) \text{ and } v_j \in T(E_e) \text{ and } E_e \in H\}.$$

Since the adjacency matrix of a directed graph can be asymmetric and thus the information of the nodes need to diffuse asymmetrically, we adapted the normalization of GCN to $\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$ and the convolution will be

$$\mathbf{X}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\mathbf{X}^{(l)}\mathbf{P}^l\right)$$

where $\tilde{\mathbf{D}} = \tilde{\mathbf{D_{in}}} + \tilde{\mathbf{D_{out}}}$.
By aggregating the three different embeddings of the different modules, we get a final embedding for each node in the directed hypergraph

$$\begin{aligned}\mathbf{X}^{(l+1)} = AGG\Big(&\sigma\left(\mathbf{D_{head}}^{-1/2}\mathbf{H_{head}}\mathbf{W}\mathbf{B_{head}}^{-1}\mathbf{H_{head}}^{T}\mathbf{D_{head}}^{-1/2}\mathbf{X}^{(l)}\mathbf{P}^l\right),\\ &\sigma\left(\mathbf{D_{tail}}^{-1/2}\mathbf{H_{tail}}\mathbf{W}\mathbf{B_{tail}}^{-1}\mathbf{H_{tail}}^{T}\mathbf{D_{tail}}^{-1/2}\mathbf{X}^{(l)}\mathbf{P}^l\right),\\ &\sigma\left(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}\mathbf{X}^{(l)}\mathbf{P}^l\right)\Big).\end{aligned}$$

To get the embeddings of the edges we aggregate the embeddings of the nodes in each hyperedge. We perform aggregation by either maxmin_+ or mean_+. maxmin_+ is defined by

$$\text{maxmin}_+\{\mathbf{x_v}\}_{v\in E} = (\max_{s\in T(E)} x_{sl} - \min_{i\in H(E)} x_{il})_+, \tag{3.1}$$

where $l = 1, 2, \ldots, d$, d is the dimension of the node embeddings $\mathbf{x_v}$ and $\text{m}_+ = m$ if $m > 0$ and $\text{m}_+ = 0$ otherwise.

mean_+ is defined by

$$\text{mean}_+\{\mathbf{x_v}\}_{v\in E} = (\underset{s\in T(E)}{\text{mean}} x_{sl} - \underset{i\in H(E)}{\text{mean}} x_{il})_+, \tag{3.2}$$

where $l = 1, 2, \ldots, d$ and d is the dimension of the node embeddings $\mathbf{x_v}$.

After we get the edge embeddings, we feed those to a multi layer perceptron (MLP). The final layer has one node which outputs if the hyperedge is (to-be) present in the graph or not.
In the architecture of MDHNN we use ReLU for σ and add dropout between each convolutional layer. We use ReLU since it is commonly used in deep learning and graph neural networks.

3.3 Research methodology

The architecture of MDHNN has been defined. In the next steps, the model will be implemented. After the implementation, proper datasets are collected/selected and pre-processed as discussed in Chapter 4. Thereafter, an experiment set-up has been determined. We refer to Chapter 5 for the experiment set-up. The results of the experiments have been reported in Chapter 6. The final conclusions and suggestions for future work are given in Chapter 7.

Chapter 4

Dataset selection

In this chapter, we first describe the challenges that were encountered in ING's use case. Next, we describe the selected datasets used for evaluation and the pre-processing steps that are applied. Finally, we compare the original dataset of ING to the validation datasets.

4.1 ING's challenge

The hypergraph used that reflects a domain model is created by using data from the Github [1] and Gitlab [2] APIs. The dataset contains a set of higher-order relations. In such a higher-order relation $< \{a, b\}, \{f\}, \{c\} >$, the first set $\{a, b\}$ is the set of inputs, the second set $\{f\}$ is the callable unit and the last set $\{c\}$ is the output. This is (almost) naturally encoded as a hypergraph where each high-order relation would represent an edge. Since the task is to predict new higher-order relations, i.e. hyperedges, in this dataset and the callable unit is unique for each hyperedge, the callable unit is removed.

After removing the callable unit there are some duplicate edges. Since we only care about the paths in the graph, the duplicate edges are removed.

A special characteristic of the hypergraph is that the output set always has a size

[1] `https://docs.github.com/en/rest`

[2] `https://docs.gitlab.com/ee/api/`

of one. Besides, the final hypergraph, retrieved on 22^{th} of February 2021, has $|V| = 956$ and $|E| = 3935$. The majority of the edges are derived, i.e. automatically generated, and only 25 edges are non-derived, i.e. manually defined. According to expert knowledge, only these manually defined edges are relevant, given that the others are systemically generated by software.

Table 4.1: Performance on ING's use case. The AUC scores have been calculated over five different runs. The set of random hyperedges has a size of five percent of the dataset size.

	Mean ± std	**Max**
AUC	0.96 ± 0.00	
Probability test non-derived hyperedges	0.57 ± 0.12	
Probability train non-derived hyperedges	0.62 ± 0.15	
Probability random hyperedges	0.21 ± 0.27	0.90

By logical reasoning, one might conclude that making relevant hyperedges recommendations based on a set of 25 hyperedges will be impossible. Since the graph is sparse, i.e. density of 0.029, the set of unobserved hyperedges is much larger than the set of (relevant) observed hyperedges and the logical reasoning that can be done on such a small dataset is limited. However, the expectation is that the full hypergraph can be used to learn a prediction function that give high probability to relevant non-derived hyperedges. Moreover, it is expected that the hypergraph will grow over time and thus the predictions a model makes will improve.

For the first experiment, a simple model has been used. The model uses node2vec [23] to learn node embeddings. After the node embeddings are generated, the embeddings for the hyperedges need to be created. Therefore, the node embeddings for the head and tail of each hyperedge are aggregated. The embeddings for the head and tail for each hyperedge are fed into a MLP where the last layer consists of one node. This node outputs the probability that the hyperedge is existent in the graph or not (the metric that measures the effort of adding hyperedges to a hypergraph), i.e. sigmoid activation. The optimizer used is stochastic gradient descent and the loss function is binary cross-entropy. The full details of the experiment can be found in Appendix A.

From the AUC in Table 4.1 it can be concluded that the model has good performance. However, most of the edges are derived and the probability for the non-derived edges need to be high in order to make relevant predictions. If we look at the predictions table, it can be seen that the probabilities for the non-derived edges are low. Combining this with the fact that the AUC is high, we can conclude that the ranking of non-derived edges is low. Besides, randomly created hyperedges sometimes get a high score while they were expected to, according to the domain expert, get a low score. The reasoning behind this is that each hyperedge has a meaning in the business context. Hence, a random hyperedge does not represent a concept known from business. Some random hyperedges get a high score because there are only 25 edges to learn from for relevant suggestions. Therefore, it is unlikely to have relevant suggestions. Hence, we use other datasets for the experiments. However, for setting up those experiments ING's use case is kept in mind. Such that we could conclude which model to test once the dataset is larger.

4.2 Alternative datasets

The graph datasets chosen represent a relevant subset of those most frequently used in literature to compare hypergraph neural networks. Some literature uses metabolic networks while others uses social graphs. The assumption is that the social graphs are most similar to ING's domain models. The reason for this is that a domain model consists of separate sub domains that are likely to be barely related or not related at all to other sub domains and the expectation is that those groups of sub domains behave like groups/clusters of people.

All graph datasets used in the experiments are publicly available. However, some pre-processing would still need to be applied.

Table 4.2: Statistics of metabolic networks.

Metabolic network	$\|V\|$	$\|E\|$	**Number of features**
iAF692	628	616	24
iAF1260b	1,668	2,063	46
iJO1366	1,805	2,231	46

4.2.1 Metabolic datasets

Metabolic networks are essential tools for, among others, understanding the metabolic basis of human diseases, genetic engineering and drug discovery [12]. However, one issue is that these networks are inherently incomplete since unknown reactions are missing from them. This severely impairs the use of these networks [36]. The problem of finding the missing hyperlinks can be modeled as a hyperlink prediction problem. A metabolic network can be modeled as a directed hypergraph by taking the metabolites (reactants and products) as vertices and the reactions as hyperarcs connecting the reactants and products. The tail of a hyperarc contains the reactants and the head contains the products of the reaction.

We have used three different metabolic networks, i.e. iAF692, iAF1260b and iJO1366, from BiGG [3] that have also been used in previous studies [52, 58, 63]. The statistics of the three metabolic networks are shown in Table 4.2. Before the data can be used, the networks have been pre-processed by performing the following steps:

1. Generate feature matrix $\mathbf{X}$. First, retrieve the features charge, compartment and the chemical elements of the formula. These are the only features related to the nodes that could be retrieved. Thereafter, create a binary matrix using the bag-of-words model.

2. Reactions can appear multiple times in metabolic networks. The copies of the reaction have different gene reaction rules or reaction bounds. Since hyperlink prediction does not use features of the edges, those reactions are considered as duplicate and are filtered. This step is only applicable to metabolic network iJO1366.

[3] `http://bigg.ucsd.edu/`

3. There are also pseudo-reactions among the reactions. Biomass, exchange, demand, and source/sink reactions are all pseudo-reactions. Pseudo-reactions do not represent a specific enzyme-catalyzed transformation. Hence, these are also filtered.

4.2.2 Co-citation datasets

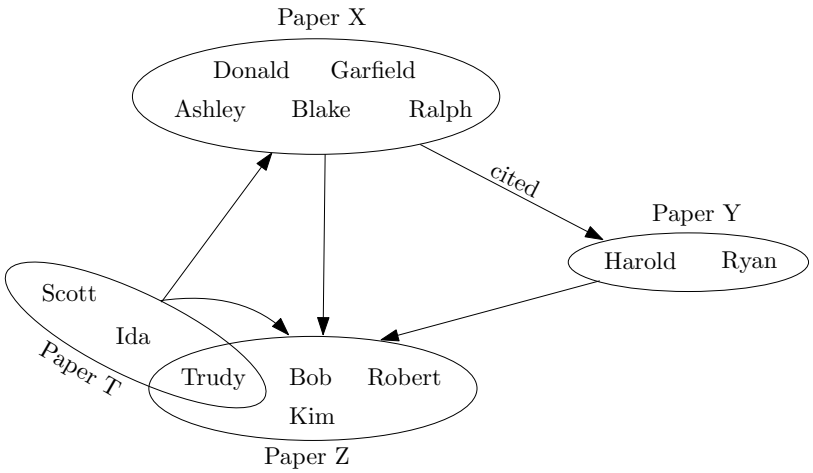


Figure 4-1: Example of a co-citation network. Each node represents an author and each tail/head represents a paper. Each hyperedge represents a citation.

Table 4.3: Statistics of co-citation networks.

Co-citation network	$\|V\|$	$\|E\|$	**Number of features**
dblp	10,804	11,097	300
MAG	28,976	25,135	9,157
MAG subset	6,480	3,898	4,027

We model two bibliography datasets as (directed) co-citation hypergraphs. In these hypergraphs, authors are the nodes and the tail and head of each hyperedge represents a paper. Where the paper representing the tail cited the paper representing the head. Hence, this graph models which authors have written a paper together and which papers a paper has cited. An example is shown in Figure 4-1.

The bibliography datasets are retrieved from dblp [4] and Microsoft Academic Graph

[4] `https://www.aminer.org/citation`

(MAG) [5]. The statistics of both social graphs are shown in Table 4.3.

In this section it should be noted that both graphs are inherently dynamic. So you could split the data based on time. However, as explained in Section 5.1, multiple splits are needed to get a fair comparison between the different model architectures. Hence, time is ignored and random splits for the sake of a fair comparison are created.

Table 4.4: Cosine similarities of different organizations. The cosine similarity is calculated between the embeddings of the different organizations.

Organization	**Cosine similarity**
Australian Centre for Field Robotics, The University of Sydney, Sydney, Australia	0.978
Australian Centre for Field Robotics, The University of Sydney, NSW, Australia	
Australian Centre for Field Robotics, The University of Sydney, Sydney, Australia	0.900
Australian Centre for Robotic Vision, Queensland University of Technology, Brisbane, Australia	
Australian Centre for Field Robotics, The University of Sydney, NSW, Australia	0.916
Australian Centre for Robotic Vision, Queensland University of Technology, Brisbane, Australia	

Organization	**Cosine similarity**
Computer Science Department, Stanford University, Stanford, CA	0.929
Computer Science Department, Stanford University, California 94305 USA	0.758
Computer Science Department, Rutgers University, New Brunswick, NJ	0.789

Table 4.5: Cosine similarities of the same organizations. The cosine similarity is calculated between the embeddings of the different organizations.

Organization	**Cosine similarity**
Microsoft Corporation, One Microsoft Way, Redmond, WA 98052	0.889
Microsoft Corporation, Redmond, WA, USA	

[5] `https://www.microsoft.com/en-us/research/project/microsoft-academic-graph/?from=https%3A%2F%2Fresearch.microsoft.com%2Fmag`

dblp

dblp is a computer science bibliography. The dataset, as retrieved on the 9th April 2020, contains almost 5M papers. Therefore, we take a subset of the data and pre-process it using the following steps:

1. Retrieve all papers and their authors and citations that have their field of study in deep learning.

2. Create hyperedges by joining papers on the citation paper ids. We only join papers that have the field of studies in the set just mentioned, i.e. the tail and the head papers must have a field of study in the mentioned set. In this way we focus on connected component(s) and do not take other fields of study into the graph.

3. Remove duplicate hyperedges.

4. The node feature matrix $\mathbf{X}$ is constructed by using the author's organization. However, no unique identifiers are known for the organizations. We only have organization strings which might differ between papers. You could perform string matching to link every organization name to a unique set of organizations. However, we do not have such a unique set of organizations. Therefore, we use a word embedding approach. We used pre-trained word vectors, of dimension 300 trained by GloVe [45] on Wikipedia's text, to get an embedding for every token in the organization name and take the mean to get a final embedding. This final embedding represents the full organization name, where the expectation is that the same organization names end up with similar embeddings. Before getting the embeddings, we pre-process the raw text by the following steps: to lowercase, remove "#tab#" at the end of some organization names, remove numbers, remove non-alphabetic characters, remove white spaces, remove stop words, remove single letters and tokenize the string.

   We tested if the embeddings of same organizations will be closer than embeddings of different organizations. In Table 4.4, it can clearly be seen that the

same organizations are closer than different organizations. Furthermore, from Table 4.5 we can conclude that even if the strings are very different, then the similarity is still high. To measure the similarity, we have used cosine similarity since this measure is widely used for text similarity. Another advantage of using embeddings is that an organization name like "Carnegie Mellon University and Microsoft Research Asia" could be well represented using embeddings since it will take the mean of the two.

