SerialExperimentsOlga

Olga Yakobson

November 26, 2023

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Department of Mathematics,
Informatics and Statistics
Institute of Informatics

Artificial Intelligence and
Machine Learning

Bachelor's Thesis

SerialExperimentsOlga

Olga Yakobson

1. Reviewer Prof. Dr. Eyke Hüllermeier
Institute of Informatics
LMU Munich

2. Reviewer John Doe
Institute of Informatics
LMU Munich

Supervisors Jane Doe and John Smith

November 26, 2023

Olga Yakobson
SerialExperimentsOlga
Bachelor's Thesis, November 26, 2023
Reviewers: Prof. Dr. Eyke Hüllermeier and John Doe
Supervisors: Jane Doe and John Smith

LMU Munich
Department of Mathematics, Informatics and Statistics
Institute of Informatics
Artificial Intelligence and Machine Learning (AIML)
Akademiestraße 7
80799 Munich

Abstract

This thesis considers the effectiveness of regularization in the research field of graph classification and regression tasks to solve the problem of overfitting and over-smoothing. The performance of four types of regularization techniques is evaluated and shown to be ineffective on two types of graph neural network (GNN) network architectures graph convolutional network (GCN) and graph isomorphism network (GIN) across five different molecular Open Graph Benchmark (OGB) datasets. The similarity between GraphDropConnect (GDC) and DropEdge (DE) is highlighted. We show that the evaluated GNN architectures are unaffected by the problem of over-smoothing on graph-level tasks, and the overfitting aspect is shown to be unmitigated by regularization.

Contents

1 Introduction

The field of ML on graph-structured data has recently become an active topic of research. One reason for this is the wide range of domains and problems that are expressible using graphs. Regularization techniques are commonly employed in node-level prediction tasks across diverse networks to mitigate overfitting and over-smoothing, a condition where the performance and predictive power of a neural network (NN) do not improve or even get worse when more layers are added. These methods perturb the values and introduce randomness, leading to improved results. In graph-level tasks, however, the final output is a readout of aggregated nodes, hypothetically minimizing the impact of over-smoothing as the emphasis shifts from distinguishing individual nodes to capturing collective behavior.

1.1 Motivation

The exploration of controlled randomness in graph-level prediction tasks may initially appear unconventional since the focus shifts from individual nodes to capturing the collective behavior of the graph. The usual thinking about graph-level predictions might question if we need controlled randomness. Since the last step in graph-level prediction models is a readout operation, which combines the whole graph or a subgraph structure, the attention shifts from distinguishing individual nodes to the representation of the entire graph, making over-smoothing potentially less problematic. Despite this counterintuitive aspect, our study aimed to empirically validate the effectiveness of regularization techniques in graph-level prediction tasks. Our curiosity was motivated by a desire to challenge existing assumptions and gain a nuanced understanding of the relationship between regularization methods and graph-level predictions. In this pursuit, we aim to extend the boundaries of understanding regarding the impact of regularization on aggregated outcomes. One noteworthy technique that caught our attention was GraphDropConnect (GDC), which introduces stochasticity through adaptive connection sampling [Has+20]. By drawing different random masks for each channel and edge independently, GDC promises to provide nuanced improvements, surpassing the effectiveness of

traditional methods or even their combinations. Also, we want to better understand the role of controlled randomness in the context of overfitting and over-smoothing by evaluating four regularization techniques.

1.2 Research Questions

In this thesis, we will answer the following research questions to assess the relevance of regularization for graph-level prediction tasks:

1. Is regularization, specifically GDC, effective in solving the problem of overfitting and over-smoothing for graph-level prediction tasks?

2. Is there a difference in performance between graph convolutional network (GCN) and graph isomorphism network (GIN) architectures regarding performance with regularization techniques?

