AdaGrid: Adaptive Grid Search for Link Prediction Training Objective

Anonymous Author(s)
Affiliation
Address
email

Abstract

One of the most important factors which contribute to the success of a machine learning model is a good training objective. Training objective crucially influences a model's performance and generalization capabilities. The automated process of designing a good training objective involves optimizing a machine learning process, therefore can be viewed as a meta-learning problem. In this paper, we specifically focus on graph neural network training objectives for link prediction, which has not been explored in the existing literature. Here, the training objective includes, among others, training mode, negative sampling strategy, and various hyperparameters, such as edge message ratio. Commonly, these hyperparameters are fine-tuned by complete grid search, which is very time-consuming and model-dependent. To mitigate these limitations, we propose Adaptive Grid Search (AdaGrid), which dynamically adjusts the edge message ratio during training. It is model agnostic and highly scalable with a fully customizable computational budget. Through extensive experiments, we show that AdaGrid can boost the performance of the models up to 1.9%, while can be nine times more efficient than a complete search. Overall, AdaGrid represents an effective automated algorithm for designing machine learning training objectives.

1 Introduction

Link prediction is one of the most important tasks on graph-structured data. For a given pair of entities, the goal of link prediction is predicting whether they are going to interact. Applications of link prediction are found in various fields, such as social networks, recommender systems, and biology. There have been many strategies to cope with the link prediction task [19], where the state-of-the-art approaches use Graph Neural Networks (GNNs) [12, 10, 36]. GNNs capture dependencies via message passing between the nodes of graphs and are suitable for various graph-related tasks. To train GNNs, an appropriate training objective has to be selected, which includes choice of the objective function, evaluation metric, and training strategy.

The automated process of designing a good training objective involves optimizing a machine learning process, therefore can be viewed as a meta-learning problem. In this paper, we specifically focus on graph neural network training objectives for link prediction, which has not been explored in the existing literature. Here, link prediction training objectives encompass training mode, negative sampling strategy, and hyperparameters such as edge message ratio; they crucially influence the performance of a model and its generalization capabilities on the link prediction task, so they have to be chosen carefully. Notably, setting an appropriate edge message ratio is notoriously hard since there are many relevant factors that determine the optimal edge message ratio: data split ratio, model configuration, the average clustering coefficient of the network, and the average shortest path length of the network.

Submitted to the 5th Workshop on Meta-Learning at NeurIPS 2021, Sydney, Australia. Do not distribute.

The standard approach to fine-tune link prediction objectives is a complete search over some hyperparameter space [37, 29]. Complete search trains more models for full epochs, each with a different edge message ratio from a set of predefined values, and selects the one with the maximal final validation AUC. Even though it exhaustively searches hyperparameter space, which is very time-consuming, it still obtains non-optimal performance. To make matters worse, even when the configuration of the model slightly changes, a complete search has to be repeated.

In this paper, we propose Adaptive Grid Search (AdaGrid) which adjusts edge message ratio during training to each specific model and dataset, therefore alleviates the above-mentioned limitations. AdaGrid is model agnostic and highly scalable with a fully customizable computational budget. Through extensive experiments, we show that AdaGrid can boost the performance of the models up to 1.9%, while can be nine times more efficient than a complete search. We also propose a new negative sampling strategy, which is based on the community structure of the network. Unlike the standard uniform negative sampling, it samples more difficult negative instances and prevents abuse of community structure of networks.

Our key contributions include: **(1)** We define GNN training modes for link prediction. **(2)** We explore the GNN training objective for link prediction. **(3)** We propose Adaptive Grid Search (AdaGrid), which resolves limitations of complete search. **(4)** We introduce community ratio-based negative sampling, which samples more difficult negative instances and prevents the abuse of the community structure of networks.

2 Related Work

Central to our research are graph neural network models, to which we tailor training objectives for the link prediction tasks. Thus, we provide a brief overview of general graph neural networks and their application to link prediction tasks.

General graph neural networks In recent years, numerous graph neural network (GNN) models have been proposed. Some of them are inspired by convolutional neural networks [3, 6, 7, 10, 12, 25, 15, 21, 28], recurrent neural networks [16, 31], recursive neural networks [2, 8], and loopy belief propagation [5]. The majority of these approaches can be united under the Graph Network framework proposed by Battaglia et al. [1]. In the Graph Network framework, a GNN is considered as a message-passing algorithm, where representations of nodes, edges, and the global attribute are iteratively updated from neighboring nodes, edges, and the global attribute using a differentiable, order-invariant aggregation function. A recent extensive overview of recent GNN models, general design pipeline for constructing such models and their applications is provided by Zhou et al. [36].

Link prediction with graph neural networks GNNs have been applied to a wide variety of tasks, including, but not limited to, node classification [10, 12], link prediction [22], graph classification [5, 7, 35], and chemoinformatics [20, 18, 9, 11]. In the context of link prediction, a few approaches have been introduced recently [14]. Schlichtkrul et al. [22] proposed a relational graph convolutional neural network model (R-GCN) which incorporates GCN [12] as the building block to model relational data, knowledge graph in particular. It encodes nodes such that a node's latent representation depends on all neighboring nodes as well as the node itself. Kipf et al. [13] introduced a variational graph autoencoder (VGAE) framework that learns latent representation on graph-structured data. The encoder part, based on GCNs, embeds nodes in latent space, then, the decoder part tries to reconstruct the adjacency matrix of the graph by the inner product of latent representations. Zhang et al. [32] proposed a novel framework called Weisfeiler-Lehman Neural Machine (WLNM), which is based on the Weisfeiler-Lehman algorithm that labels nodes of the graph according to the topology of the underlying graph. For each prospective node pair, WNLM extracts a subgraph in their neighborhood and encodes it as an adjacency matrix. Based on these adjacency matrices, WLNM then trains a classifier. Furthermore, Zhang et al. [33] introduced SEAL, a new heuristic learning paradigm, which captures first, second, and higher-order structural information in the form of local subgraphs. It unifies local subgraph, embedding, and attribute information using the graph convolutional network.

Although numerous approaches for link prediction with GNN models have been proposed, their training objective is usually vaguely specified. This inspired us to define graph neural network training objectives for link prediction and research their properties.

3 Preliminaries

We represent graph as $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ is the node set and $E \subseteq V \times V$ is the edge set. Nodes can be paired with features $X = \{x_v : v \in V\}$. Given a set of of labeled node pairs $D = \{(u_1, v_1, y_1), (u_2, v_2, y_2), \ldots\}$, where y_i denotes presence/absence of the edge between the node pair $(u_i, v_i) \in V \times V$, the goal of link prediction is to learn a mapping $f : V \times V \to \{0, 1\}$ that for a given pair of nodes predicts whether they are likely to be connected by an edge.