Microsoft Academic Graph

The Microsoft Academic Graph (MAG) contains over 200M scientific publication records. Hence, we take a subset of the data, retrieved on the 7th of June 2021, and pre-process it by taking the following steps:

1. Retrieve all papers and their authors and citations that have their field of study in either: deep neural networks, deep cnn, deep belief network, super resolution convolutional neural network, autoencoder or softmax function.

2. Create hyperedges by joining papers on the citation paper ids. We only join papers that have the field of studies in the set just mentioned, i.e. the tail and the head papers must have a field of study in the mentioned set. In this way we focus on connected component(s) and do not take other fields of study into the graph.

3. Remove duplicate hyperedges.

4. Retrieve authors' citation count, publication count and affiliation id for the authors that appear in the graph. Generate feature matrix $\mathbf{X}$ by creating a binary matrix using the bag-of-words model and the authors' data.

4.2.3 Final pre-processing

The problem with the datasets is that the hypergraphs are sparse. Therefore, extreme class imbalance is introduced since most edges are unobserved. The amount of unobserved edges are of order $\mathcal{O}(2^{|V|} - |F|)$, where F is the set of unobserved edges. For this reason, we need to undersample the negative class and need to decide on a ratio of negative edges. We use a ratio of 1.5, i.e. 1.5 more negative edges than positive edges. This is done such that the problem does not get too unbalanced. The ratio of 1.5 has been chosen because Caselles-Dupré et al. [13] found that a similar parameter setting was optimal for word2vec. The negative edges are created by taking a positive edge and permuting 50 percent of the nodes. This approach is called sized negative sampling. Sized negative sampling is preferred over randomly sampling negative edges, also known as uniform negative sampling. Randomly sampling of non-hyperedges is not preferred because the distribution of non-hyperlink sizes would be completely different than the distribution of hyperedge sizes [43]. Therefore, in that case we could solve the problem by a single trivial feature, i.e. hyperlink size. By using sized negative sampling the model would learn a more interesting function since the edges would be harder to separate. Moreover, two state-of-the-art models that have been evaluated on hyperlink prediction, NHP [58] and HyperSAGNN [65], did also use sized negative sampling.

Finally, before inputting the data to the models we need to represent the hypergraph data in the right format, e.g. incidence matrix in coordinate or dense format, edgelist, etc.

4.3 Hypergraph similarity

To test if our assumption that the domain model graph is most similar to the social graphs holds, we have made a comparison based on different graph metrics. The idea is that similar graphs probably share certain properties. The calculated measure could be aggregated by using a similarity measures to have one final measure which can be used to assess the similarity of the different graphs. However, since it is not known which measures are important, the measures are not aggregated. The measures are compared individually.
This method of graph similarity is used since this method is powerful and scales well since the graphs are mapped to several measures which are much smaller in size than the graphs.

The used measures are simple measures because of their feasibility to calculate them in limited time. The following measures are calculated over the graphs:

1. ratio strongly connected components (SCCs): number of strongly connected components of the directed clique expansion divided by the number of nodes;

2. ratio weakly connected components (WCCs): number of weakly connected components of the directed clique expansion divided by the number of nodes;

3. assortativity coefficient: the assortativity coefficient measures the preference of nodes to attach to other nodes which have similar degrees. This is calculated by the Pearson correlation coefficient of degrees between pairs of connected vertices. If assortativity is positive, then nodes of similar degrees are correlated. Else, nodes of different degrees are correlated. The assortativity coefficient is calculated over the directed clique expansion;

4. mean/variance tail size: mean/variance of the number of nodes in the tails;

5. mean/variance head size: mean/variance of the number of nodes in the heads;

6. mean/variance edge size: mean/variance of the number of nodes in the edges (head+tail);

7. mean/variance in-degree: mean/variance of the number of times nodes appear in the tail;

8. mean/variance out-degree: mean/variance of the number of times nodes appear in the head;

9. mean/variance degree: mean/variance of in-degree plus out-degree;

Besides reporting those statistics, violin plots have been created, see Appendix C, over the degrees and sizes.

Table 4.6: Hypergraph metrics of the different datasets.

	base	**MAG**	**dblp**	**iJO1366**	**iAF1260b**	**iAF692**
Ratio SCC	0.530	0.149	0.159	0.171	0.174	0.041
Ratio WCC	0.011	0.002	0.005	0.0006	0.0006	0.003
Assortativity	0.914	-0.093	-0.098	-0.289	-0.292	-0.315
Density	0.029	0.0005	0.002	0.005	0.005	0.014
Mean tail size	13.222	3.893	4.066	2.027	1.999	2.234
Var tail size	570.999	4.197	21.069	0.553	0.515	0.539
Mean head size	1.000	3.717	4.331	2.274	2.270	2.502
Var head size	0.000	3.499	24.885	0.886	0.875	0.886
Mean edge size	14.222	7.610	8.397	4.301	4.268	4.735
Var edge size	570.999	8.200	48.080	2.446	2.358	2.315
Mean in-degree	4.100	3.224	4.448	2.811	2.807	2.454
Var in-degree	1328.369	1738.732	947.194	423.753	394.992	141.063
Mean out-degree	54.212	3.377	4.176	2.505	2.472	2.191
Var out-degree	23712.555	20.749	33.967	267.929	239.921	85.846
Mean degree	58.312	6.601	8.624	5.316	5.279	4.645
Var degree	24912.762	1859.408	1133.836	1021.327	899.925	333.891

As you can infer from Table 4.6, both the social graphs and the metabolic networks have very similar statistics. Especially the metabolic networks have very similar means, variances and measures. Only for some metrics iAF692 is a bit different compared to iJO1366 and iAF1260b. The social graphs also are very similar, but the variances vary a bit.

However, the statistics for the social graphs and metabolic networks do not compare at all. So you can conclude that the two groups of graphs are really different. This can also be confirmed by the violin plots in Figures C-1-C-6 in Appendix C.

Base is the graph representing the domain model. If you compare base to the other graphs with respect to the first set of statistics, i.e. ratio SCC, ratio WCC, assortativity and density, then you cannot conclude that base is highly similar to another graph. Assortativity is positive, where it is negative for all the other graphs. Besides, density and ratio of SCC and WCC is higher.
For the sizes, we can conclude that the metabolic networks have relatively small edge sizes. The sizes of the social graphs are larger. The edge sizes of the base are also large, even larger than the social graphs, with an exception for the head since this is always fixed to one.
A similar pattern is seen for the degrees. The metabolic networks have small degrees and the degrees of the social graphs are larger. Again, the degrees of the base are the largest.
If we compare the base network to the other graphs based on the distributions in Appendix C, then we can conclude that the distributions, except for the in-degree, for the base graph are very different. The in-degree distribution is most similar to the ones of the metabolic networks.

Since the base graph has large degrees and sizes and the social graphs also has large degrees and sizes, we conclude that the social graphs are the most similar to the domain graph. Furthermore, from the social graphs we can conclude that dblp is most similar to base. Because the ratio WCC and assortativity is most closest to the dblp dataset.

Chapter 5

Experiments

In this chapter, it is first described how the experiments are designed. Thereafter, we will discuss which models are used as baseline models in the experiments. Finally, the evaluation metrics used are explained.

5.1 Experiment design

We want to guarantee experimental reproducibility and replicability of the experiments. Moreover, we want to make a fair comparison between the different models in order to attribute, with high certainty, the differences in performance to the differences in model architectures. Hence, we need to have the same experimental set-up for all models [40]. Therefore, the set-up as reported by Shchur et al. [53] is closely followed. This set-up ensures that the tuning of the hyperparameters and training and evaluation procedure are set up fairly for all models. Even though we operate in a different setting (hyperlink instead of node classification), we follow the authors' suggestions by evaluating models under a standardized framework.

First of all the hyperparameter search procedure, also called model selection, will be explained. In Algorithm 1 in Appendix B an algorithmic overview can be found. For each dataset, we randomly split the data in five different train/test splits. Each train set will be further split using 5-fold cross-validation. For each configuration,

a model is trained and evaluated with AUC on the different splits and training and validation folds. The configuration with the highest average will be used for training and testing.

We take the average over five different splits since Shchur et al. [53] showed that using different splits of the data leads to different ranking of the architectures. Zügner et al. [69] also confirmed that small data perturbations causes the performance of graph neural networks to greatly change. Besides, by taking the average over multiple splits it will allow us to more accurately find the model with the best generalization properties [27] and not the model that performs well on one specific test set.

There is no common train/test ratio applied for graph learning. Most papers use more train than test data. Some training percentages seen are: 80 [8, 65], 75 [33], 70 [50, 30], 60 [5, 38, 61] and 50 percent [11, 59]. The ratios seen if papers use more test than train data are: 20 [58], 5 [17] and 0.3 percent [17]. Other papers let the percentage of train data depend on the dataset size [17, 63, 64]. For node classification you also have the option to use x nodes of each class for training [6, 57, 53, 39]. This results in a training percentage in the range of 0.05 - 0.003 percent. Therefore, we use a common split that is used in general machine learning. We use 80 percent for training and 20 percent for testing.

The hyperparameter search space for each model is shown in Table **??** in Appendix D. We use Bayesian optimization (BO) with the default setting of 20 trials for searching the space. It would be better to use grid search to get a more fair comparison. However, all possible configurations grow exponentially in the number of hyperparameters and the number of choices and thus it is too computationally expensive. Moreover, grid search wastes time trying useless hyperparameter configurations.

In the next stage, we will perform the training. Adam [31] is used as the optimizer with L_2 regularization. The weights are initialized using Glorot initialization [21] and the biases are initialized as zeros. All models get the same search space for the learning rate, L_2 regularization parameter and the number of epochs. Besides, all

data is used in one epoch, i.e. full-batch training.

After the model selection and training phase we need to get unbiased test evaluation metrics, see Algorithm 2 in Appendix B for an overview.
For the model assessment phase it is crucial that it is being well separated from the other phases to get an unbiased evaluation metric. Hence, the test data is only used in this phase. Furthermore, we perform three separate runs for each split to minimize the effect of unfavorable random weight initialization.

We followed more good experimental practices as reported by Errica et al. [14]. Firstly, we split the data by means of the stratification technique to preserve class proportions across the different splits and folds. Secondly, we make sure that all models are selected and evaluated on the same data splits. Lastly, we use the same input representations for the nodes for all models. We use the feature matrix $\mathbf{X}$ as described in the pre-processing steps. In this way, we are able to test if the model is able to learn structural features.

Table 5.1: Technical specifications of the HPC cluster.

#nodes	**#cores/node**	**total cores**	**RAM [GB/core]**	**CPU type**
12	48	576	10.7	Xeon Platinum 8260

All models are implemented in PyTorch Geometric [18] and the experiments were performed on an HPC cluster. The technical specifications of the HPC cluster have been shown in Table 5.1. We did use CPUs, since then we could apply multiprocessing. Running the models on CPUs was faster than using GPU/CUDA and running it sequentially. Using multiprocessing was faster since the number of processes to run is too large such that the gain of running it sequentially on a GPU does not speed up the full computation.

5.2 Baseline models

As previously explained, most literature makes an unfair comparison between the baseline models and the proposed model. Sometimes, not enough details are given to conclude if the comparison is fair. Hence, for example, we cannot conclude if we could eliminate HGNN from our baseline models because comparison of HyperGCN is unfair since they do not optimize the hyperparameters [57]. Besides, most baseline models are not directly compared in the literature. Therefore, there are a number of models considered as the baseline models.

For the inductive models, i.e. models that use a pipeline to learn node embeddings and make predictions for the hyperedges, we compare HGNN [17], Hypergcn [57], HyperSAGNN [65] and NHP [58].
We excluded Hypergraph Convolution and Hypergraph Attention [6] and hyperedge2vec [50]. Hypergraph Convolution is the exact same model as HGNN and thus would not add any new information. Hypergraph Attention cannot be used since it is only feasible when the vertex and hyperedge set are from the same homogeneous domain. In the validation datasets, the vertex set either represents authors or metabolites and the hyperedge set citations or metabolic reactions, respectively. Therefore, the vertex set and the hyperedge set are from two heterogeneous domains. hyperedge2vec proposes a Hasse denoising autoencoder which generates hyperedge embeddings based on the full hypergraph. This approach can be used for hyperedge classification, but is not suitable for (hyper)link prediction.
All models generate node embeddings. For each model, those embeddings are aggregated in hyperedge embeddings by using Equation 3.1 or Equation 3.2. To get a final score for each hyperedge, a small MLP is applied.

Both HyperGCN and HGNN use the fulll structure of the graph and the information in the nodes to make predictions. Since NHP and HyperSAGNN only input a hyperedge to make a prediction, those models are structure-agnostic and thus can be used

to conclude if topological information is needed to make correct predictions.
However, all the inductive models are based on generating embeddings. Therefore, we have also included a simple inductive model that is not based on embeddings. We pre-process the data such that each row represents one hyperedges and each column is a combination of a node feature and its corresponding value indicating how many nodes in the hyperedge got that value for the feature. We did not use one-hot encoding to represent the nodes since this would require $n \times \sum_{f \in features} \#possible\ values\ of\ f$ columns. The pre-processed data is fed into a MLP.

We also want to use a transductive method that only use the graph structure and ignore node features. Some transductive methods are: node2vec [23], hyper2vec [29], Deep Hyperedges [44] and hyperedge2vec [50]. node2vec and hyper2vec work by generating node embeddings by performing random walks on each node and inputting the random walks to the skip-gram model. Deep Hyperedges performs node2vec with specialized random walks on the hypergraph and hypergraph dual to get respectively node and hyperedge embeddings. hyperedge2vec's Hypergraph Tensor Decomposition works by performing tensor decompositions over the hypergraph (dual) to generate node and hyperedge embeddings. The only issue with these models is that all nodes (and for Deep Hyperedges and hyperedge2vec also all hyperedges because you use the dual hypergraph) need to be present in the training set. Hence, the *full dataset − positive edges in validation fold − positive edges in test set* must contain all nodes. We tested if this is requirement holds for all five random data splits and even with a 90/10 percent split this requirement does not hold. Therefore, we can conclude that transductive methods that generate embeddings are sufficient for node and hyperedge classification and regression, but not for hyperlink prediction.
Moreover, transductive methods that do not generate embeddings can also not be used since those are all designed for undirected hypergraphs. For example, Coordinated Matrix Minimization [63] and Clique-Closure based Coordinated Matrix Minimization [52] are methods based on matrix completion on an undirected hypergraph.
Since we still want to test if the node features add any information, we decided to add

the best performing structure-based inductive model to which we input node feature matrix $\mathbf{X}$ as the identity matrix. This model will be called structure.