3. Are there similarities and differences between different regularization techniques in terms of performance?

1.3 Structure

Chapter 2: Related Work In order to answer our three research questions, we first take a closer look at three common prediction tasks in graph neural networks (GNNs) and give a general overview of how GNNs organize and process graph-structured data. We discuss the relation of the message-passing mechanism to the Weisfeiler-Lehman (WL)-test and give a formal definition and description of two GNNs architectures, GCN and GIN, before taking a closer look at typical issues occurring in GNNs. Finally, we present four regularization techniques, which mitigate two commonly occurring problems and Mean Average Distance (MAD), a measure for smoothness between nodes [Che+20].

Chapter 3: Implementation This section overviews the implementation of graph convolutions using TensorFlow's `gather` and `scatter` operations. We then explain

how we implement four regularization techniques by masking values during different convolution steps. We provide an intuitive approach for looking at GDC. Lastly, we discuss the benefits of using Open Graph Benchmark (OGB) datasets.

Chapter 4: Evaluation We start with an overview of used datasets and metrics before proceeding to the experimental setup. We then describe our parameter grid and explain how we choose the best set of hyperparameters. Finally, we present the experimental results and provide insights into how different numbers of layers and probability affect the performances of GNN and GIN and draw a conclusion from our findings.

Chapter 5: Conclusion Finally, the results of this thesis are summarized, and a brief outline of promising directions for future research is given.

2 Related Work

Before describing the problem, and later on the experimental setup, we first

1. Introduce three common prediction tasks in GNNs.

2. Give a general overview of how GNNs organize and process graph-structured data.

3. We further discuss the relation of the message-passing mechanism to the WL-test, an algorithm for inspecting whether two graphs are isomorphic.

4. Give a formal definition and description of two GNN architectures, which will be used in our experiments.

5. Discuss typical issues in GNNs and methods for addressing those issues.

6. Present four regularization techniques, which mitigate issues of overfitting and over-smoothing in GNNs.

2.1 Prediction Tasks and Typical Problems

Graphs naturally appear in numerous application domains, ranging from social analysis to bioinformatics to computer vision. A Graph $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is a set of $N = |V|$ nodes and $E \subseteq V \times V$ a set of edges betwen those nodes. The unique capability of graphs enables capturing the structural relations among data and thus allows the harvesting of more insights compared to analyzing data in isolation [Zha+19]. Graphs, therefore, can be seen as a general language for describing entities and relationships between those entities. Graph neural networks (GNNs) then organize graph-structured data to tackle various prediction and classification tasks. Typically, one is interested in one of the following three tasks:

1. **Link prediction:** Predict whether there are missing links between two nodes e.g., knowledge graph completion.

2. **Vertex classification & regression:** Predict a property of a node, e.g., categorize online users/items.

3. **Graph classification & regression:** Here, we are interested in classifying or predicting a continuous value for the entire graph, e.g., predicting a property of a molecule.

In this work, the main focus will be on node classification (NC), graph classification (GC), and graph regression (GR) for small- as well as medium-sized graphs.

2.2 Passing Messages in GNNs

Graphs, by nature, are unstructured. Vertices in graphs have no natural order and can contain any type of information. For machine learning algorithms to use graph-structured data, a mechanism is needed to organize them in a suitable way [Zho+20a; HYL17; Zha+19].

Message passing is a mechanism that embeds information about its neighborhood into every node [Xu+19; Zho+20a]. This can be done in several ways. One way of classifying a GNN is by looking at the underlying message-passing mechanism. This paper will look at a network where message passing is done via convolutions (graph convolutional network (GCN)). However, we will occasionally use the more general term message passing, as the separation is rather blurred, and message passing describes a neighborhood aggregation scheme seen as a generalization of other, more specific mechanisms.