Graph neural networks In this paper we approach the link prediction problem by message passing GNNs [27]. The goal of GNNs is to learn meaningful node embeddings h_v, which are based on iterative aggregation of local neighborhoods and should be informative for the task in question. The k-th iteration/layer of message passing can be written as [30]:

$$m_u^{(k)} = \text{MSG}^{(k)}(h_u^{(k-1)}), \tag{1}$$

$$h_v^{(k)} = \text{AGG}^{(k)}(\{m_u^{(k)}, u \in \mathcal{N}(v)\}, h_v^{(k-1)}), \tag{2}$$

where $h_v^{(k)}$ is the node embedding after k iterations, $h_v^0 = x_v$, $m_v^{(k)}$ is message embedding, and $\mathcal{N}(v)$ is the local neighborhood of node v. Definitions of $\text{MSG}^{(k)}(\cdot)$ and $\text{AGG}^{(k)}(\cdot)$ depend on version of the GNN. The final node embeddings $h_v = h_v^{(K)}$ can be then used for various node, edge, and graph level tasks. In the case of link prediction, the embeddings can be optimized so that the following equation resembles the probability of an edge between nodes u and v:

$$P((u, v) \in E) = \sigma\left(h_u \cdot h_v\right), \tag{3}$$

where σ represents sigmoid function and $\cdot$ denotes dot product.

Standard learning-based link prediction experimental setting Learning-based link prediction task is frequently formulated as binary classification, where potential edges are classified as true or false. The standard learning-based link prediction setting [33] first splits the graph's edge set E into a training set, validation set, and test set. The ratio between the cardinalities of these sets is called split ratio (i.e. training/validation/test split ratio or training/validation split ratio). For link prediction using GNN models, each set of edges have to be further divided into message-passing edges and objective edges. Message-passing edges are used for the propagation of information between nodes, while loss is calculated based on objective edges. During the evaluation of the model's performance, all validation and test edges are used only for the estimation of loss. For propagation of information, validation utilizes training message-passing edges, while testing exploits all training and validation edges. So far, objective edges comprise only of edges, which are present in the graph – they are positive instances or positive edges for the link prediction task. Therefore, the node pairs that are not connected by an edge are required for a fair binary classification task. These so-called negative instances or negative edges are usually obtained by uniformly randomly sampling nonexistent edges (unconnected node pairs). The standard learning-based link prediction setting samples negative instances until each set of objective edges is balanced – there is the same number of positive and negative instances.

Training objective for link prediction When training a machine learning model, one has to specify model evaluation function, objective function for optimization, as well as strategy how to use the available data. All of these components are captured under the term training objective. Models are usually evaluated by different metrics, while optimization is performed based on loss functions. On the other hand, strategies define how data is split, which instances are presented to the model in which epoch, how to sample negative instances, and so on. In the case of link prediction using GNNs, the latter includes also the choice of appropriate training mode, edge message ratio, and negative sampling. While evaluation and objective functions are well studied, not much attention has been given to training objective strategies. Therefore, we provide the *first* categorization of different training modes, which are also supported by DeepSNAP [24] – a popular network library for GNNs.

Training modes For link prediction using GNN models, edges have to be split into message-passing edges and objective edges. This split can be performed in different ways, each corresponding to a different training mode. We recognize two main training modes for link prediction: `all mode` and `disjoint mode`. The `all mode` exploits the whole edge set for message passing and calculation

of loss, so each edge is used for propagation of information as well as for updating weights of the model. Such training mode is employed by graph autoencoders [13]. However, prior researchers have identified shortcomings of the autoencoder approaches [34]: The setup potentially leads to overfitting of the GNN model, since the model is never asked to predict edges that are not part of input, and thus tend to only memorize the message-passing edges. The `disjoint mode` resolves this by dividing edges in a set for message passing, which is disjoint from the set of objective edges. In this case, each edge belongs to only one of the roles. Consequentially, `disjoint mode` introduces an additional hyperparameter – *edge message ratio*, which control the proportion of message-passing edges. For instance, if there are 70 message-passing edges and 30 objective edges, the edge message ratio equals to 0.7. Normally, the split into objective and message-passing edges is performed prior to the training. Nevertheless, there exists an extension to the `disjoint mode` – the `resample disjoint mode`, which resamples both sets every fixed number of epochs to allow training to take full advantage of all edges for both purposes.

4 Proposed Method: Adaptive Grid Search (AdaGrid)

The key idea behind our approach is that we dynamically adapt training objectives during training to each configuration of the model and each dataset. We first compare training modes, highlight limitations of training objective, and expose disadvantages of complete search, which is the standard approach for selection of GNN training objective for link prediction (Section 4.1). We then propose Adaptive Grid Search (AdaGrid), which resolves drawbacks of complete search (Section 4.2). At the end, we introduce a novel negative sampling method, which takes into consideration the community structure of the network and creates more challenging negative instances (Section 4.3).

4.1 Limitations of Training Objective

We first compare training modes to establish which one yields the best results. As is shown in Appendix C.1, the `resample disjoint mode` consistently outperforms the `all mode` and the vanilla `disjoint mode` according to final validation AUC. However, additional experiments disclose that the `resample disjoint mode` is superior only when combined with an appropriate edge message ratio. Selection of bad edge message ratio results in performance considerably worse than the `all mode`. So the `resample disjoint mode` improves the model but in exchange for some extra work with an additional hyperparameter tuning. This suggests that the edge message ratio has to be fine-tuned carefully. The most straightforward and standard approach is to do a complete search on some predefined values, but this is very time-consuming. Based on the above observations, it is beneficial to estimate a good edge message ratio quickly. However, it turns out that the edge message ratio is notoriously hard to set. Appendix C.2 illustrates that there are no patterns that depend on: data split ratio, model configuration, the average clustering coefficient of the network, and the average shortest path length of the network.

Complete search Since a good training objective is specific to each configuration of the model and dataset, the standard approach to finding a good edge message ratio is a complete search over some predefined set of values Q with cardinality $L = |Q|$ [29]. Complete search is very time-consuming and computationally demanding because the model has to be trained multiple times with different edge message ratios. A possible speedup is to train each version of the model only for part of all epochs and then take the one which performs best at that time. This relaxation assumes that the best model is already evident before all models are trained for full epochs. While this seems as an adequate solution, there are still certain limitations. Even by slightly changing the configuration of the model, the optimal edge message ratio changes and complete search has to be repeated, which is extremely inconvenient. Another, possible false, assumption of this approach is that the optimal edge message ratio does not change during training. To resolve these drawbacks of the standard approach, we propose Adaptive Grid Search (AdaGrid).