5.3 Evaluation metrics

We use Area Under the ROC Curve (AUC) to get the overall performance of the different models. An ROC (receiver operating characteristic) curve plots the True Positive Rate (TPR) versus the False Positive Rate (FPR) at different classification thresholds. The TPR is defined by $\frac{True\ Positives}{True\ Positives+False\ Positives}$ and the FPR is defined by $\frac{False\ Positives}{False\ Positives+True\ Positives}$. The AUC provides an aggregated measure by performing integration over the ROC curve.

For ING's use case we want to retrieve edges that are likely missing, i.e. it needs to function as a recommender system. Hence, we want the top predictions to be relevant. Therefore, we also evaluate the models based on precision@k and recall@k for a range of k values. Precision@k and recall@k are precision and recall under the assumption that the top k predictions is the recommended set.
Precision@k is defined by

$$\frac{number\ of\ relevant\ hyperedges\ in\ top\ k}{k},$$

where relevant hyperedges are (to-be) existing hyperedges. Thus it represents the proportion of top k hyperedges that are relevant.
Recall@k is defined by

$$\frac{number\ of\ relevant\ hyperedges\ in\ top\ k}{number\ of\ relevant\ hyperedges},$$

where relevant hyperedges are (to-be) existing hyperedges. Thus it represents the proportion of relevant hyperedges that are in the top k.

Chapter 6

Results and discussion

The results of the experiments will be presented in this chapter. First, we discuss specific experiment settings. In the next section, we discuss the performance results on the different models and datasets. After that, we will compare the performance of a directed and non-directed version of HGNN. Moreover, we will discuss the performance of MDHNN in more detail. We will evaluate the three different modules separately and determine the sensitivity of MDHHN with respect to embedding dimension and train/test ratio. Finally, we also discuss the threats to validity of the experiments.

6.1 Specific experiment settings

Since some experiments were too computationally expensive, we needed to make some changes to the experiment design. First, running the hyperparameter search on MAG and dblp would be too expensive. For this reason, we run the hyperparameter search on a subset of MAG. The subset is created the same way as the MAG dataset was pre-processed, only we take the papers where the field of study is in deep cnn, residual neural network, super resolution convolutional neural network or softmax function. This subset of MAG is used because it contains different fields of study, i.e. connected components, and hence is most similar to the MAG and dblp datasets. Since this subset is from the same domain as the MAG and dblp dataset, we use the optimal

hyperparameter set to get the test performance for dblp and MAG.

Secondly, performing full batching for NHP and HyperSAGNN is infeasible. Since the models are structure-agnostic, we can perform batching to speed up the training. Hence, we applied batching with a batch size of 64. A batch size of 64 has been chosen since this was also used by NHP.

6.2 Results

The mean AUC, precision@k and recall@k scores over the several runs have been reported in Tables 6.1/6.4, 6.3/6.6 and 6.2/6.5, respectively. We also reported the standard deviation over the multiple runs. It only should be noted that standard deviations are not the best representation of the variance of the scores, since the scores are not normally distributed. However, we still report the standard deviations to give an indication of the variance of the results for each model. Figures F-1-F-5 in Appendix F give a full picture of the distributions. Moreover, HyperSAGE and HyperSAGNN could not be evaluated on the MAG dataset since this would take more than two weeks to run. Hence, the performance is reported as not available.

Table 6.1: AUC scores of the different models on the metabolic networks. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split.

	iAF692	iAF1260b	iJO1366
HyperSAGE	**0.932 ± 0.011**	0.958 ± 0.004	0.963 ± 0.006
HyperGCN	0.685 ± 0.034	0.701 ± 0.018	0.712 ± 0.021
HGNN	0.796 ± 0.030	0.902 ± 0.007	0.894 ± 0.005
HyperSAGNN	0.784 ± 0.082	0.952 ± 0.013	0.958 ± 0.010
MDHNN	0.787 ± 0.011	0.905 ± 0.035	0.897 ± 0.036
NHP	0.916 ± 0.017	**0.970 ± 0.005**	**0.976 ± 0.004**
MLP	0.618 ± 0.169	0.890 ± 0.153	0.880 ± 0.153
Structure	0.593 ± 0.042	0.559 ± 0.043	0.553 ± 0.017

Table 6.2: Recall@10 scores of the different models on the metabolic networks. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split.

	iAF692	iAF1260b	iJO1366
HyperSAGE	0.075 ± 0.006	0.023 ± 0.001	0.021 ± 0.001
HyperGCN	0.051 ± 0.009	0.014 ± 0.003	0.015 ± 0.004
HGNN	0.068 ± 0.009	0.024 ± 0.000	0.022 ± 0.001
HyperSAGNN	0.064 ± 0.011	0.023 ± 0.001	0.022 ± 0.001
MDHNN	0.067 ± 0.009	0.022 ± 0.002	0.022 ± 0.001
NHP	**0.092 ± 0.009**	**0.029 ± 0.001**	**0.028 ± 0.000**
MLP	0.047 ± 0.024	0.021 ± 0.005	0.020 ± 0.004
Structure	0.041 ± 0.016	0.010 ± 0.004	0.006 ± 0.004

Table 6.3: Precision@10 scores of the different models on the metabolic networks. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split.

	iAF692	iAF1260b	iJO1366
HyperSAGE	**0.927 ± 0.077**	0.960 ± 0.061	0.953 ± 0.062
HyperGCN	0.633 ± 0.114	0.567 ± 0.125	0.673 ± 0.157
HGNN	0.847 ± 0.109	**1.000 ± 0.000**	0.967 ± 0.047
HyperSAGNN	0.793 ± 0.139	0.960 ± 0.049	0.967 ± 0.060
MDHNN	0.840 ± 0.114	0.927 ± 0.085	0.973 ± 0.057
NHP	0.907 ± 0.085	0.973 ± 0.044	**1.000 ± 0.000**
MLP	0.580 ± 0.295	0.873 ± 0.198	0.873 ± 0.181
Structure	0.507 ± 0.202	0.407 ± 0.148	0.267 ± 0.174

Table 6.4: AUC scores of the different models on the co-citation networks. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split.

	dblp	MAG
HyperSAGE	N/A	N/A
HyperGCN	0.504 ± 0.016	0.762 ± 0.115
HGNN	0.830 ± 0.027	0.906 ± 0.011
HyperSAGNN	**0.932 ± 0.006**	N/A
MDHNN	0.927 ± 0.003	0.933 ± 0.004
NHP	0.768 ± 0.009	**0.969 ± 0.001**
MLP	0.620 ± 0.036	0.522 ± 0.004
Structure	0.834 ± 0.007	0.841 ± 0.005

Table 6.5: Recall@10 scores of the different models on the co-citation networks. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split.

	dblp	MAG
HyperSAGE	N/A	N/A
HyperGCN	0.002 ± 0.001	**0.002 ± 0.0004**
HGNN	0.004 ± 0.001	**0.002 ± 0.0003**
HyperSAGNN	0.004 ± 0.0005	N/A
MDHNN	0.004 ± 0.0004	**0.002 ± 0.000**
NHP	0.002 ± 0.001	**0.002 ± 0.000**
MLP	**0.005 ± 0.000**	0.001 ± 0.0002
Structure	0.003 ± 0.001	**0.002 ± 0.0003**

Table 6.6: Precision@10 scores of the different models on the co-citation networks. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split.

	dblp	MAG
HyperSAGE	N/A	N/A
HyperGCN	0.413 ± 0.171	0.767 ± 0.189
HGNN	0.907 ± 0.124	0.893 ± 0.153
HyperSAGNN	0.913 ± 0.109	N/A
MDHNN	0.940 ± 0.071	0.993 ± 0.025
NHP	0.360 ± 0.125	**1.000 ± 0.000**
MLP	**1.000 ± 0.000**	0.280 ± 0.117
Structure	0.767 ± 0.162	0.820 ± 0.168

From the results we can conclude that MLP is performing rather good considering that it is such a simple model. A possible explanation for this is that only from the node features it can be determined if the hyperedge should exist or not. This is also confirmed by the fact that structure-agnostic models NHP and HyperSAGNN have high performance.

In general, NHP and HyperSAGNN have the highest performance, where overall NHP is the best performing model. However, the dblp dataset has very different performance across models compared to the other datasets. For the dblp dataset, HGNN and MDHNN have the best performance. The performance of the other models dropped. An explanation could be that the features are created differently for

the dblp dataset. The features could be less informative such that the models have less information to base their predictions on. Another explanation could be that the models could not generalize the hyperparameters learned on the MAG subset to the dblp dataset since the node features are so different. Both explanations are motivated by the fact that HGNN and MDHNN use the full graph structure and the fact that the performance on the MAG dataset has higher scores.

MDHNN especially performs good on the larger datasets. MDHNN has good evaluation scores and MDHNN has acceptable training times, see Table 6.7. However, on the small datasets it also has better performance than most of the baseline models.

The distribution in the figures in Appendix F show that NHP and HyperSAGE have low standard deviations over the different splits and runs. Hence, both models have good generalization power. MDHNN especially has low standard deviations for the two co-citation networks. This motivates that MDHNN has good (generalization) performance on larger datasets.

The good performance of NHP has also been confirmed in the tables in Appendix G that show the precision@k and recall@k performance for various values of k. For most datasets, NHP has the highest precision@k and recall@k scores. However, for the dblp dataset, MDHNN again has the highest scores.

From the comparison of HGNN, HyperGCN and HyperSAGE on the datasets, we can observe that sampling the nodes is a good regularization technique. Moreover, we can conclude by comparing HGNN and HyperGCN that representing a hyperedge with a linear number of edges really decreases performance.

The optimal hyperparameter sets have been reported in Appendix E. For most models, the hyperparameters are the same for the metabolic networks. This motivates the application of the learned hyperparameters on the subset of MAG to the MAG

and dblp dataset.

Table 6.7: CPU training time in minutes. The training time for MLP is reported in seconds. Training time is measured over five different train/test splits with three runs per split.

	iAF1260b	**dblp**
HyperSAGE	85	N/A
HyperGCN	55	244
HGNN	15	283
HyperSAGNN	75	8,526
MDHNN	27	1,006
NHP	71	197
MLP	15	819

Table 6.7 shows the training time of the different models for one metabolic and one co-citation dataset. The training time is measured over the model testing phase. It can clearly be seen that the training time of MLP, HGNN and MDHNN have the largest increase. The training time of HGNN and MDHNN grow relatively fast because no batching/sampling is applied and each hyperedge is represented by a quadratic number of edges. MLP has a large increase because the data is handled inefficiently. The indices of the data points used in each split and fold are not stored, but the full data is stored. Hence, the training time could be decreased by handling the data more efficiently.

Directed vs undirected

To determine if including the directed information in the model has effect on the performance, we also compared a directed version of HGNN to the undirected version of HGNN. HGNN is made directed by considering a directed clique expansion as follows:

$$\mathbf{X}^{(l+1)} = \sigma\left(\mathbf{D}^{-1}\mathbf{S}\mathbf{X}^{(l)}\mathbf{P}^{l}\right),$$

where

$$\mathbf{S} = \mathbf{H_{head}}\mathbf{W}\mathbf{B}^{-1}\mathbf{H_{tail}}^{T} + \mathbf{H_{head}}\mathbf{W}\mathbf{B}^{-1}\mathbf{H_{head}}^{T} + \mathbf{H_{tail}}\mathbf{W}\mathbf{B}^{-1}\mathbf{H_{tail}}^{T}$$

$$\mathbf{D} = (D_{ii}),\ \text{where } D_{ii} = \sum_{e=1}^{m} W_{ee} H_{ie}^{tail} + W_{ee} H_{ie}^{head},\ \text{is a } n \times n \text{ vertex degree matrix,}$$

$$\mathbf{B} = (B_{ee}),\ \text{where } B_{ee} = \sum_{i=1}^{n} H_{ie}^{tail} + H_{ie}^{head},\ \text{is a } m \times m \text{ hyperedge degree matrix,}$$

$$\mathbf{W} = (W_{ee}) \text{ is a } m \times m \text{ hyperedge weight matrix.}$$

$\mathbf{H_{tail}}$ and $\mathbf{H_{head}}$ are the incidence matrices of the tail and head respectively.

Which is the vector form of

$$x_i^{(l)} = \sigma\left(\sum_{j=1}^{n}\sum_{e=1}^{m} \frac{H_{ie}^{tail} H_{je}^{head} W_{ee}}{B_{ee} D_{ii}} + \frac{H_{ie}^{tail} H_{je}^{tail} W_{ee}}{B_{ee} D_{ii}} + \frac{H_{ie}^{head} H_{je}^{head} W_{ee}}{B_{ee} D_{ii}} x_j^{(l)} \mathbf{P}\right).$$

Table 6.8: AUC scores of directed and non-directed HGNN on the different networks. Non-directed HGNN refers to the original HGNN architecture. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split.

	HGNN	HGNN-d
iAF692	**0.796 ± 0.030**	0.787 ± 0.015
iAF1260b	**0.902 ± 0.007**	0.897 ± 0.005
iJO1366	0.894 ± 0.005	**0.933 ± 0.079**
dblp	0.830 ± 0.027	**0.973 ± 0.057**
MAG	0.906 ± 0.011	**0.952 ± 0.011**

Table 6.8 shows that for larger datasets adding the direction in the models adds a gain in performance of about 5 percent. The smaller datasets do most likely not have enough data such that it can learn the direction as well.

Modules of Modular Directed Hypergraph Neural Network

The performance of individual modules, as introduced in Chapter 3, in Table 6.9 is higher than the modules combined together. This could be an effect of the size of the dataset. There might not be enough data to learn all the relations. Another option is that weight sharing between the different modules has a negative effect. Because of this, individual optimal embeddings cannot be learned.

Parameter sensitivity

Moreover, we also show the sensitivity of MDHNN embedding dimension on the dblp dataset in Figure 6-1. We can conclude that the performance of MDHNN is relatively stable within a range of embedding dimensions and the performance drops when the embedding dimension is too small. Besides, we also have shown in Figure 6-2 that MDHNN still can learn meaningful embeddings if the training set is small. Naturally, the performance drops, but it only decays linearly.

Table 6.9: AUC scores of different modules of MDHNN on the dblp dataset. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split.