Formally, message passing in a GNN can be described as using two functions: `AGGREGATE` and `COMBINE`. The expressive and representational power of a GNN can be determined by looking at the concrete functions and their properties used to implement aggregation and combination. `AGGREGATE` mixes the hidden representation of every node's neighborhood in every iteration. `COMBINE` then combines the mixed representation with the representation of the node. Each node uses the information from its neighbors to update its embeddings; thus, a natural extension is to use the

information to increase the receptive field by performing `AGGREGATE` and `COMBINE` multiple times.

$$a_v^k = \text{AGGREGATE}^{(k)}(\{h_u^{(k-1)} : u \in \mathcal{N}_{(v)}\})$$
$$h_v^{(k)} = \text{COMBINE}^{(k)}(h_v^{(k-1)}, a_v^{(k)})$$

For graph-level predictions, an additional READOUT- operation is used:

$$h_G = \text{READOUT}(\{h_v^{(K)} \mid v \in G\})$$

One useful type of information that the message-passing framework should be able to capture is the local graph structure. This can be done by choosing functions with appropriate properties. A more detailed explanation will follow in section 2.2.2. In spatial GNNs, we assume the similarity of neighbor nodes. To exploit this spatial similarity, we perform a composition by stacking multiple layers together, increasing the receptive field.

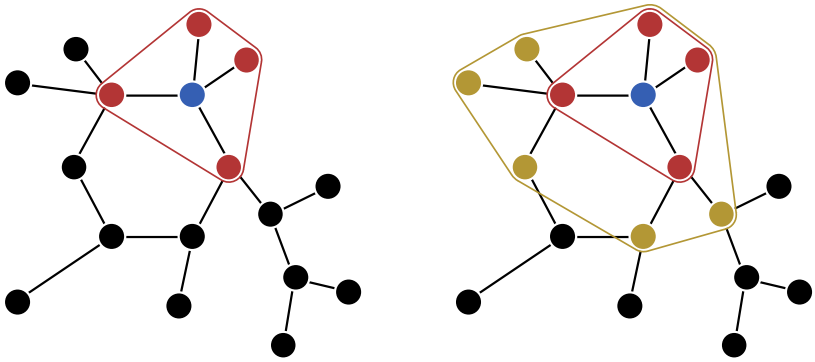

Fig. 2.1.: By performing aggregation k-times, we can reach and capture the structural information of the k-hop neighborhood

2.2.1 Weisfeiler-Lehman Graph Colorings

The Message passing mechanism is closely related to how the Weisfeiler-Lehman (WL) test [WL68; DMH20; HV22] works, an algorithm for deciding if two graphs are isomorphic. Before describing the algorithm, we introduce notations and preliminaries.

Let $G = (V, E, X)$ denote an undirected graph, where $V = \{v_1, ..., v_n\}$ is a set of $N = |V|$ nodes and $E \subseteq V \times V$ a set of edges between those nodes. For simplicity, we represent an edge $\{v, u\} \in E$ by $(v, u) \in E$ or $(u, v) \in E$. $X = [x_1, ..., x_n]^T \in \mathbb{R}^{n \times d}$ is the node feature matrix, where $n = |V|$ is the number of nodes and $x_v \in \mathbb{R}^d$ represents the *d*-dimensional feature of node v. $\mathcal{N}_v = \{u \in V \mid (v, u) \in E\}$ is the set of neighboring nodes of node v. A multiset is denoted as $\{\!\{...\}\!\}$ and formally defined as follows.

Definition 2.1 (Multiset)**.** A multiset is a generalization of a set allowing repeating elements. A multiset $\mathcal{X}$ can be formally represented by a 2-tuple as $X = (S_X, m_X)$, where S_X is the underlying set formed by the distinct elements in the multiset and $m_X : S_X \to \mathbb{Z}^+$ gives the multiplicity (i.e., the number of occurrences) of the elements. If the elements in the multiset are generally drawn from a set X (i.e., $S_X \subseteq \mathcal{X}$), then $\mathcal{X}$ is the universe of X, and we denote it as $X \subseteq \mathcal{X}$ for ease of notation.

Definition 2.2 (Isomorphism)**.** Two Graphs $\mathcal{G} = (V, E, X)$ and $\mathcal{H} = (P, F, Y)$ are *isomorphic*, denoted as $\mathcal{G} \simeq \mathcal{H}$, if there exists a *bijective* mapping $g : V \to P$ such that $x_v = y_{g(v)}$, $\forall v \in V$ and $(v, u) \in E$ iff $(g(v), g(u)) \in F$.