4.2 Adaptive Grid Search (AdaGrid)

Description The key feature of AdaGrid is its ability to adapt edge message ratio during training to each configuration of the model and each dataset. AdaGrid changes the edge message ratio every $adapt\_epochs$. Then, it trains L copies of the model with different edge message ratios in parallel, where each copy is trained for try_epochs. A set of predefined edge message ratios Q which are taken into consideration can be chosen according to preference. After try_epochs of training, the

edge message ratio is selected based on one of two possible criteria: *validation criterion* and *gap criterion*. The validation criterion selects the edge message ratio which corresponds to the model with the highest final validation AUC. Since sometimes validation AUC is a bit unstable during training also smoothing can be performed – instead of the final validation AUC, rather average of a few last validation AUCs is taken. On the other hand, the gap criterion chooses the edge message ratio of the model with minimal absolute difference between the final training and validation AUC. Analogously, also gap criterion can utilize smoothing. One of the main upsides of AdaGrid is its flexibility since it can be adjusted to each application separately. The $adapt\_epochs$ and the try_epochs regulate training time, while selection criterion and set of considered edge message ratios Q can be tailored to each specific task.

Computational budget of AdaGrid By setting the $adapt\_epochs$ and the try_epochs appropriately, AdaGrid's computational budget can be fully customizable. If model is trained for $full\_epochs$ and L edge message ratios are considered, overall number of training epochs of AdaGrid is:

$$\text{number of epochs} = full\_epochs \cdot \left(1 + \frac{(L-1) \cdot try\_epochs}{adapt\_epochs}\right), \tag{4}$$

while complete search requires:

$$\text{number of epochs} = full\_epochs \cdot L. \tag{5}$$

For instance, if $try\_epochs = adapt\_epochs$ both required the same number of epochs, while if $try\_epochs = 5$, $adapt\_epochs = 50$, and $L = 9$, AdaGrid needs five times fewer training epochs.

4.3 Community Ratio-based Negative Sampling

Uniform negative sampling seems to be an appropriate approach for obtaining negative instances, however, after careful analysis, it turns out that these instances are rather simple negative examples for link prediction. A lot of networks inherently display some kind of community structure – the network can be partitioned into disjoint communities so that connections within communities are denser than the connections with the rest of the network. Let us define edges that have both endpoints in the same community as the *within community edges*, and edges that have endpoints in the different communities as the *between communities edges*.

Limitations of uniform negative sampling In Appendix B.1, we have theoretically shown that, under certain assumptions, uniform negative sampling yields mostly between community edges. The same findings are confirmed empirically in Appendix B.2. This can be exploited by a naive model which does not even consider nodes' features to perform unfairly well. By always predicting there is an edge between u and v for node pairs from the same community, and always predicting there is not one for node pairs from different communities, classification accuracy far beyond baseline 0.5 can be obtained.

Community ratio-based negative sampling To improve uniform negative sampling, we propose a negative sampling which also considers the community structure of the network. First, let us define *community ratio* as the proportion of node pairs with nodes from the same community. Our community ratio-based negative sampling obtains negative edges in such a way that sets of negative and positive instances can not be distinguished based on community ratio. It first performs community detection on the graph with all training edges, then it measures the community ratio on validation edges. Afterward, negative instances for all three sets are sampled in compliance with the gauged community ratio. Our approach is beneficial because community structure can not be abused anymore. At the same time, it generates more challenging negative instances, which can better differentiate the performance of models.

5 Experiments

We test the performance of AdaGrid on various model configurations, datasets, data split ratios, negative sampling strategies, and hyperparameter settings of AdaGrid. Firstly, we describe datasets, the model configuration used in experiments, baselines against which AdaGrid is compared, and experimental set-up. In Section 5.1, we interpret the results of AdaGrid on the standard uniform negative sampling setting, while Section 5.2 explains the benefits of both, community ratio-based negative sampling and AdaGrid. In the end, in Section 5.3 we unfold why AdaGrid performs better than the baselines.

Table 1: AUC in percent for AdaGrid, complete search, and random search with uniform negative sampling evaluation. Model has $K = 2$, $o = 64$, and is trained in the `resample disjoint mode`.

		Datasets					
		Cora			CiteSeer		
Methods		$20/40/40$	$50/25/25$	$80/10/10$	$20/40/40$	$50/25/25$	$80/10/10$
Complete search		94.85	96.65	97.17	95.84	97.78	98.46
Random search		94.71	96.40	97.02	95.88	97.67	98.45
AdaGrid							
$adapt\_epochs$	try_epochs						
100	1	95.03	96.89	97.54	95.96	97.82	98.69
100	5	95.03	96.87	97.63	95.91	97.80	98.64
100	100	94.93	96.98	97.59	**95.97**	97.79	98.67
50	1	95.01	96.98	97.68	95.94	97.81	**98.72**
50	5	95.05	97.04	97.75	95.87	97.81	98.68
50	50	95.03	97.06	97.76	**95.97**	97.83	98.71
10	1	95.01	97.07	**97.90**	95.79	**97.89**	98.70
10	5	**95.14**	**97.10**	97.83	95.81	97.83	98.71
10	10	95.10	97.09	97.86	95.94	97.88	98.71
Gain		0.29	0.45	0.73	0.09	0.11	0.26

Datasets All experiments in the paper are conducted on Cora and CiteSeer datasets [23]. Additionally, limitations of training objective experiments use also the PubMed dataset [23]. All three datasets are well-known citation networks (additional details are available in the Appendix A).

Model configuration Experiments were conducted using the following model configuration. Our model consists of K GCN layers applied sequentially, where before each GCN layer there is a dropout of 0.2, and after each layer but the last there is a ReLU activation. As input it accepts nodes' features $h_v^{(0)} = x_v \in \mathbb{R}^d$ and it outputs final hidden representations $h_v = h_v^{(K)} \in \mathbb{R}^o$. All intermediate hidden representations have the same dimensionality as the final representation: $h_v^{(i)} \in \mathbb{R}^o$ for $i = 1, 2, \ldots, K-1$. The model predicts the probability of an edge between nodes u and v according to Equation (3). In experiments, d depends solely on the dimensionality of the graph's features, while o is a hyperparameter. The model is always trained for 500 epochs using binary cross-entropy as loss function, however, the quality of the model is rather measured by AUC metric. Its parameters are optimized by stochastic gradient descent (SGD) with the learning rate of 0.1, the momentum of 0.9, and the weight decay of $5 \cdot 10^{-4}$. During training, a cosine annealing [17] schedule is used for alternation of the learning rate.