Module	**AUC**	**Recall@10**	**Precision@10**
Directed graph	0.938 ± 0.004	0.003 ± 0.001	0.633 ± 0.114
Induced tail hypergraph	0.923 ± 0.002	0.004 ± 0.0003	0.953 ± 0.062
Induced head hypergraph	0.950 ± 0.002	0.005 ± 0.0001	0.993 ± 0.025

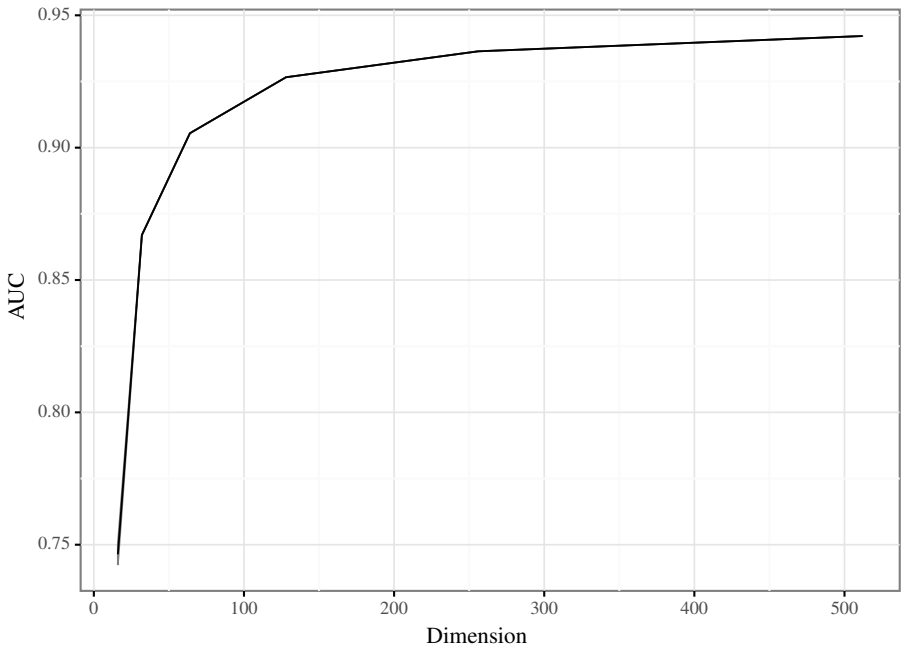


Figure 6-1: AUC scores of MDHNN over embedding dimension. The mean and standard deviation bands over five different train/test splits with three runs per split have been plotted.

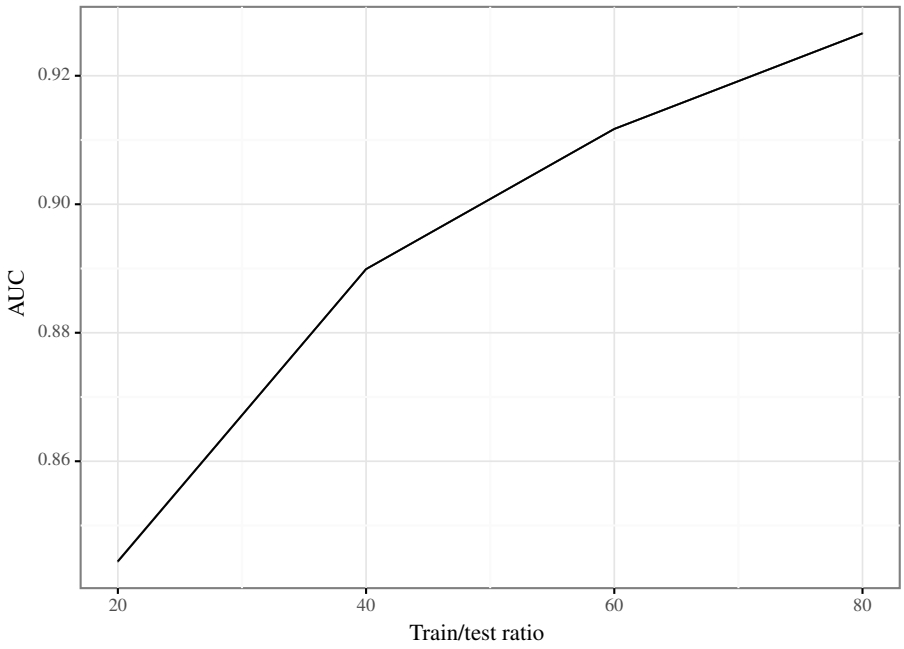


Figure 6-2: AUC scores of MDHNN over train/test ratio. The mean and standard deviation bands over five different train/test splits with three runs per split have been plotted.

ING

Table 6.10: Performance on ING's dataset from February 2021. The dataset has 4,551 edges of which 25 are not derived.

	HGNN	***MDHNN***
AUC	0.915 ± 0.006	0.904 ± 0.007
Probability test non-derived	0.395 ± 0.438	0.503 ± 0.465
Probability train non-derived	0.651 ± 0.461	0.980 ± 0.104
Probability random	0.319 ± 0.267	0.022 ± 0.138

Table 6.11: Performance on ING's dataset from July 2021. The dataset has 3,845 edges of which 29 are not derived.

	HGNN	***MDHNN***
AUC	0.976 ± 0.002	0.961 ± 0.005
Probability test non-derived	0.311 ± 0.422	0.418 ± 0.365
Probability train non-derived	0.474 ± 0.486	0.950 ± 0.105
Probability random	0.113 ± 0.146	0.014 ± 0.092

We have also evaluated two hypergraph neural networks on ING's dataset. As Tables 6.10 and 6.11 show, the performance is very similar to the performance in Section 4.1. The probabilities for the non-derived edges are low. Combining this with the fact that the AUC is high, we can conclude that the ranking of non-derived edges is low and thus no relevant predictions can be made.

Table 6.11 shows the performance of a recent version of the dataset. For this hypergraph, the number of edges has decreased from 4,551 to 3,845. Moreover, the number of non-derived edges has increased from 25 to 29. A comparison between Tables 6.10 and 6.11 show that the AUC has increased, but the probabilities have decreased. Since the structure of the hypergraph has changed a lot, it is not clear why the performance changes. It could be that the change in structure makes it harder to make predictions for the non-derived edges. Another option is that the new non-derived edges are hard instances.

6.3 Threats to validity

In this thesis, the alternative datasets chosen are not most favorable. The datasets are from another domain than the original dataset. It would have been preferred to have a dataset which has a higher similarity. However, measuring the similarity is a hard task. Hypergraph similarity can be measured by many different approaches. Moreover, we used a simple approach to determine the similarities of the graphs. The usage of complex algorithms would not have been possible due to computational limitations. In the end, the best alternative dataset would have similar features of the graph for the features that are important for making predictions. However, these features are unknown. Furthermore, the alternative datasets have different sizes than the original dataset.

Another limitation is that we do not use grid search for the hyperparameter search. If we would have used grid search, then the full hyperparameter search space would have been explored and then we could conclude that the difference in performance is due to the difference in architecture. However, it is too computationally expensive. The one thing we do to make Bayesian optimization as similar as possible is to set the random seed. In this case, models with the same hyperparameter search space will start in the same point. But still this starting point could be more optimal for a certain model.

Finally, the sizes of the alternative datasets are limited. Therefore, we cannot conclude how the models perform on larger datasets. If we would have faster machines or more time to conduct the experiments, the last two limitations could be diminished.

Chapter 7

Conclusions and future work

7.1 Conclusions

Hyperlink prediction as a field of research is still emerging. The majority of the proposed hyperlink prediction models are designed for undirected hypergraphs. Directed hyperlink prediction is underexplored. The model that has been proposed for directed hyperlink prediction, i.e. NHP, only uses the direction of the edges in the training procedure by leveraging a direction scoring layer. Moreover, NHP also does not use the structure of the graph to make predictions. Hence, a model that uses the directions and the structure of the hypergraph as information to make predictions has not been proposed yet. To address those challenges, we proposed MDHNN, a model which takes the direction of the edges and the structure of the hypergraph into account when generating node embeddings. Therefore, this can generate more informative node embeddings. Those node embeddings are useful for the problem of directed hyperlink prediction as emerged within ING. Furthermore, we have compared different baseline models for directed hyperlink prediction under a similar setting. This gave an overview of the performance of different hypergraph link prediction models since they have not all been compared prior to this thesis.

The dataset provided by ING is too small to get reasonable results. Therefore, we evaluated the different models on open source datasets. The five datasets used are:

three metabolic datasets and two co-citation datasets (dblp and MAG). By comparing the different networks, we have concluded that the dblp dataset is most similar to ING's dataset.

The evaluation of the different models showed that MDHNN overall has good performance. MDHNN is the best performing model for the dblp dataset. For the other datasets, NHP has the best performance. Since NHP is a structure-agnostic model, this shows that only with node information we can get optimal performance. Therefore, we argue that models that use the structure of the graph to make predictions do not fully exploit this structure yet. If we compare the methods that use the structure, we can conclude that both MDHNN and HyperSAGE have the highest performance. However, HyperSAGE is too computationally expensive and thus not feasible for most real world networks. Moreover, from the comparison of the directed and non-directed HGNN we can conclude that incorporating the direction in a model will help in performance for larger datasets.

7.2 Future work

Since the expectation is that ING's hypergraph will grow over time, we recommend to test MDHNN and NHP on the dataset once it is richer. We recommend to evaluate both MDHNN and NHP because those are the best two models for the different datasets. MDHNN has the best performance for the dblp dataset. However, it cannot be concluded with certainty that this will also have the best performance on ING's dataset. Since the small dataset in this thesis has about 600 edges, it is recommended to evaluate the performance once the dataset has hundreds of non-derived edges. According to the domain expert, it is expected that the graph will grow this big. Since the derived edges are not relevant, it is recommended to discard those and encode the information provided by those edges in the node features, e.g. indicating which nodes are directly connected. Keeping the derived edges will make the problem unbalanced since they are regarded as relevant, e.g. belong to the positive class.

In general, it is recommended to evaluate the performance of ING's dataset by including meaningful node features. Before doing this, it has to be determined which features might be added to the nodes such that they improve the quality of the models. Besides, once the directed hypergraph is turned into a weighted directed hypergraph, it is recommended to re-train the model.
If a final model is trained, a search procedure needs to be developed which takes the hyperlink prediction function as the objective function. This will enable ING to make suggestions for hyperlinks to introduce into the domain model.

Since most real world networks are large and hypergraph neural networks can be computationally expensive, it is of great interest to research batchable hypergraph neural networks. Furthermore, HyperSAGE has good performance as well. Therefore, it is valuable to research the possibility of an optimized sampling procedure such that it is feasible for larger graphs.
Finally, to get better insights in MDHNN, the performance of the different models on a large ($>$ 100.000 edges) hypergraph can be compared. Moreover, MDHNN with non-shared weights could be evaluated to determine if the performance would increase. It should be noted that if MDHNN is trained with non-shared weights the model is more prone to overfitting since the number of parameters to fit almost triple in size. As a result, the training time will increase. Additionally, an attention mechanism could be explored for aggregating the different embeddings of the different modules. Besides, the models can be compared on a node classification task. The results of this will confirm if the models also perform well in another setting.

Appendix A

ING's experiment details

The model that was used for ING's experiment has already been explained. In this appendix, the preparation of the data and the hyperparameter search will be described.

A.1 Data preparation

The two classes present in a hyperlink prediction problem , i.e. positive and negative edges, are highly unbalanced since graphs are sparse. Hence sampling of negative edges needs to be done. We have developed the following sampling methods with domain knowledge:

- `random`: randomly pick a sample of vertices of size between the smallest and largest hyperedge and of this random sample pick one vertex as the output.
- `half random`: randomly pick a positive edge and randomly change *per* percentage of the nodes in the edge.
- `path`: create edges from shortest paths in the directed or undirected clique expansion. The output is the last node on the path.
- `walk`: create edges from random walks in the directed or undirected clique expansion. The output is the last node on the walk.

- `component:` create edges for induced subgraph for a node in undirected clique expansion of each node and its neighbors. The output is the considered node.

Since node2vec is transductive and we will split the data in a train and test set, we need to make sure that all nodes are present among the positive edges in the train set. The following two methods to split positive edges have been evaluated:

- `normal`: makes sure that all nodes are present in the train set, i.e. do not introduce isolated nodes when removing the test set.
- `connected`: makes sure that the number of connected components remains unchanged.

A.2 Hyperparameter search

Many hyperparameters in this model could be optimized. However, we kept some of the hyperparameters fixed:

- number of random walks in node2vec: 10,
- length of random walks in node2vec: 80,
- number of hidden layers in NN: 1,
- number of nodes in hidden layer: 128.

The hyperparameter search space is:

- methods to generate negative edges: [`random`, `random half`, `path`, `walk`, `component`],
- *per* in `random half`: [0.3, 0.5, 0.7],
- directed clique expansions in path and walk: [True, False],
- ratio of negative edges: [1, 1.5, 2, 3],
- test set size: [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8],

- dimension of node embeddings: [32, 64, 128, 256],
- aggregation function for node embeddings = [min, mean, max, maxmin].

The hyperparameters have been optimized using cross-validation over 5 folds. Besides, we have also use sample weighting to give higher weights to the non-derived edges.

Appendix B

Model selection and testing algorithms

Algorithm 1: Model Selection(TF, M, t)
Input: Training folds TF, model M that you want to evaluate over the number of trials t.
Output: Optimal hyperparameter set θ_{opt}.

```
1  AUCs_θ := {} ;
2  for i := 0 to t do
3      AUCs := {} ;
4      Pick next hyperparameter set θ ;
5      for tf in TF do
6          AUC_avg := Perform CV on tf for configuration θ and model M ;
7          AUCs := AUCs ∪ AUC_avg ;
8      AUCs_θ := AUCs_θ ∪ avg(AUCs) ;
9  θ_opt := argmax_θ(AUCs_θ) ;
10 return θ_opt
```

Algorithm 2: Model Testing(D, M, θ)

Input: Dataset D split into training trf and testing tef fold, model M that you want to test for hyperparameter set θ over k different runs.