The 1-dimensional WL Algorithm (Color Refinement)

In the 1-dimensional WL algorithm (1-WL), a label, called *color*, c_v^0 is assigned to each vertex of a graph. Then, in every iteration, the colors get updated based on the multiset representation of the neighborhood of the node until convergence. If, at some iteration, the colorings of the graphs differ, 1-WL decides that the graphs are not isomorphic.

$$c_v^l \leftarrow \text{HASH}\left(c_v^{l-1}, \{\!\{c_u^{l-1} \mid u \in \mathcal{N}_v\}\!\}\right)$$

Algorithmically, this can be expressed as follows:

Algorithm 1 1-dim. WL (color refinement)

Input: $G = (V, E, X_V)$

1: $c_v^0 \leftarrow hash(X_v)$ for all $v \in V$
2: **repeat**
3: $\quad c_v^l \leftarrow hash(c_v^{l-1}, \{\!\{c_w^{l-1} : w \in \mathcal{N}_G(v)\}\!\})$ for all $v \in V$
4: **until** $(c_v^l)_{v \in V} = (c_v^{l-1})_{v \in V}$
5: **return** $\{\!\{c_v^l : v \in V\}\!\}$

The 1-WL is a heuristic method that can efficiently distinguish a broad class of non-isomorphic graphs [BK79]. However, the algorithm fails to classify simple shapes as non-isomorphic in some corner cases. This is the case for non-isomorphic graphs with the same number of nodes and equivalent sets of node degrees, as shown in fig. 2.3.

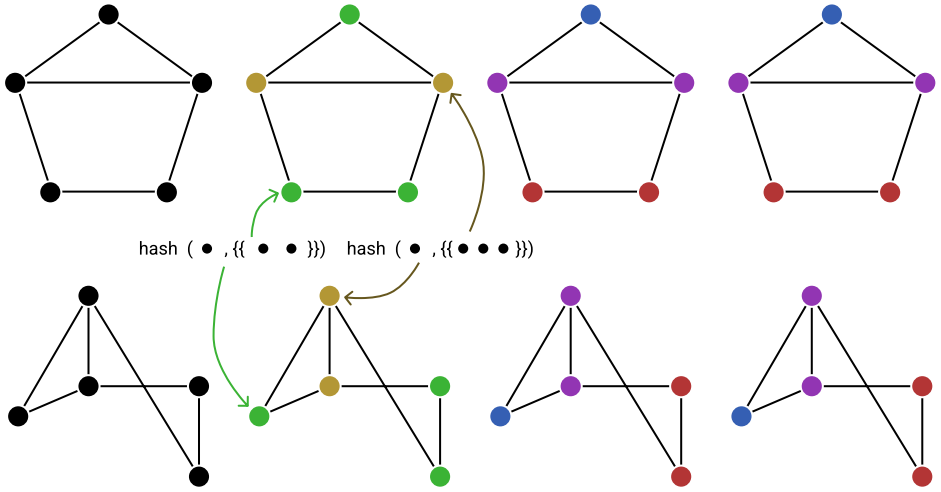

Fig. 2.2.: Two isomorphic graphs. 1-WLassigns the same representation to those graphs.