Baselines To contextualize the empirical results of our approach, we compare AdaGrid against two baselines: complete search and random search. Complete search trains more models for full epochs, each with a different edge message ratio from a set of predefined values, and selects the one with the maximal final validation AUC. It exhaustively searches hyperparameter space, which makes it very time-consuming. It also does not change the edge message ratio during training. On the other hand, random search modifies edge message ratio to a random value from $[0.1, 0.9]$ interval after every training epoch. This makes it very fast, however, different edge message ratios are not inspected. We show that both baselines perform inferior to AdaGrid: complete search has static edge message ratio and is slow, while random search does not explore edge message ratio space.

Experimental set-up When comparing AdaGrid with the standard complete search and random search, we are interested principally in absolute performance and the trade-off between training time and performance. Experiments are systematically conducted over various settings, which include different model configurations, datasets, data split ratios, and negative samplings. We use Cora and CiteSeer datasets. To get more representative results we test the model with two configurations: $K = 2$, $o = 64$ and $K = 3$, $o = 128$. The model is always trained in the `resample disjoint mode`, however, data is every time divided according to other split ratios: $20/40/40$, $50/25/25$, and $80/10/10$. Models are trained and evaluated using the standard uniform sampling as well as community ratio-based negative sampling, proposed in Section 4.3. In a community ratio-based negative sampling setting, community detection is performed using the Clauset-Newman-Moore greedy modu-

Table 2: AUC in percent for AdaGrid, complete search, and random search with community ratio-based negative sampling evaluation. Model has $K = 2$, $o = 64$, and is trained in the `resample disjoint mode`.

		Datasets					
		Cora			CiteSeer		
Methods		$20/40/40$	$50/25/25$	$80/10/10$	$20/40/40$	$50/25/25$	$80/10/10$
Complete search		84.01	82.98	84.62	83.87	82.22	83.96
Random search		84.00	82.05	83.82	83.87	82.61	83.65
AdaGrid							
$adapt\_epochs$	try_epochs						
100	1	84.35	83.05	85.27	**84.01**	83.20	84.58
100	5	84.52	82.88	85.43	83.92	82.86	84.34
100	100	84.49	83.61	85.57	84.00	83.12	84.75
50	1	84.51	83.48	86.02	83.88	83.11	84.79
50	5	84.59	83.56	85.98	83.90	82.94	84.84
50	50	84.59	83.74	85.74	84.00	82.99	84.81
10	1	84.49	83.69	86.27	83.65	83.11	**84.90**
10	5	**84.65**	83.82	86.54	83.75	**83.37**	84.89
10	10	84.59	**84.07**	**86.56**	83.94	83.04	84.83
Gain		0.64	1.09	1.94	0.14	0.76	0.94

larity maximization algorithm [4]. AdaGrid and complete search both consider only the following set of edge message ratios: $Q = \{0.1, 0.2, \ldots, 0.9\}$. To examine power of AdaGrid, it is assessed with various configurations of $(adapt\_epochs, try\_epochs) \in \{10, 50, 100\} \times \{1, 5, adapt\_epochs\}$ and both criteria from Section 4.2. Both criteria utilize smoothing. The selection criterion is considered a hyperparameter of AdaGrid, so we present results for the better of the two. Each experiment is repeated three times to mitigate the effect of randomness and the average performance is reported.

5.1 AdaGrid and Uniform Negative Sampling

We first evaluate AdaGrid on a uniform negative sampling setting, because this is the standard approach for link prediction. Table 1 contains results of AdaGrid and baselines on $K = 2$ and $o = 64$ model configuration, however, $K = 3$ and $o = 128$ displays similar performance. The table shows that AdaGrid consistently performs better than the best baseline approach, no matter its configuration. Improvement of more than 0.7% is especially evident for the $80/10/10$ split ratio, which is the most similar to the usual experimental data splits. Another crucial aspect of AdaGrid is its adjustability in terms of computational budget. Even when AdaGrid is trained for considerably fewer training epochs than complete search, it constantly performs better than the best baseline approach. If AdaGrid is trained with $adapt\_epochs = 10$ and $try\_epochs = 1$, it requires almost five times fewer epochs than a complete search and twice as many as a random search, while it can surpass both for more than 0.7%. AdaGrid even outperforms both of them, when $adapt\_epochs = 100$ and $try\_epochs = 1$. In this case, AdaGrid is about nine times faster than a complete search and is computationally almost equivalent to the training of a single model. When decreasing the number of try_epochs, the performance also slowly decreases, however, a similar performance can be obtained in only a fraction of computational time. This can be particularly useful for huge models.

5.2 AdaGrid and Community Ratio-based Negative Sampling

We also evaluate AdaGrid on the proposed community ratio-based negative sampling setting to display its advantages, as well as advantages of AdaGrid. Table 2 shows that AdaGrid nearly always performs better than the best baseline approach, regardless of its configuration. By comparing results of negative sampling strategies, it is evident that community ratio-based negative sampling creates a more challenging evaluation setting since AUC scores are considerably lower. It also better differentiates the performance of models, because gains of AdaGrid are always bigger than the ones of uniform negative sampling. Especially outstanding is the gain of more than 1.9%, which again confirms the benefits of AdaGrid.