Output: Performance p.

```
perfs := {} ;
for (trf, tef) in D do
    for i := 0, ..., k do
        set_seed(i) ;
        model := Train(M, trf, θ) ;
        perf := Test(model, tef) ;
        perfs := perfs ∪ perf ;
p := avg(perfs) ;
return p
```

Appendix C

Violin plots hyperedge degrees and sizes

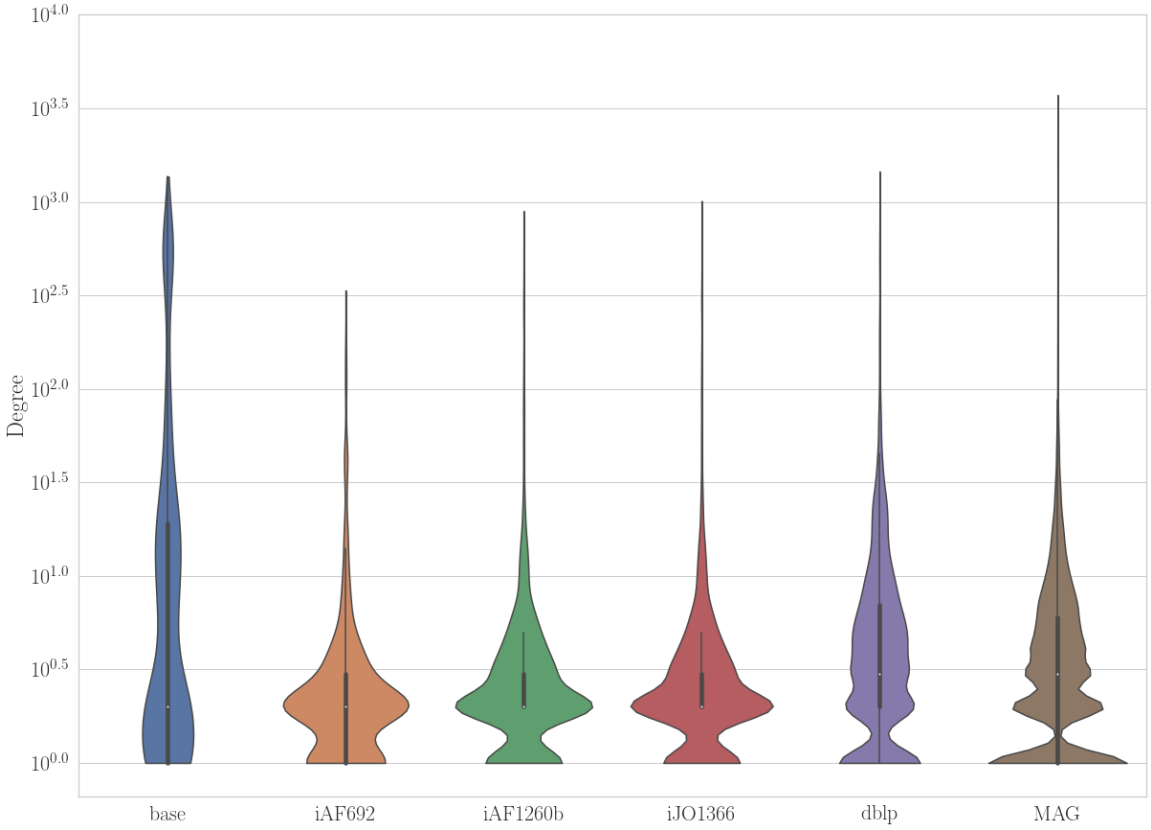


Figure C-1: Hyperedge degree distribution of the different datasets.

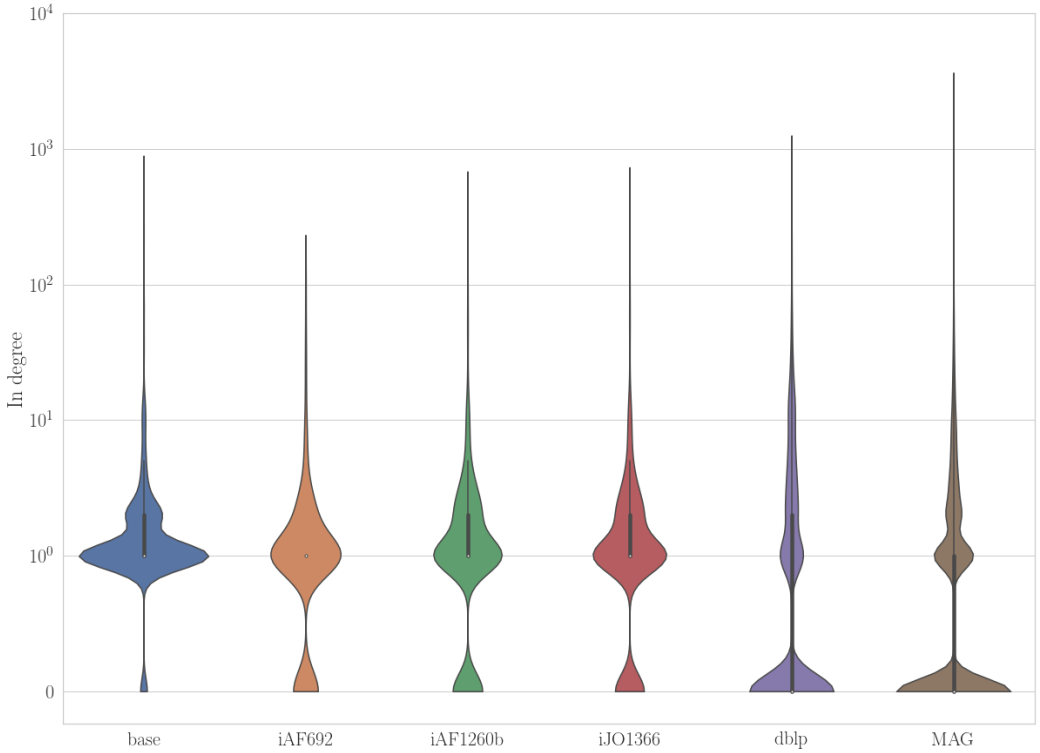


Figure C-2: Hyperedge in-degree distribution of the different datasets.

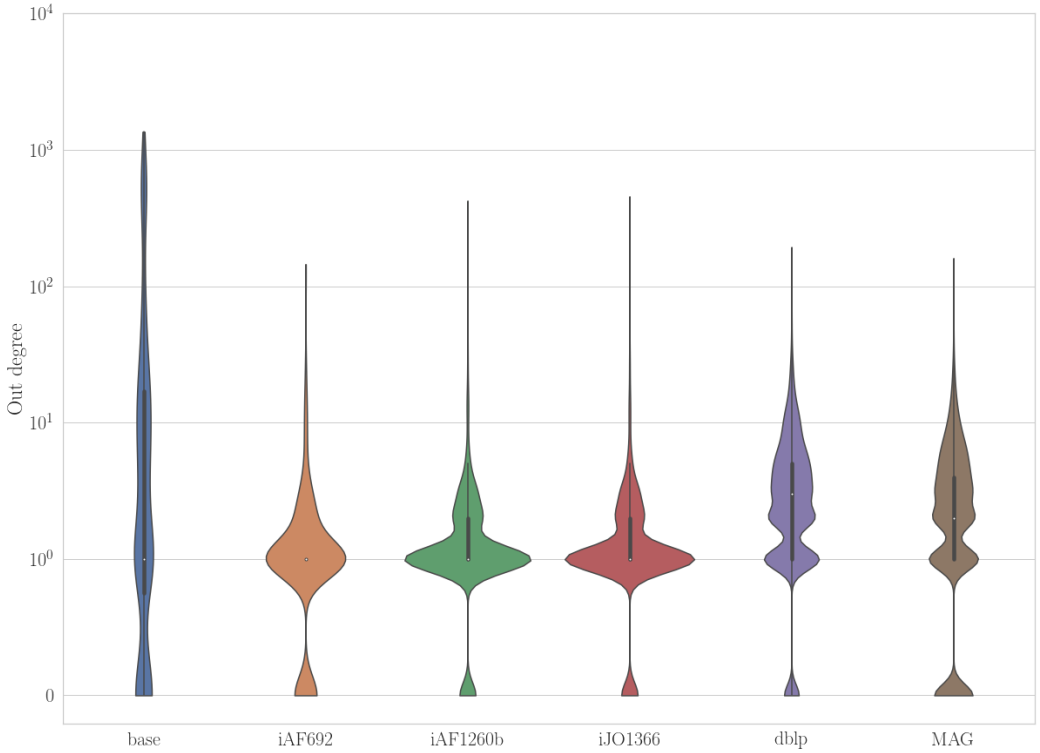


Figure C-3: Hyperedge out-degree distribution of the different datasets.

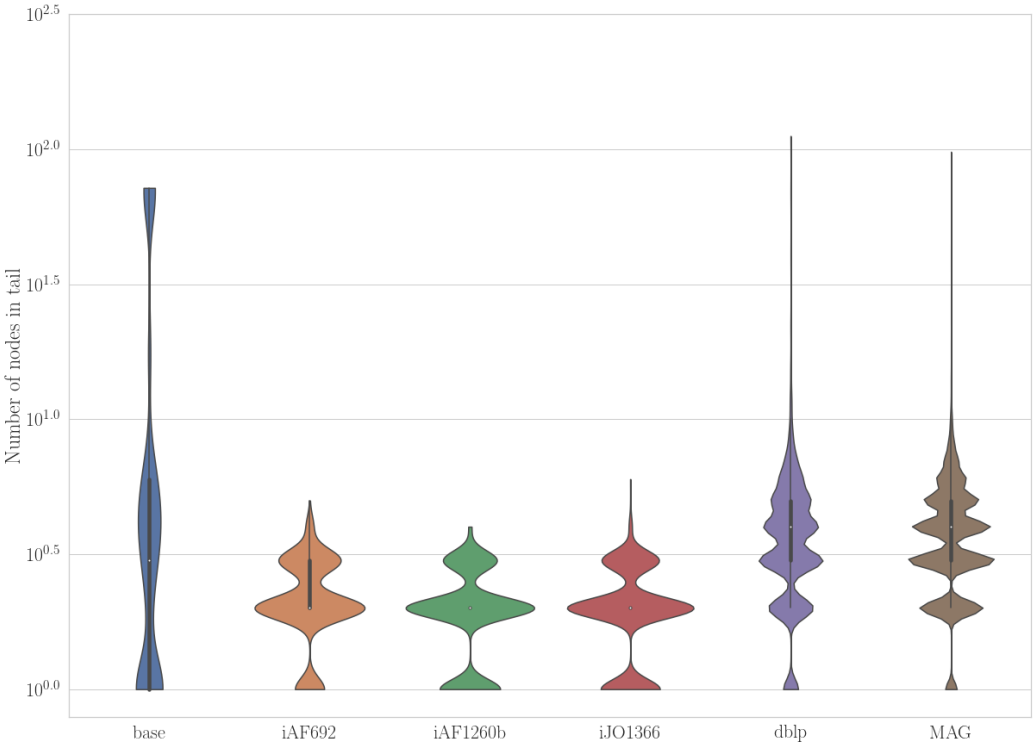


Figure C-4: Tail size distribution of the different datasets.

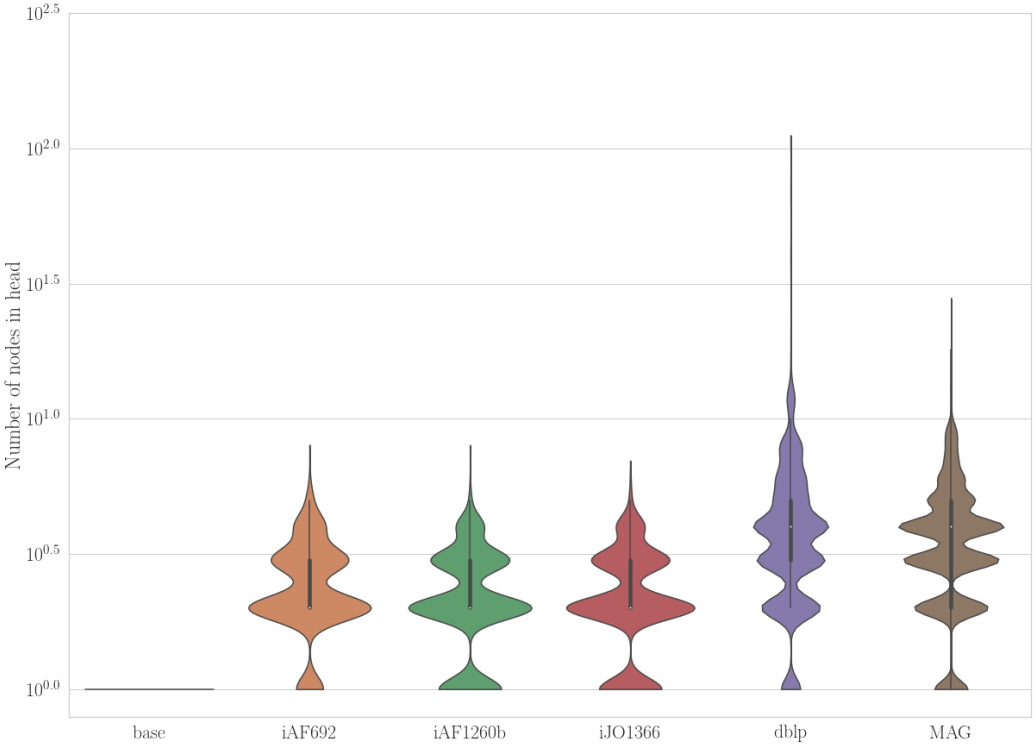


Figure C-5: Head size distribution of the different datasets.

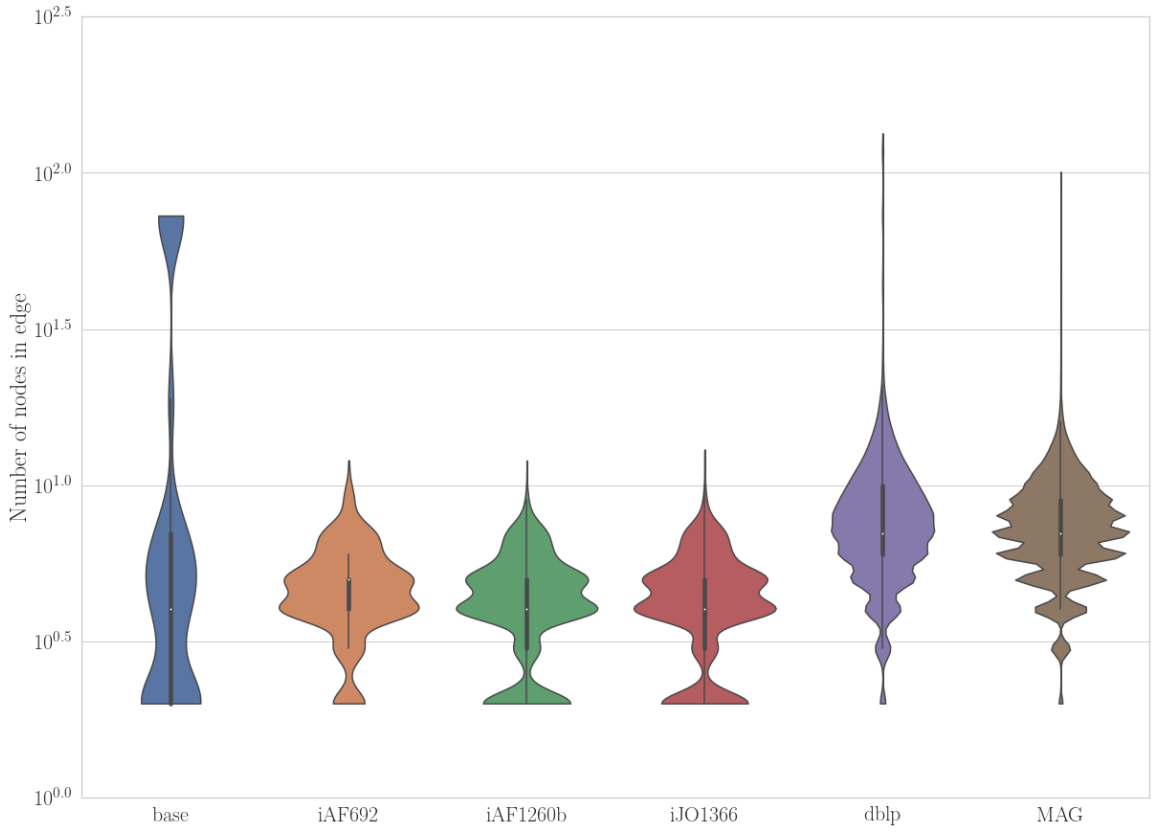


Figure C-6: Edge size distribution of the different datasets.