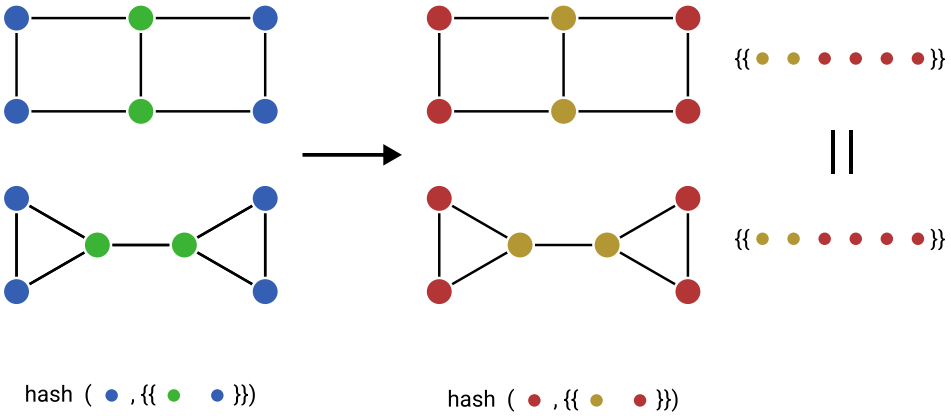

Fig. 2.3.: 1-WL assigned the same labeling to two non-isomorphic graphs [LYJ22].

2.2.2 GNN Architectures in this Paper

In the following section, we briefly introduce and motivate the choice of two types of networks, which we have chosen to experimentally verify the efficacy of several regularization techniques, which will be discussed in section 2.2.4.

Since all GNNs incorporate message passing in a way, we decided to choose two architectures for our experiments, which are powerful, efficient, scalable, and broadly used.

Graph Convolutional Network (GCN)

Graph convolutional network (GCN) was originally proposed by Kipf and Welling [KW17] to tackle the problem of semi-supervised node classification, where labels are available for a small subset of nodes. GCN is a simple but powerful architecture that scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes.

A GCN can formally be expressed via the following layer-wise propagation rule:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

Where $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph $\mathcal{G}$ with added self-connections. I_N is the identity matrix. $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and W^l is a layer-specific trainable weight-matrix. $\sigma(\cdot)$ denotes an activation function, such as $ReLU(\cdot) = max(0, \cdot)$. $H^l \in N \times D$ is the matrix of activations in the l^{th} layer; $H^0 = X$.

Because we consider every neighbor to be of equal importance, normalization is accomplished by dividing by the number of neighbors, one can view this operation as performing an element-wise mean-pooling [Xu+19].

$$h_v^{(k)} = \text{ReLU}(\text{W} \cdot \text{MEAN}\{h_u^{k-1} \mid \forall u \in \mathcal{N}_{(v)} \cup \{v\}\})$$

An application of a two-layer GCN is given by:

$$Z = f(X, A) = \text{softmax}(\hat{A}\ \text{ReLU}(\hat{A}XW^0)W^l)$$

where $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is calculated in a preprocessing step. The model uses a single weight matrix per layer and deals with varying node degrees by appropriately normalizing the adjacency matrix. This model consisting of a 2-layer GCN performed well in a series of experimental tasks, including semi-supervised document classification, semi-supervised node classification in citation networks, and semi-supervised entity classification in a bipartite graph extracted from a knowledge graph. The prediction accuracy was evaluated on a set of 1000 examples, and additional experiments on deeper models with up to 10 layers have also been provided. Being capable of encoding both graph structure and node features, GCN outperformed numerous related methods by a significant margin [KW17]. Graph convolutional networks (GCNs) are widely and successfully used today in many fields due to their simplicity and scalability.

Graph Isomorphism Network (GIN)

To overcome the lack of expressivity of popular GNN architectures, Xu et al. [Xu+19] designed a new type of GNN, GIN. They prove that GINs are strictly more expressive than a variety of previous GNN architectures and that they are, in fact, as powerful as the commonly used 1-dimensional Weisfeiler-Lehman (WL)-test.

Two requirements must be met for a network to have the same expressive and representational power as the WL isomorphism test:

1. The framework must be able to represent the set of feature vectors of a given nodes neighbors as a multiset.

2. Choosing an injective function for the aggregation step. Such a function would never map two different neighborhoods to the same representation.

The more discriminative the multiset function is, the more powerful the representational power of the underlying GNN.