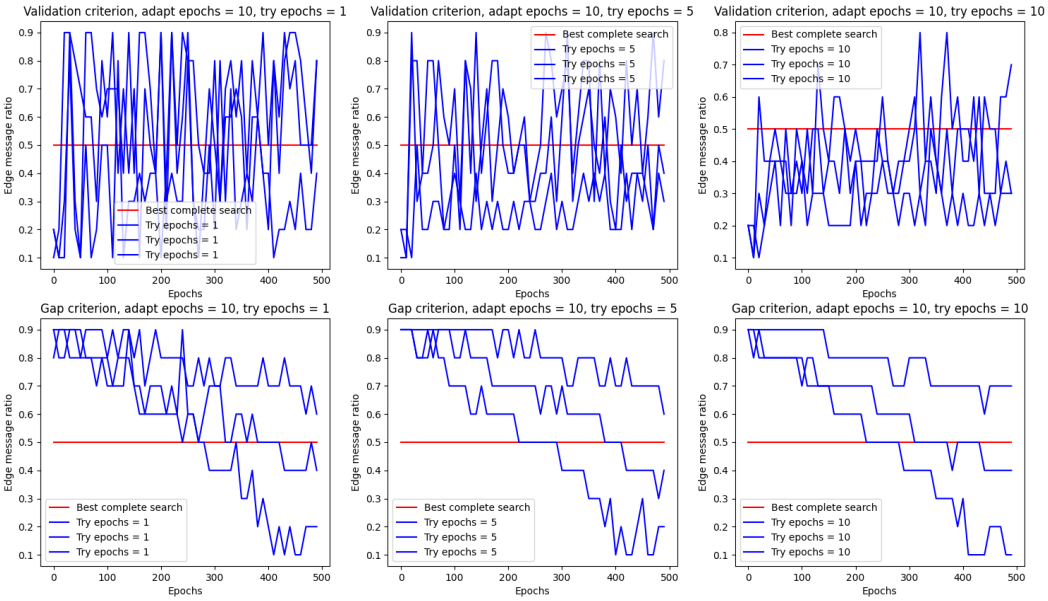

Figure 1: Edge message ratio during training, regulated by validation criterion and gap criterion. Model has $K = 2$ and $o = 64$. It is trained on CiteSeer with $80/10/10$ split ratio in the `resample disjoint mode`.

5.3 AdaGrid and Edge Message Ratio

We have investigated the properties of AdaGrid, which could explain its improvement of performance. The success of AdaGrid probably stems from its ability to change edge message ratio during training in an informed way. According to Figure 1 AdaGrid modifies edge message ratio almost every $adapt\_epochs$ in conjunction with both criteria. Additionally, Tables 1 and 2 show that AdaGrid performs better with a lower number of $adapt\_epochs$ – it is beneficial to be capable of changing edge message ratio more frequently. Therefore, complete search results in non-optimal performance, because it assumes the edge message ratio is static. On the other hand, random search does not explore edge message ratio space, so it incorrectly alters it.

6 Conclusion

The aim of the paper is to explore graph neural network training objectives for link prediction. We first propose community-based negative sampling, which is fairer and harder than the standard uniform negative sampling. It makes distributions of negative and positive instances indistinguishable based only on community ratio, as well as disables some naive models which rely solely on community detection algorithms. We also show that it is very hard to find a suitable training objective, since it is specific to each dataset and model configuration. To diminish the inconvenience of edge message ratio fine-tuning, we propose AdaGrid, which adapts edge message ratio "on-the-fly" during training and overcomes limitations of complete search. It is model agnostic and has a fully customizable computational budget. More importantly, AdaGrid can also reduce training time and boost performance at once. It can improve the performance of the models up to 1.9%, while can be nine times more efficient than a complete search.

Future work Since AdaGrid performs well on link prediction, it would be interesting to apply AdaGrid to other graph learning tasks. Due to its generality, it would be suitable even for other deep learning tasks. Instead of edge message ratio, AdaGrid can optimize any parameter which can be dynamically changed during training. To even further speed up AdaGrid, instead of considering all edge message ratios only those which are adjacent to the current one can be considered during adaptation phases. Another intriguing idea is to eliminate a predefined set of values from which edge message ratios are selected. By locally interpolating validation AUCs near the current edge message ratio, a new edge message ratio can be chosen so that it maximizes validation AUC.

References

[1] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

[2] M. Bianchini, M. Gori, and F. Scarselli. Processing directed acyclic graphs with recursive neural networks. *IEEE Transactions on Neural Networks*, 12(6):1464–1470, 2001. doi: 10.1109/72.963781.

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[4] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70:066111, Dec 2004. doi: 10.1103/PhysRevE.70.066111. URL `https://link.aps.org/doi/10.1103/PhysRevE.70.066111`.

[5] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2702–2711, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `https://proceedings.mlr.press/v48/daib16.html`.

[6] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

[7] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL `https://proceedings.neurips.cc/paper/2015/file/f9be311e65d81a9ad8150a60844bb94c-Paper.pdf`.

[8] Matthias Fey, J. E. Lenssen, F. Weichert, and H. Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[9] A. Fout, Jonathon Byrd, B. Shariat, and A. Ben-Hur. Protein interface prediction using graph convolutional networks. In *NIPS*, 2017.

[10] 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*, pages 1025–1035, 2017.

[11] Wengong Jin, Connor W. Coley, R. Barzilay, and T. Jaakkola. Predicting organic reaction outcomes with weisfeiler-lehman network. In *NIPS*, 2017.

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

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

[14] Ajay Kumar, Shashank Sheshar Singh, Kuldeep Singh, and Bhaskar Biswas. Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications*, 553:124289, 2020.

[15] Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2024–2033. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/lei17a.html`.

[16] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[17] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

[18] A. Lusci, G. Pollastri, and P. Baldi. Deep architectures and deep learning in chemoinformatics: The prediction of aqueous solubility for drug-like molecules. *Journal of chemical information and modeling*, 53 7:1563–75, 2013.

[19] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2010.11.027. URL https://www.sciencedirect.com/science/article/pii/S037843711000991X.

[20] C. Merkwirth and Thomas Lengauer. Automatic generation of complementary descriptors with molecular graph networks. *Journal of chemical information and modeling*, 45 5:1159–68, 2005.

[21] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2014–2023, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/niepert16.html.

[22] M. Schlichtkrull, Thomas Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and M. Welling. Modeling relational data with graph convolutional networks. *ArXiv*, abs/1703.06103, 2018.

[23] P. Sen, Galileo Namata, M. Bilgic, L. Getoor, B. Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29:93–106, 2008.

[24] Stanford. DeepSNAP, 2021. URL https://github.com/snap-stanford/deepsnap.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[26] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

[27] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[28] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *NeurIPS*, 2018.

[29] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[30] Jiaxuan You, Jonathan Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. *AAAI Conference on Artificial Intelligence*, 2021.

[31] Victoria Zayats and Mari Ostendorf. Conversation Modeling on Reddit Using a Graph-Structured LSTM. *Transactions of the Association for Computational Linguistics*, 6:121–132, 02 2018. ISSN 2307-387X. doi: 10.1162/tacl_a_00009. URL https://doi.org/10.1162/tacl_a_00009.

[32] Muhan Zhang and Yixin Chen. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 575–583, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3097996. URL https://doi.org/10.1145/3097983.3097996.

[33] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, 2018.

[34] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.

[35] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

[36] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. ISSN 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2021.01.001. URL https://www.sciencedirect.com/science/article/pii/S2666651021000012.