Appendix D

Hyperparameter search spaces

The table can be found on the next page.

Hyperparameter	**HyperGCN**	**HGNN**	**HyperSAGNN**	**NHP**	**HyperSAGE**	**MDHNN**
Number of epochs = {1,300}	x	x	x	x	x	x
Learning rate = {1e-6, 0.4}	x	x	x	x	x	x
L_2 regularization parameter = {0, 5e-4}	x	x	x	x	x	x
Embedding size [a] = {16, 32, 64, 128, 256, 512 }	x	x	x	x	x	x
Number of layers = {1, 2, 3}	x	x			x	x
Aggregation = {mean, maxmin}	x	x	x	x	x	x
Dropout = {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8}	x	x		x	x	x
Hidden aggregation = {mean, max, min, concat}						x
Power = {-1, 0.01, 1, 2, 3, 5}					x	
Sample = {2, 3, 5, 10, 15}					x	
Diagonal mask = {True, False}			x			
Number of heads = {1, 2, 4, 6, 8, 10, 12, 14, 16 }			x			
Mediators = {True, False}	x					

	MLP
Number of epochs = {1,300}	x
Learning rate = {1e-6, 0.4}	x
L_2 regularization parameter = {0, 5e-4}	x
Number of layers = {1, 2, 3}	x
Batch normalization = {True, False}	x
Dropout = {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8}	x

[a]For HyperSAGNN, embedding size space is used for both d_k and d_v.

Appendix E

Optimal hyperparameters

Table E.1: Optimal hyperparameters for HyperGCN for the different datasets. MAG represents the subset generated from the MAG dataset.

	iAF692	**iAF1260b**	**iJO1366**	**MAG**
Number of epochs	107	213	213	92
Learning rate	2.428e-02	2.977e-03	2.977e-03	0.0251
L_2 **regularization**	1.705e-04	1.225e-05	1.225e-05	4.359e-04
Embedding size	256	64	64	256
Number of layers	3	3	3	1
Aggregation	mean	mean	mean	maxmin
Dropout	0.7	0.2	0.2	0.6
Mediators	True	True	True	True

Table E.2: Optimal hyperparameters for HGNN for the different datasets. MAG represents the subset generated from the MAG dataset.

	iAF692	**iAF1260b**	**iJO1366**	**MAG**
Number of epochs	202	202	202	140
Learning rate	3.721e-04	3.721e-04	3.721e-04	2.721e-02
L_2 **regularization**	1.302e-04	1.302e-04	1.302e-04	3.783e-04
Embedding size	512	512	512	256
Number of layers	1	1	1	1
Aggregation	mean	mean	mean	maxmin
Dropout	0.4	0.4	0.4	0.8

Table E.3: Optimal hyperparameters for HyperSAGE for the different datasets. MAG represents the subset generated from the MAG dataset.

	iAF692	**iAF1260b**	**iJO1366**	**MAG**
Number of epochs	209	209	209	N/A
Learning rate	1.932e-03	1.932e-03	1.932e-03	N/A
L_2 **regularization**	3.979e-05	3.979e-05	3.979e-05	N/A
Embedding size	64	64	64	N/A
Number of layers	2	2	2	N/A
Aggregation	mean	mean	mean	N/A
Dropout	0.2	0.2	0.2	N/A
Power	0.01	0.01	0.01	N/A
Sample	2	2	2	N/A

Table E.4: Optimal hyperparameters for HyperSAGNN for the different datasets. MAG represents the subset generated from the MAG dataset.

	iAF692	**iAF1260b**	**iJO1366**	**MAG**
Number of epochs	272	205	205	29
Learning rate	1.641e-03	0.0117	0.0117	1.189e-04
L_2 **regularization**	2.737e-05	4.953e-04	4.953e-04	2.841e-04
d_k	64	64	64	512
d_v	32	128	128	128
Aggregation	maxmin	mean	mean	maxmin
Number of heads	8	6	6	10
Diag	False	False	False	True

Table E.5: Optimal hyperparameters for NHP for the different datasets. MAG represents the subset generated from the MAG dataset.

	iAF692	**iAF1260b**	**iJO1366**	**MAG**
Number of epochs	250	250	250	267
Learning rate	0.0186	0.0186	0.0186	2.854e-03
L_2 **regularization**	1.578e-04	1.578e-04	1.578e-04	5.769e-05
Embedding size	256	256	256	128
Aggregation	mean	mean	mean	maxmin
Dropout	0.2	0.2	0.2	-

Table E.6: Optimal hyperparameters for MDHNN for the different datasets. MAG represents the subset generated from the MAG dataset.

	iAF692	**iAF1260b**	**iJO1366**	**MAG**
Number of epochs	261	261	261	242
Learning rate	0.161	0.161	0.161	3.079e-04
L_2 **regularization**	1.522e-04	1.522e-04	1.522e-04	2.844e-04
Embedding size	256	256	256	128
Number of layers	1	1	1	2
Aggregation	mean	mean	mean	maxmin
Dropout	0.2	0.2	0.2	0.2
Hidden aggregation	mean	mean	mean	max

Table E.7: Optimal hyperparameters for MLP for the different datasets. MAG represents the subset generated from the MAG dataset.

	iAF692	**iAF1260b**	**iJO1366**	**MAG**
Number of epochs	78	78	78	103
Learning rate	0.0714	0.0714	0.0714	5.762e-05
L_2 **regularization**	4.740e-04	4.740e-04	4.740e-04	2.975e-04
Number of layers	3	3	3	2
Batch normalization	True	True	True	True
Dropout	-	-	-	0.7

Appendix F

Violin plots AUC scores

Violin plots start from the next page.

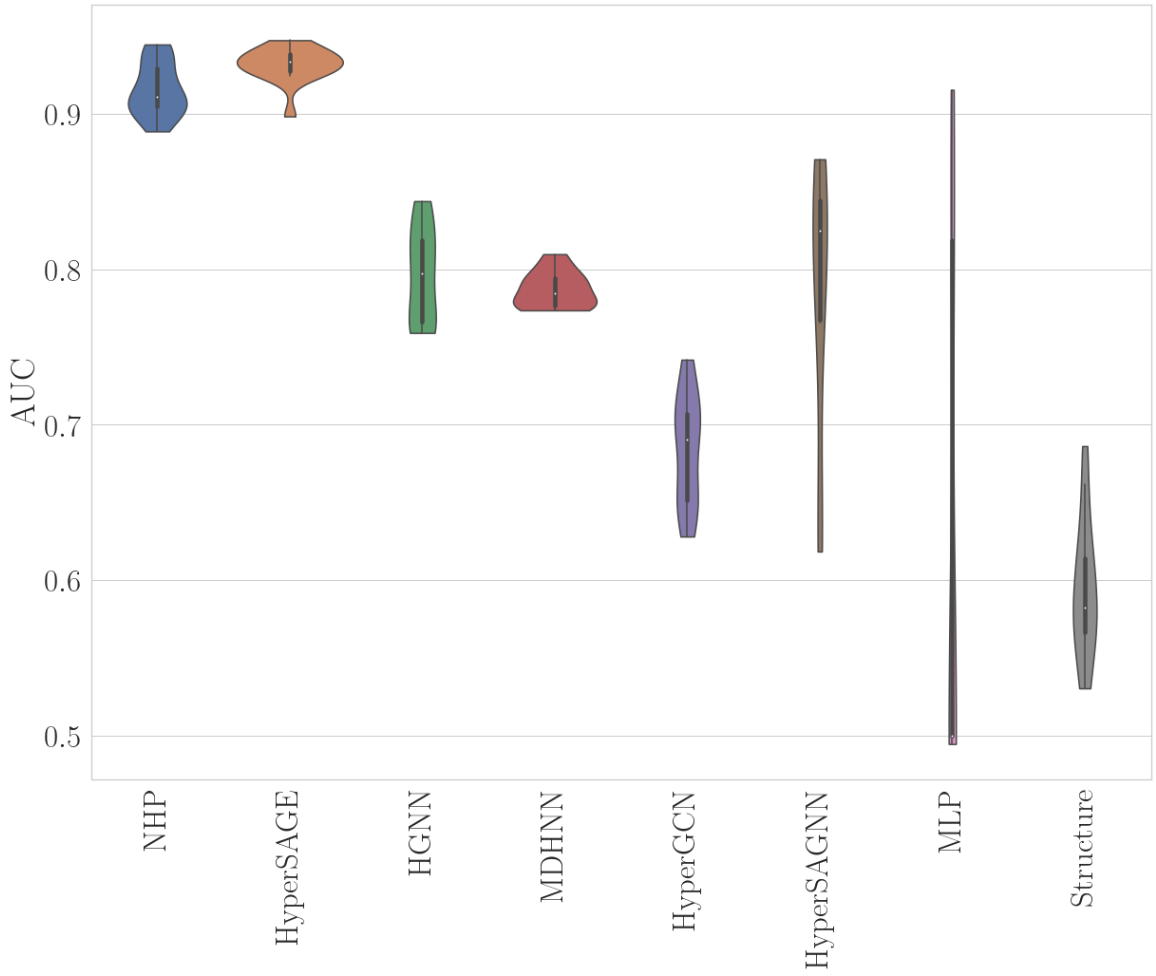


Figure F-1: AUC scores distribution of the different models on the iAF692 metabolic network. AUC scores are calculated over five different train/test splits with three runs per split.

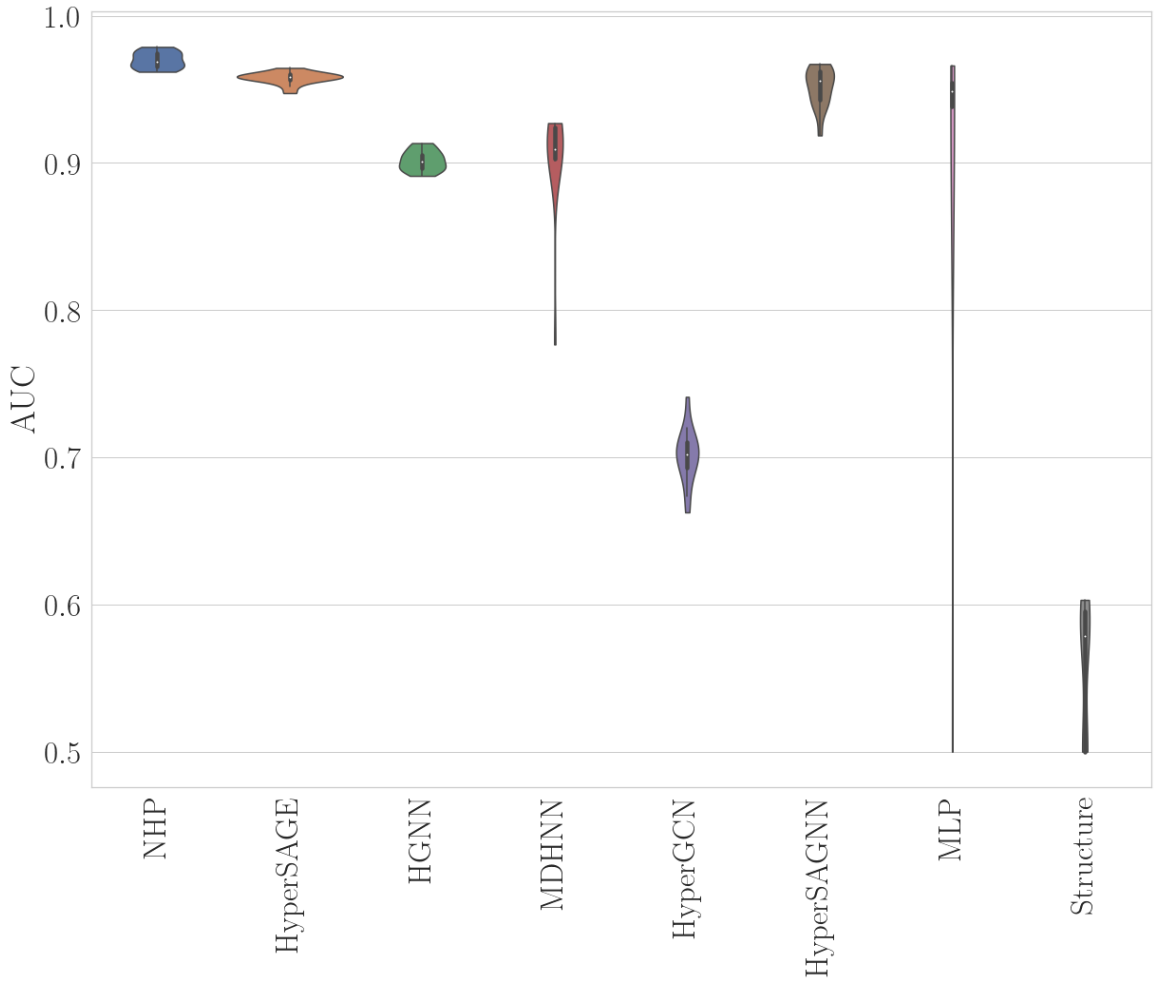


Figure F-2: AUC scores distribution of the different models on the iAF1260b metabolic network. AUC scores are calculated over five different train/test splits with three runs per split.