Formally, a graph isomorphism network (GIN) can be expressed as follows:

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left((1+\epsilon^{(k)})\cdot h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right)$$

The choice of such an architecture is motivated by the necessity to learn two functions with certain properties, f and ϕ. This task can be accomplished using a multilayer perceptron (MLP). The following lemma and corollary, proven by Xu et al. [Xu+19], show the properties and application of the functions:

Theorem 2.3. *Let $A : G :\to \mathbb{R}^d$ be a GNN. With a sufficient number of GNN layers, A maps any grphs G_1 and G_2 to different embeddings, the WL-test of isomorphism decides as non-isomorphic, to different embeddings if the following conditions hold:*

(a) A aggregates and updates node features iteratively with

$$h_v^{(k)} = \phi(h_v^{(k-1)}, f(\{h_u^{(k-1)} \mid u \in \mathcal{N}_{(v)}\}))$$

where the functions f, which operates on multisets and, and ϕ are injective.

(b) A's graph-level readout, which operates on the multiset of node features $\{h_v^{(k)}\}$, is injective.

Lemma 2.4. *Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that $h(X) = \sum_{x \in X} f(x)$ is unique for each multiset $X \subseteq \mathcal{X}$ of bounded size. Moreover, any multiset function g can be decomposed as $g(X) = \phi(\sum_{x \in X} f(x))$ for some function ϕ.*

Corollary 2.5. *Assume $\mathcal{X}$ is countable. There exists a function $f : \mathcal{X} \to \mathbb{R}^n$ so that for infinitely many choices of ϵ, including all irrational numbers, $h(c, X) = (1 + \epsilon) \cdot f(c) + \sum_{x \in X} f(x)$ is unique for each pair (c, X), where $c \in \mathcal{X}$ and $X \subseteq \mathcal{X}$ is a multiset of bounded size. Moreover, any function g over such pairs can be decomposed as $g(c, X) = \varphi(1 + \epsilon)\dot{f}(c) + \sum_{x \in X} f(x)$ for some function φ.*

graph isomorphism network is as powerful as 1-dimensional WL

GIN is a neural network-based approach designed to handle graph data and detect graph isomorphisms. It operates on each vertex and updates its representation based on its features and the aggregated features of its neighbors. A fundamental similarity exists between the GIN and how the 1-WL algorithm works.

2.2.3 Weaknesses and Obstacles in GNNs

Because of the way GNNs operate, they tend to suffer from two main obstacles: Overfitting and over-smoothing.

Overfitting hinders the generalization ability of a neural network (NN), making it perform poorly on previously unseen data. This occurs especially when using small datasets since the model tends to 'memorize' instead of learning the pattern.

Over-smoothing is a condition where the performance and predictive power of a NN do not improve or even get worse when more layers are added. This happens because by stacking multiple layers together, aggregation is being performed repeatedly. This way, the representation of a node is being smoothed, i.e., mixed with features of very distant, possibly unrelated nodes. Over-smoothing is a problem mainly for node classification tasks. There is a trade-off between the expressiveness of the model (capturing graph structure by applying multiple layers) and over-smoothing,

which leads to a model where nodes have the same representation because they all converge to indistinguishable vectors [Zho+20b; Has+20].[1]

A closer examination of underlying causes of over-smoothing was conducted by Chen et al. [Che+20], who suggested that it is not message passing itself but the type of interaction between nodes that causes this issue. For node classification (NC) tasks, intra-class communication (interaction between two nodes sharing the same class) is useful (signal). In contrast, inter-class communication (the communication between two nodes sharing different labels) is considered harmful because it brings interference noise into the feature representations by mixing unrelated features, making unrelated nodes more similar. Because of that, the quality of shared information is essential and should, therefore, be considered a benchmark for improvement.

2.2.4 Regularization Techniques

Kukacka et al. [KGC17] define regularization as any supplementary technique that aims at making the model generalize better, i.e., produce better results on the test set, which can include various properties of the loss function, the loss optimization algorithm, or other techniques.

One subgroup of regularization is via data, where the training set $\mathcal{D}$ is transformed into a new set $\mathcal{D}_R$ using some stochastic parameter π, which can be used in various ways, including to manipulate the feature space, create a new, augmented dataset or change, e.g., thin out the hidden layers of the NN.