[37] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. Auto-gnn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184*, 2019.

A Dataset Details

Cora Cora dataset [23] consists of 2708 scientific publications from Cora and 5429 citation links. Each scientific publication belongs to one of the seven classes and is described by a binary-valued word vector, which indicates the absence/presence of the corresponding words from the dictionary. The dictionary contains 1433 unique words.

CiteSeer CiteSeer dataset [23] consists of 3312 scientific publications from CiteSeer and 4732 citation links. Each scientific publication belongs to one of the six classes and is described by a binary-valued word vector, which indicates the absence/presence of the corresponding words from the dictionary. The dictionary contains 3703 unique words.

PubMed PubMed Diabetes dataset [23] consists of 19717 scientific publications from PubMed related to diabetes and 44338 citation links. Each scientific publication belongs to one of the three classes and is described by a TF-IDF weighted word vector from a dictionary with a size of 500.

B Limitations of Uniform Negative Sampling

Although uniform negative sampling is the standard approach for the evaluation of link prediction tasks, it has a serious drawback. We first theoretically show that under certain assumptions uniform negative sampling creates simple link prediction evaluation (Appendix B.1). Then we confirm theoretical results also empirically on a real network (Appendix B.2).

B.1 Theoretical Analysis of Uniform Negative Sampling

Imagine that a network with N nodes is partitioned into $R \ll N$ communities of equal size, where N is a multiple of R. Then for a random node pair (u, v), it is much more likely that nodes are from different communities than conversely. Let A denote the event that u and v are from the same community, and B that they are from different communities:

$$P(B) = 1 - P(A) = 1 - \frac{R \cdot (\frac{N}{R} \cdot (\frac{N}{R} - 1)/2)}{N \cdot (N-1)/2} \tag{6}$$

$$= 1 - \frac{(\frac{N}{R} - 1)}{(N-1)} \tag{7}$$

$$\approx 1 - \frac{1}{R}, \tag{8}$$

if N and $\frac{N}{R}$ are large enough. Since it makes sense to have multiple communities, it is safe to assume $P(B) \geq \frac{1}{2}$. The latter estimate is not tight for real networks, because they usually have R at least a bit larger than 2. For example, Equation (8) yields probability $P(B) = 0.9$ for $R = 10$. Therefore, if node pairs are sampled uniformly, the majority of negative instance node pairs correspond to nodes from different communities.

The above observation can be exploited so that a baseline model based only on community structure can already get high classification accuracy. For the following analysis let's assume that positive edges are equally likely to be within community and between communities edges. This assumption provides a lower bound on classification accuracy because positive edges should contain way more within community edges. So by always predicting there is an edge between u and v for node pairs from the same community, and always predicting there is not one for node pairs from different communities, the following accuracy score (ACC) is obtained:

$$ACC = P(\text{correct} \,|NEG) \cdot P(NEG) + P(\text{correct} \,|POS) \cdot P(POS) \tag{9}$$

$$\approx (1 - \frac{1}{R}) \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2}, \tag{10}$$

where POS and NEG denote events that an edge belongs to positive instances or negative instances. Accuracy above is considerably larger than expected 0.5 (e.g. ≈ 0.7 for $R = 10$), which is not desired.

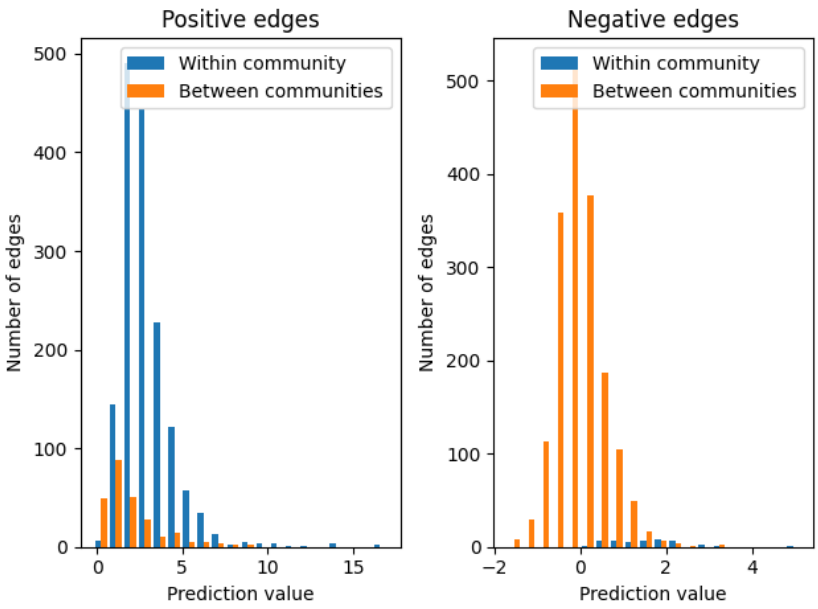

Figure 2: Prediction values (dot product of nodes' hidden representations) of negative and positive edges before sigmoid activation on CiteSeer dataset. To find edges within and between communities, community detection is performed using the Clauset-Newman-Moore greedy modularity maximization algorithm [4]. Model has $K = 3$, $o = 128$, and $edge\_message\_ratio = 0.6$. It is trained with $80/20$ split ratio in the `resample disjoint mode`.

B.2 Empirical Analysis of Uniform Negative Sampling

The assumption that all communities are of equal size is usually not true for real networks, however, on real networks similar findings are discovered empirically. From Figure 2, it is evident that on the CiteSeer dataset the vast majority of negative instances corresponds to nodes from different communities and that positive instances are mostly within community edges. It can also be observed that the model has difficulties with the prediction of negative edges within communities because it assigns them considerably above-average prediction values. These edges are probably also misclassified by the model.

C Limitations of Training Objective

In this section, we explore the GNN training objective for link prediction. We compare different training modes (Appendix C.1) and highlight limitations of the `resample disjoint mode`, which is the best-performing training mode (Appendix C.2).

C.1 Comparison of Training Modes

In this section we compare the following training modes: `all mode`, `disjoint mode`, and `resample disjoint mode`. These training modes are presented in detail in Section 3. Figure 3 shows that `resample disjoint mode` is superior according to final validation AUC, however, additional experiments reveal that only if combined with an appropriate edge message ratio. The `resample disjoint mode` has also the smallest difference between training and validation AUC, since `resample disjoint mode` significantly reduces overfitting. These findings are consistent overall datasets from Section 5, so we recommend using the `resample disjoint mode`.