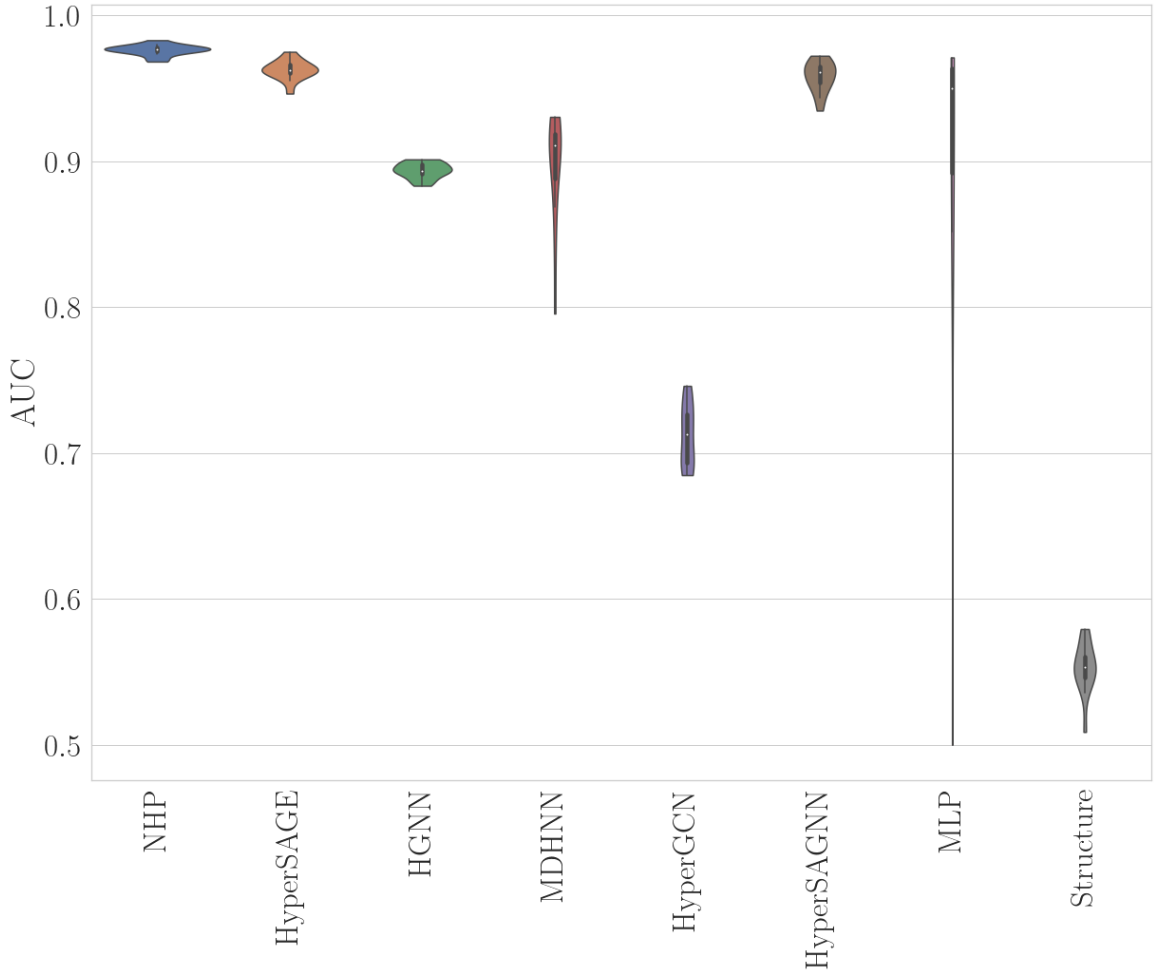


Figure F-3: AUC scores distribution of the different models on the iJO1366 metabolic network. AUC scores are calculated over five different train/test splits with three runs per split.

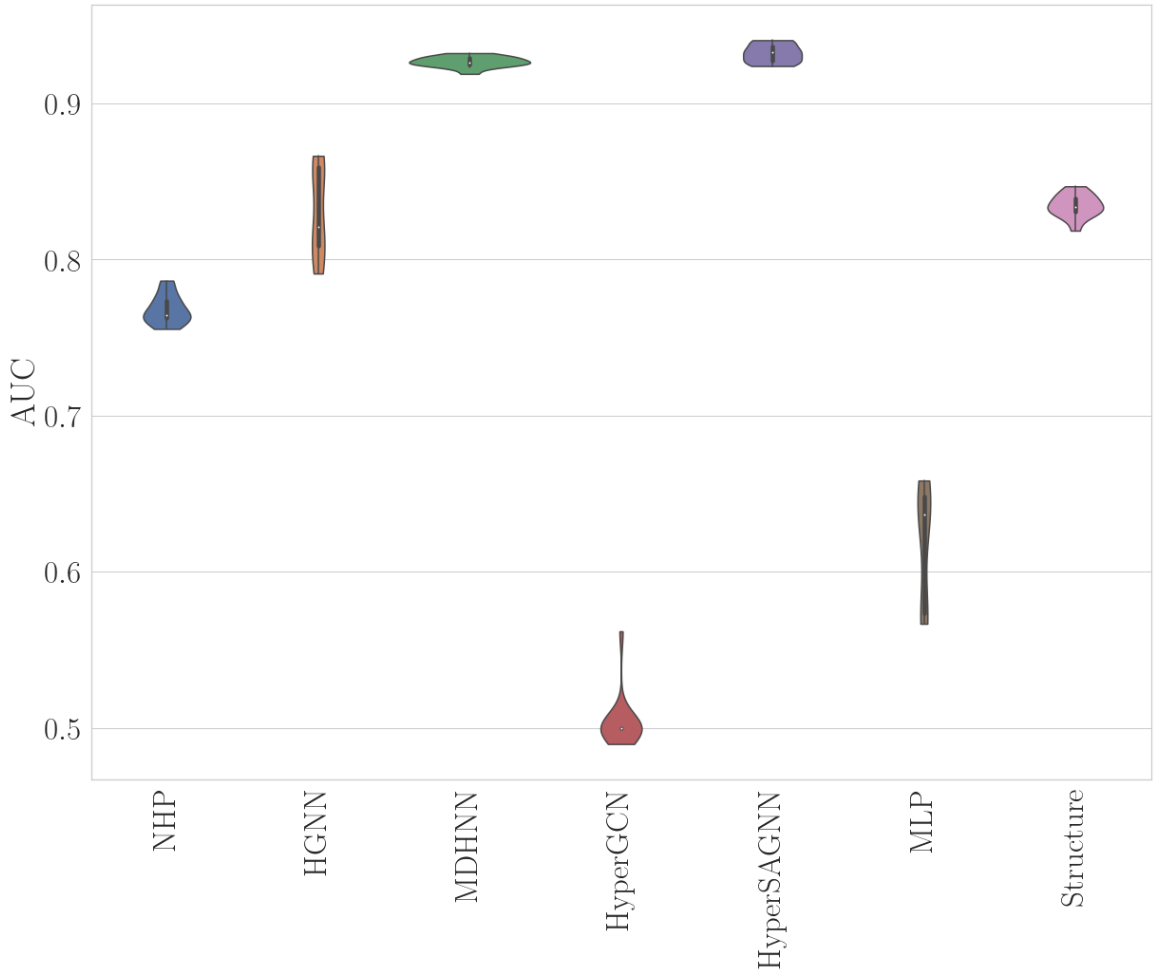


Figure F-4: AUC scores distribution of the different models on the dblp dataset. AUC scores are calculated over five different train/test splits with three runs per split.

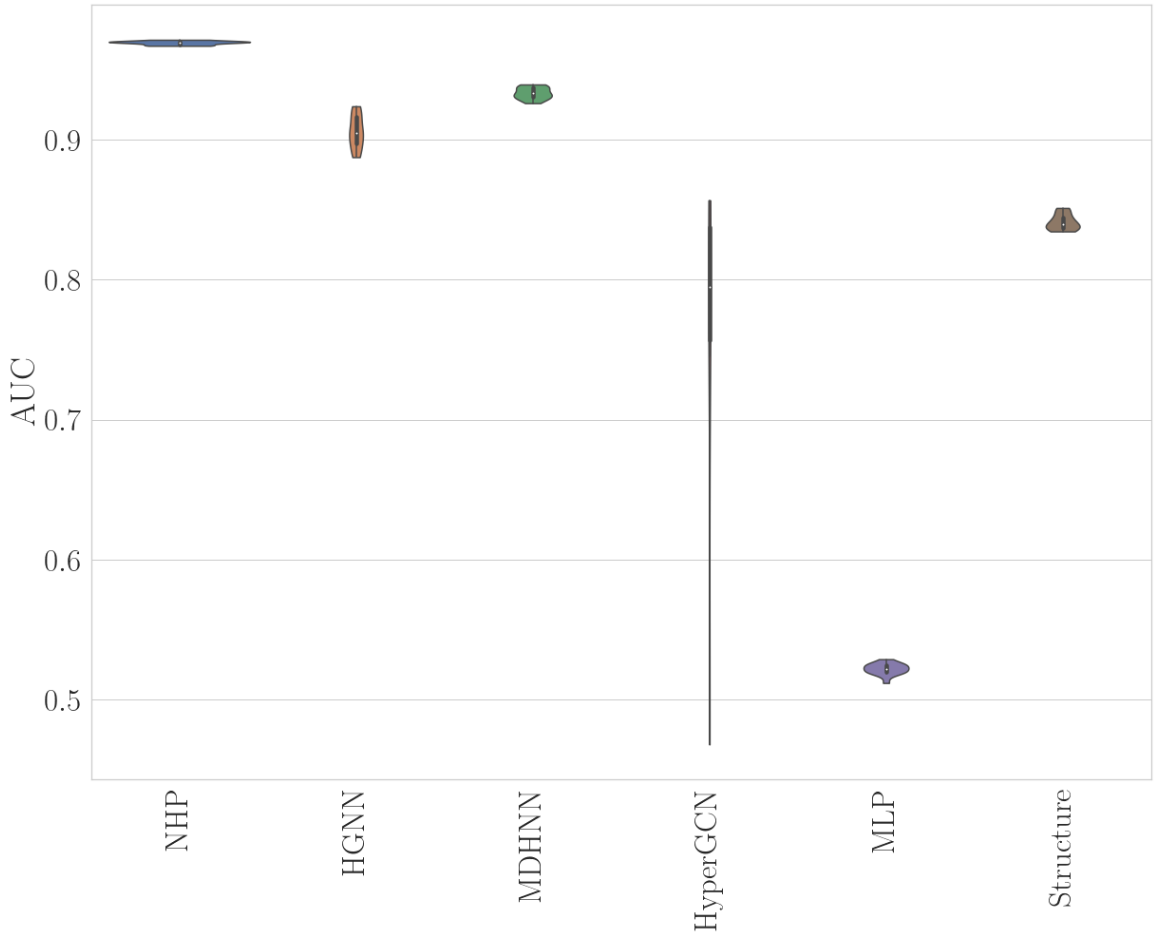


Figure F-5: AUC scores distribution of the different models on the MAG dataset. AUC scores are calculated over five different train/test splits with three runs per split.

Appendix G

Full recall and precision scores

Table G.1: Full recall@k results of the different models on the iAF692 metabolic network. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split. Half has a size of 62 edges.

	Recall@half	Recall@100	Recall@50	Recall@10
HyperSAGE	**0.456 ± 0.012**	0.710 ± 0.016	0.374 ± 0.008	0.075 ± 0.006
HyperGCN	0.298 ± 0.026	0.468 ± 0.039	0.243 ± 0.024	0.051 ± 0.009
HGNN	0.380 ± 0.019	0.559 ± 0.030	0.313 ± 0.016	0.068 ± 0.009
HyperSAGNN	0.344 ± 0.062	0.535 ± 0.088	0.281 ± 0.048	0.064 ± 0.011
MDHNN	0.369 ± 0.020	0.567 ± 0.024	0.308 ± 0.021	0.067 ± 0.009
NHP	0.432 ± 0.020	**0.813 ± 0.029**	**0.440 ± 0.021**	**0.092 ± 0.009**
MLP	0.286 ± 0.118	0.454 ± 0.161	0.234 ± 0.096	0.047 ± 0.024
Structure	0.252 ± 0.030	0.383 ± 0.034	0.205 ± 0.027	0.041 ± 0.016

Table G.2: Full precision@k results of the different models on the iAF692 metabolic network. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split. Half has a size of 62 edges.

	Precision@half	Precision@100	Precision@50	Precision@10
HyperSAGE	**0.912 ± 0.023**	**0.880 ± 0.019**	**0.927 ± 0.019**	**0.927 ± 0.077**
HyperGCN	0.597 ± 0.052	0.580 ± 0.048	0.603 ± 0.060	0.633 ± 0.114
HGNN	0.760 ± 0.038	0.693 ± 0.037	0.777 ± 0.040	0.847 ± 0.109
HyperSAGNN	0.687 ± 0.124	0.663 ± 0.109	0.696 ± 0.120	0.793 ± 0.139
MDHNN	0.738 ± 0.041	0.704 ± 0.030	0.763 ± 0.054	0.840 ± 0.114
NHP	0.867 ± 0.041	0.800 ± 0.029	0.865 ± 0.042	0.907 ± 0.085
MLP	0.572 ± 0.236	0.563 ± 0.200	0.581 ± 0.238	0.580 ± 0.295
Structure	0.503 ± 0.060	0.475 ± 0.043	0.508 ± 0.068	0.507 ± 0.202

Table G.3: Full recall@k results of the different models on the iAF1260b metabolic network. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split. Half has a size of 206 edges.

	Recall@half	Recall@100	Recall@50	Recall@10
HyperSAGE	0.471 ± 0.006	0.231 ± 0.005	0.117 ± 0.003	0.023 ± 0.001
HyperGCN	0.309 ± 0.015	0.156 ± 0.005	0.076 ± 0.006	0.014 ± 0.003
HGNN	0.458 ± 0.005	0.232 ± 0.003	0.118 ± 0.003	0.024 ± 0.000
HyperSAGNN	0.472 ± 0.010	0.233 ± 0.005	0.117 ± 0.003	0.023 ± 0.001
MDHNN	0.444 ± 0.031	0.220 ± 0.016	0.112 ± 0.006	0.022 ± 0.002
NHP	**0.482 ± 0.009**	**0.292 ± 0.006**	**0.146 ± 0.004**	**0.029 ± 0.001**
MLP	0.445 ± 0.010	0.218 ± 0.045	0.109 ± 0.024	0.021 ± 0.005
Structure	0.222 ± 0.016	0.103 ± 0.010	0.051 ± 0.009	0.010 ± 0.004

Table G.4: Full precision@k results of the different models on the iAF1260b metabolic network. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split. Half has a size of 206 edges.