C.2 Limitations of the `Resample Disjoint Mode`

Another important parameter that can drastically boost performance is the edge message ratio. In Figure 4 can be observed that the difference between the best and the worst edge message ratio is more than 1.4%. This indicates that the edge message ratio should be fine-tuned carefully. The most

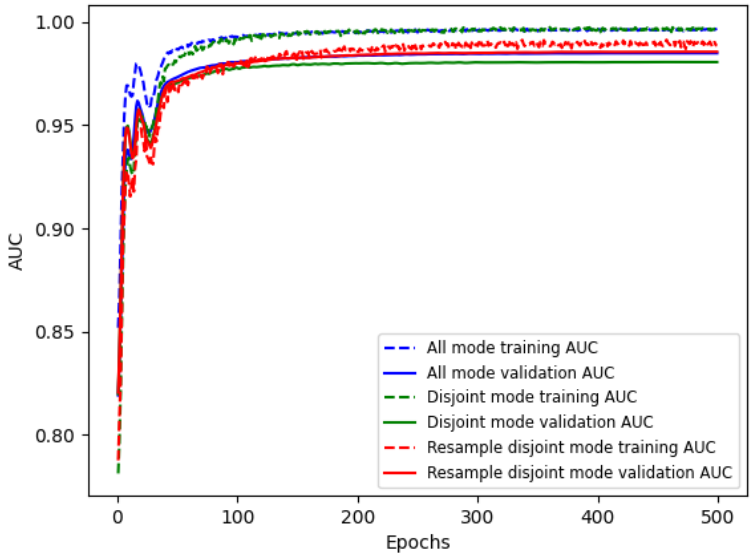

Figure 3: AUC scores on training and validation sets for `all mode`, `disjoint mode`, and `resample disjoint mode`. Model has $K = 2$, $o = 128$, and $edge\_message\_ratio = 0.8$ for the both disjoint modes. It is trained on CiteSeer dataset with $80/20$ split ratio.

straightforward approach is to do a complete search on some predefined values, but this is very time consuming. Another strategy is to randomly change the edge message ratio after each epoch. It surprisingly achieves almost as good results as a complete search (Figure 4), even though, it has much lower computational complexity. Based on the above observations, it is beneficial to estimate a good edge message ratio quickly. However, it turns out that the edge message ratio is notoriously hard to set. In the following paragraphs is illustrated that there are no patterns that depend on: data split ratio, model configuration, the average clustering coefficient of the network, and the average shortest path length of the network.

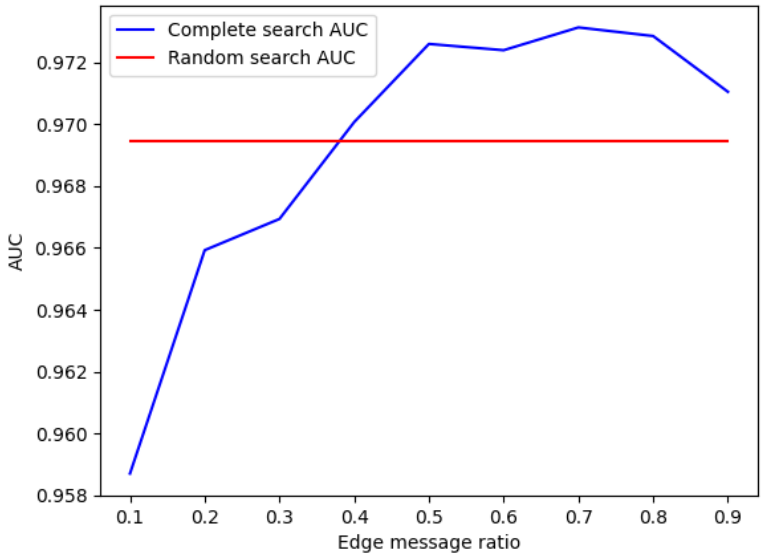

Figure 4: AUC scores for different edge message ratios and selection strategies on Cora dataset. Model has $K = 3$ and $o = 128$. It is trained with $80/10/10$ split ratio in the `resample disjoint mode`.

Data split ratio and model configuration The data split ratio controls how much information is available for training, so the model should prefer a different edge message ratio when given a new data split ratio. Although the optimal edge message ratio indeed changes, patterns are inconsistent across various model configurations and networks. In Figures 5 and 6, the optimal edge message ratios are plotted with the same configuration of the model but on different networks. The trend of the optimal edge message ratios fundamentally differs, although both datasets correspond to citation networks. Even on the same network, trends are often not coherent across distinct model configurations. Figures 6 and 7 display discrepancy between the optimal edge message ratio trends on PubMed network when the models are configured differently.

Average clustering coefficient and average shortest path length of the network The average clustering coefficient is a local measure, while the average shortest path length is a global one. These two characteristics should impact the optimal edge message ratio. For example, a higher average clustering coefficient implies there are a lot of edge triangles, so by removing some of the edges, it is expected that the distance between a pair of nodes does not drastically increase. This might indicate that networks with a higher average clustering coefficient would not need that high edge message ratio. By similar consideration, a higher average shortest path length should intuitively require a higher edge message ratio. However, Figures 8 and 9 depict that the average clustering coefficient and average shortest path length are uncorrelated with the optimal edge message ratio.

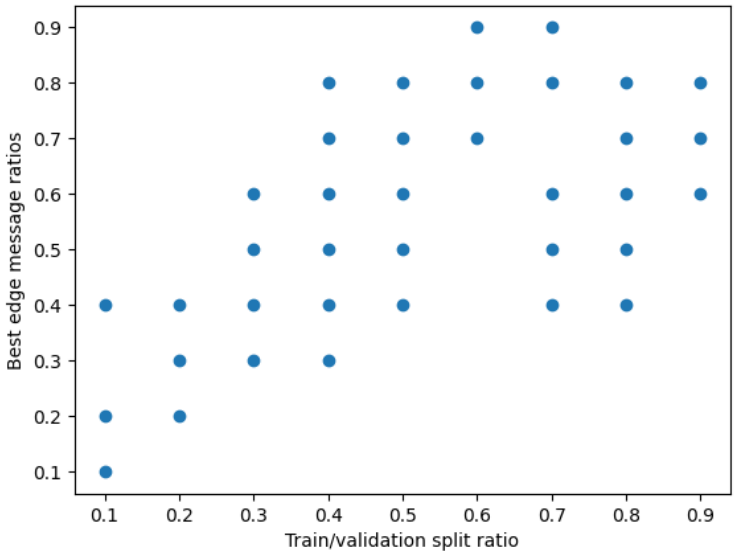

Figure 5: The optimal edge message ratios within 0.1% margin for different training/validation splits. Model has $K = 2$ and $o = 64$. It is trained on CiteSeer dataset in the `resample disjoint mode`.

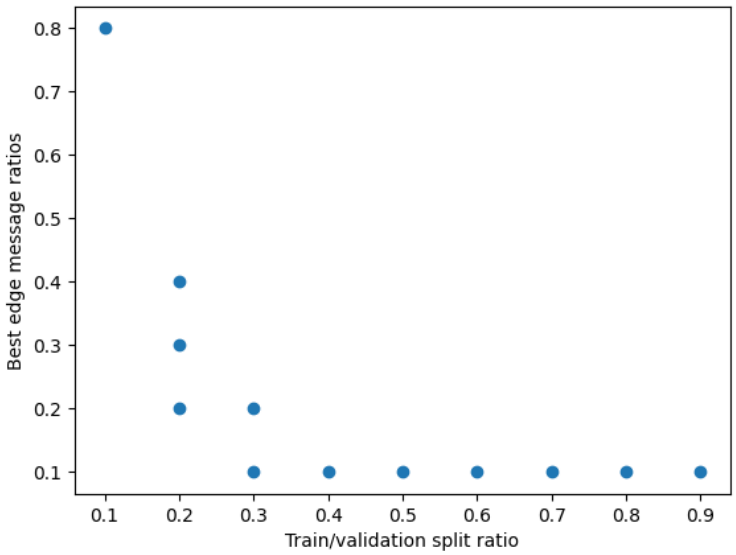

Figure 6: The optimal edge message ratios within 0.1% margin for different training/validation splits. Model has $K = 2$ and $o = 64$. It is trained on PubMed dataset in the `resample disjoint mode`.

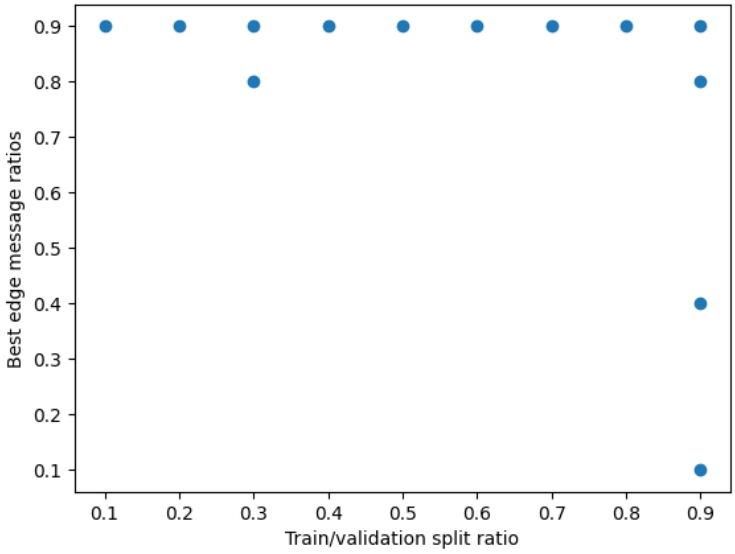

Figure 7: The optimal edge message ratios within 0.1% margin for different training/validation splits. Model has $K = 3$ and $o = 128$. It is trained on PubMed dataset in the `resample disjoint mode`.

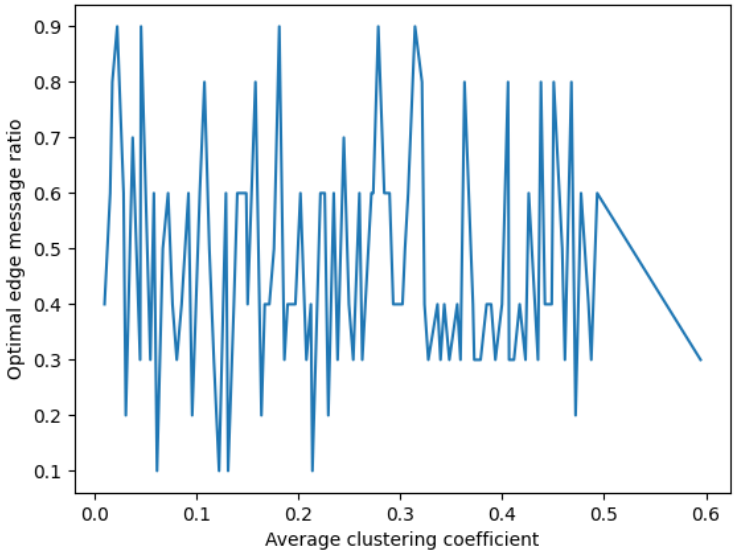

Figure 8: The optimal edge message ratio for different average clustering coefficients on artificial Watts-Strogatz small-world graphs [26] with 256 nodes and average degree 6. Model has $K = 2$ and $o = 32$. It is trained with $80/20$ split ratio in the `resample disjoint mode`.

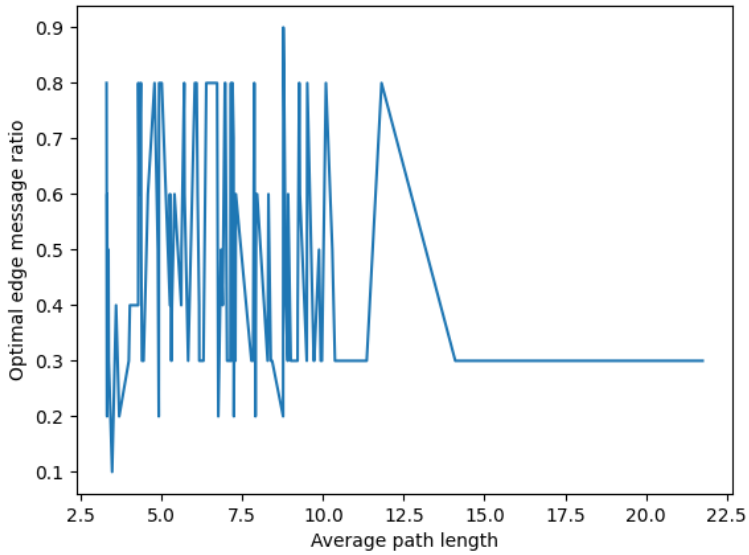

Figure 9: The optimal edge message ratio for different average shortest path lengths on artificial Watts-Strogatz small-world graphs [26] with 256 nodes and average degree 6. Model has $K = 2$ and $o = 32$. It is trained with $80/20$ split ratio in the `resample disjoint mode`.