	Precision@half	Precision@100	Precision@50	Precision@10
HyperSAGE	0.943 ± 0.013	0.952 ± 0.020	0.964 ± 0.023	0.960 ± 0.061
HyperGCN	0.620 ± 0.030	0.644 ± 0.019	0.625 ± 0.050	0.567 ± 0.125
HGNN	0.919 ± 0.010	0.960 ± 0.013	**0.972 ± 0.023**	**1.000 ± 0.000**
HyperSAGNN	0.947 ± 0.021	**0.964 ± 0.021**	0.965 ± 0.027	0.960 ± 0.049
MDHNN	0.890 ± 0.062	0.907 ± 0.067	0.924 ± 0.052	0.927 ± 0.085
NHP	**0.964 ± 0.018**	**0.964 ± 0.019**	0.965 ± 0.027	0.973 ± 0.044
MLP	0.891 ± 0.198	0.901 ± 0.186	0.900 ± 0.197	0.873 ± 0.198
Structure	0.445 ± 0.033	0.424 ± 0.043	0.417 ± 0.073	0.407 ± 0.148

Table G.5: Full recall@k results of the different models on the iJO1366 metabolic network. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split. Half has a size of 223 edges.

	Recall@half	Recall@100	Recall@50	Recall@10
HyperSAGE	0.475 ± 0.011	0.215 ± 0.006	0.108 ± 0.002	0.021 ± 0.001
HyperGCN	0.321 ± 0.023	0.150 ± 0.012	0.076 ± 0.010	0.015 ± 0.004
HGNN	0.451 ± 0.012	0.212 ± 0.007	0.108 ± 0.002	0.022 ± 0.001
HyperSAGNN	0.476 ± 0.008	0.216 ± 0.004	0.107 ± 0.003	0.022 ± 0.001
MDHNN	0.439 ± 0.029	0.205 ± 0.013	0.104 ± 0.005	0.022 ± 0.001
NHP	**0.486 ± 0.005**	**0.272 ± 0.004**	**0.138 ± 0.003**	**0.028 ± 0.000**
MLP	0.448 ± 0.097	0.202 ± 0.045	0.101 ± 0.022	0.020 ± 0.004
Structure	0.209 ± 0.012	0.085 ± 0.009	0.039 ± 0.008	0.006 ± 0.004

Table G.6: Full precision@k results of the different models on the iJO1366 metabolic network. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split. Half has a size of 223 edges.

	Precision@half	Precision@100	Precision@50	Precision@10
HyperSAGE	0.953 ± 0.021	0.963 ± 0.025	0.968 ± 0.022	0.953 ± 0.062
HyperGCN	0.643 ± 0.046	0.671 ± 0.053	0.679 ± 0.088	0.673 ± 0.157
HGNN	0.904 ± 0.023	0.948 ± 0.033	0.968 ± 0.022	0.967 ± 0.047
HyperSAGNN	0.954 ± 0.016	0.965 ± 0.017	0.957 ± 0.029	0.967 ± 0.060
MDHNN	0.880 ± 0.059	0.918 ± 0.060	0.925 ± 0.044	0.973 ± 0.057
NHP	**0.973 ± 0.011**	**0.972 ± 0.016**	**0.981 ± 0.020**	**1.000 ± 0.000**
MLP	0.898 ± 0.193	0.903 ± 0.200	0.899 ± 0.201	0.873 ± 0.181
Structure	0.418 ± 0.025	0.381 ± 0.039	0.351 ± 0.072	0.267 ± 0.174

Table G.7: Full recall@k results of the different models on the dblp dataset. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split. Half has a size of 1110 edges.

	Recall@half	Recall@100	Recall@50	Recall@10
HyperSAGE	N/A	N/A	N/A	N/A
HyperGCN	0.202 ± 0.018	0.018 ± 0.003	0.009 ± 0.002	0.002 ± 0.001
HGNN	0.400 ± 0.016	0.039 ± 0.004	0.020 ± 0.002	0.004 ± 0.001
HyperSAGNN	0.448 ± 0.007	0.041 ± 0.001	0.021 ± 0.001	0.004 ± 0.0005
MDHNN	**0.460 ± 0.003**	**0.044 ± 0.001**	**0.022 ± 0.001**	0.004 ± 0.0003
NHP	0.375 ± 0.009	0.033 ± 0.003	0.013 ± 0.003	0.002 ± 0.001
MLP	0.320 ± 0.034	0.042 ± 0.002	0.021 ± 0.001	**0.005 ± 0.000**
Structure	0.410 ± 0.005	0.035 ± 0.003	0.018 ± 0.002	0.003 ± 0.001

Table G.8: Full precision@k results of the different models on the dblp dataset. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split. Half has a size of 1110 edges.

	Precision@half	Precision@100	Precision@50	Precision@10
HyperSAGE	N/A	N/A	N/A	N/A
HyperGCN	0.404 ± 0.035	0.405 ± 0.062	0.393 ± 0.069	0.413 ± 0.171
HGNN	0.799 ± 0.032	0.875 ± 0.084	0.880 ± 0.102	0.907 ± 0.124
HyperSAGNN	0.895 ± 0.015	0.921 ± 0.028	0.928 ± 0.038	0.913 ± 0.109
MDHNN	**0.921 ± 0.007**	**0.968 ± 0.014**	**0.967 ± 0.027**	0.940 ± 0.071
NHP	0.750 ± 0.018	0.589 ± 0.059	0.453 ± 0.094	0.360 ± 0.125
MLP	0.639 ± 0.068	0.943 ± 0.041	0.948 ± 0.051	**1.000 ± 0.000**
Structure	0.821 ± 0.009	0.787 ± 0.070	0.787 ± 0.099	0.767 ± 0.162

Table G.9: Full recall@k results of the different models on the MAG dataset. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split. Half has a size of 2513 edges.

MAG	Recall@half	Recall@100	Recall@50	Recall@10
HyperSAGE	N/A	N/A	N/A	N/A
HyperGCN	0.374 ± 0.061	0.015 ± 0.003	0.008 ± 0.002	**0.002 ± 0.0003**
HGNN	0.433 ± 0.008	0.018 ± 0.003	0.009 ± 0.001	**0.002 ± 0.0003**
HyperSAGNN	N/A	N/A	N/A	N/A
MDHNN	0.478 ± 0.003	0.020 ± 0.0002	0.010 ± 0.0002	**0.002 ± 0.000**
NHP	**0.491 ± 0.001**	**0.025 ± 0.0001**	**0.012 ± 0.000**	**0.002 ± 0.000**
MLP	0.199 ± 0.009	0.005 ± 0.001	0.003 ± 0.001	0.001 ± 0.0002
Structure	0.404 ± 0.008	0.016 ± 0.001	0.008 ± 0.001	0.002 ± 0.0003

Table G.10: Full precision@k results of the different models on the MAG dataset. Result reported per model is the mean ± standard deviation over five different train/test splits with three runs per split. Half has a size of 2513 edges.

	Precision@half	Precision@100	Precision@50	Precision@10
HyperSAGE	N/A	N/A	N/A	N/A
HyperGCN	0.747 ± 0.122	0.775 ± 0.158	0.785 ± 0.164	0.767 ± 0.189
HGNN	0.866 ± 0.016	0.888 ± 0.136	0.892 ± 0.113	0.893 ± 0.153
HyperSAGNN	N/A	N/A	N/A	N/A
MDHNN	0.956 ± 0.006	0.983 ± 0.012	0.984 ± 0.018	0.993 ± 0.025
NHP	**0.982 ± 0.002**	**0.997 ± 0.006**	**0.999 ± 0.005**	**1.000 ± 0.000**
MLP	0.397 ± 0.019	0.260 ± 0.050	0.259 ± 0.060	0.280 ± 0.117
Structure	0.807 ± 0.017	0.815 ± 0.074	0.808 ± 0.076	0.820 ± 0.168

Bibliography

[1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(10):1533–1545, 2014.

[2] Sameer Agarwal, Kristin Branson, and Serge Belongie. Higher order learning with graphs. ICML '06, page 17–24, New York, NY, USA, 2006. Association for Computing Machinery.

[3] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[4] Devanshu Arya, Deepak K. Gupta, Stevan Rudinac, and Marcel Worring. Hypersage: Generalizing inductive representation learning on hypergraphs, 2020.

[5] Devanshu Arya and Marcel Worring. Exploiting relational information in social networks using geometric deep learning on hypergraphs. In *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*, ICMR '18, page 117–125, New York, NY, USA, 2018. Association for Computing Machinery.

[6] Song Bai, Feihu Zhang, and Philip H. S. Torr. Hypergraph convolution and hypergraph attention. *CoRR*, abs/1901.08150, 2019.

[7] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[8] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences*, 115(48):E11221–E11230, Nov 2018.

[9] Claude Berge. *Graphs and hypergraphs*, volume 6 of *North-Holland Mathematical Library*. Elsevier, 1976.

[10] Claude Berge. *Hypergraphs*, volume 45 of *North-Holland Mathematical Library*. Elsevier, 1984.

[11] Lidong Bing, Sneha Chaudhari, Richard Wang, and William Cohen. Improving distant supervision for information extraction using label propagation through lists. pages 524–529, 01 2015.

[12] A. Bordbar, Jonathan M. Monk, Zachary A. King, and B. Palsson. Constraint-based models predict metabolic and associated cellular functions. *Nature Reviews Genetics*, 15:107–120, 2014.

[13] Hugo Caselles-Dupré, Florian Lesaint, and Jimena Royo-Letelier. Word2vec applied to recommendation: Hyperparameters matter. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, page 352–356, New York, NY, USA, 2018. Association for Computing Machinery.

[14] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *CoRR*, abs/1912.09893, 2019.

[15] Ernesto Estrada and Juan A. Rodríguez-Velázquez. Subgraph centrality and clustering in complex hyper-networks. *Physica A: Statistical Mechanics and its Applications*, 364:581–594, May 2006.

[16] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, July 1983.

[17] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks, 2019.

[18] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[19] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2):177–201, 1993.

[20] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

[21] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[22] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005.

[23] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

[24] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[25] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[26] Jason Hartford, Devon R Graham, Kevin Leyton-Brown, and Siamak Ravanbakhsh. Deep models of interactions across sets, 2018.

[27] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: Data mining, inference, and prediction. *Math. Intell.*, 27:83–85, 11 2004.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[29] Jie Huang, C. Chen, F. Ye, J. Wu, Zibin Zheng, and Guohui Ling. Hyper2vec: Biased random walk for hyper-network embedding. In *DASFAA*, 2019.

[30] Kishan KC, Rui Li, Feng Cui, and Anne R. Haake. Predicting biomedical interactions with higher-order graph convolutional networks. *CoRR*, abs/2010.08516, 2020.

[31] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[32] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

[33] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

[34] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[35] Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and cellular networks. *PLoS computational biology*, 5:e1000385, 06 2009.

[36] V. Kumar, Madhukar S. Dasika, and C. Maranas. Optimization based automated curation of metabolic reconstructions. *BMC Bioinformatics*, 8:212 – 212, 2006.

[37] Dong Li, Zhiming Xu, Sheng Li, and Xin Sun. Link prediction in social networks based on hypergraph. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, page 41–42, New York, NY, USA, 2013. Association for Computing Machinery.

[38] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. 11 2015.

[39] Renjie Liao, Marc Brockschmidt, Daniel Tarlow, Alexander L. Gaunt, Raquel Urtasun, and Richard Zemel. Graph partition neural networks for semi-supervised classification, 2018.

[40] Zachary C. Lipton and Jacob Steinhardt. Troubling trends in machine learning scholarship: Some ml papers suffer from flaws that could mislead the public and stymie future research. *Queue*, 17(1):45–77, February 2019.

[41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[42] James Raymond Munkres. *Elements of algebraic topology*, volume 2. Addison-Wesley Menlo Park, 1984.

[43] Prasanna Patil, Govind Sharma, and M. Narasimha Murty. Negative sampling for hyperlink prediction in networks. In Hady W. Lauw, Raymond Chi-Wing Wong, Alexandros Ntoulas, Ee-Peng Lim, See-Kiong Ng, and Sinno Jialin Pan, editors, *Advances in Knowledge Discovery and Data Mining*, pages 607–619, Cham, 2020. Springer International Publishing.

[44] Josh Payne. Deep hyperedges: a framework for transductive and inductive learning on hypergraphs, 2019.

[45] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[46] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 701–710, New York, NY, USA, 2014. Association for Computing Machinery.

[47] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2110–2119, New York, NY, USA, 2018. Association for Computing Machinery.

[48] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Deep learning with sets and point clouds, 2017.

[49] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[50] A. Sharma, Shafiq R. Joty, H. Kharkwal, and J. Srivastava. Hyperedge2vec: Distributed representations for hyperedges. 2018.

[51] Ankit Sharma, Jaideep Srivastava, and Abhishek Chandra. Predicting multi-actor collaborations using hypergraphs, 2014.

[52] Govind Sharma, Prasanna Patil, and M. Murty. C3mm: Clique-closure based hyperlink prediction. pages 3336–3342, 07 2020.

[53] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018.

[54] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[55] Xiaolong Wang, Yufei Ye, and Abhinav Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs, 2018.

[56] Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence*, pages 1–1, 2021.

[57] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. HyperGCN: A new method of training graph convolutional networks on hypergraphs, 2019.

[58] Naganand Yadati, Vikram Nitin, Madhav Nimishakavi, Prateek Yadav, Anand Louis, and Partha Talukdar. *NHP: Neural Hypergraph Link Prediction*, page 1705–1714. Association for Computing Machinery, New York, NY, USA, 2020.

[59] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *CoRR*, abs/1603.08861, 2016.

[60] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 974–983, New York, NY, USA, 2018. Association for Computing Machinery.

[61] Se-eun Yoon, Hyungseok Song, Kijung Shin, and Yung Yi. How much and when do we need higher-order information in hypergraphs? a case study on hyperedge prediction. *Proceedings of The Web Conference 2020*, Apr 2020.

[62] Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep sets. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3394–3404, Red Hook, NY, USA, 2017. Curran Associates Inc.

[63] Muhan Zhang, Zhicheng Cui, Shali Jiang, and Yixin Chen. Beyond link prediction: Predicting hyperlinks in adjacency space, 2018.

[64] Muhan Zhang, Zhicheng Cui, Tolutola Oyetunde, Yinjie Tang, and Yixin Chen. Recovering metabolic networks using a novel hyperlink prediction method, 2016.

[65] Ruochi Zhang, Yuesong Zou, and Jian Ma. Hyper-SAGNN: a self-attention based graph neural network for hypergraphs, 2019.

[66] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey, 2020.

[67] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, page 1601–1608, Cambridge, MA, USA, 2006. MIT Press.

[68] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[69] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2847–2856, New York, NY, USA, 2018. Association for Computing Machinery.