An example of such a transformation is the corruption of inputs by Gaussian noise.

$$\tau_0(x) = x + \theta, \theta \backsim \mathcal{N}(0, \Sigma)$$

This work focuses on stochastic regularization techniques, which perform data augmentation in one way or another and whose main benefits lie in alleviating overfitting and over-smoothing [Has+20]. We will use the following notation:

[1] In spatial GNNs we make the assumption of relatedness by proximity.

Notation	Description
$H^{(l)} = [h_0^{(l)}, \dots h_n^{(l)}]^T \in \mathbb{R}^{n \times f_l}$	Output of the l-th hidden layer in GNN
n	Number of nodes
f_l	The number of output features at the l-th layer
$H^0 = X \in \mathbb{R}^{n \times f^0}$	Input matrix of node attributes
f_0	Number of nodes features
$W^l \in \mathbb{R}^{f_l \times f_{l+1}}$	The GNN parameters at the l-th layer
$\sigma(\cdot)$	Corresponding activation function
$\mathcal{N}(v)$	Neighborhood of node v
$\tilde{\mathcal{N}}(v) = \mathcal{N}(v) \cup v$	$\mathcal{N}(v)$ with added self-connection
$\mathfrak{N}(\cdot)$	Normalizing operator
$\odot$	Hadamard product

DropOut (Srivastava et al.)

DropOut (DO)[Sri+14] randomly removes elements of its previous hidden layer $H^{(l)}$ based on independent Bernoulli random draws with a constant success rate at each training iteration:

$$H^{(l+1)} = \sigma(\mathfrak{R}(A)(Z^{(l)} \odot H^{(l)})W^{(l)})$$

where Z^l is a random binary matrix with the same dimensions as H^l, whose elements are samples of Bernoulli(π).

During training, the random drop of units (along with their connections) from the neural network prevents units from co-adapting too much. A neural net with n units can be seen as a collection of 2^n possible networks. Applying dropout with a certain probability π can be interpreted as sampling "thinned" networks from all possible 2^n networks. Since averaging over all possible networks is computationally expensive, an approximation for combining the prediction is used. This averaging method entails using a single neural net with weights, which are scaled-down weights obtained during training time.

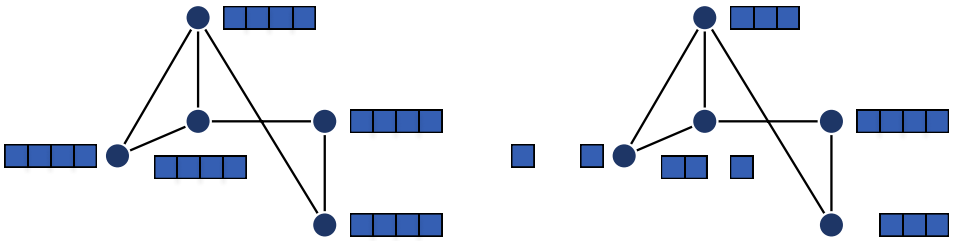

Fig. 2.4.: DropOut (DO) preserves connections between nodes as well as the nodes itself, unless we chose a large probability π, which drops all of the nodes features.

DropEdge (Rong et al.)

DropEdge (DE) [Ron+20] randomly removes a certain number of edges from the input graph at each training epoch and can be formally expressed as follows:

$$H^{(l+1)} = \sigma(\mathfrak{R}(A \odot Z^{(l)})H^{(l)}W^{(l)})$$

The random binary mask Z^l has the same dimensions as A. Its elements are the random samples of Bernoulli(π) where their corresponding elements in A are non-zero and zero everywhere else.
Message passing in GNNs happens along the edges between neighbors. Randomly removing edges makes the connections more sparse, which leads to slower convergence time and thus prevents the network from over-smoothing and allows for a deeper architecture. Intuitively, this makes sense since removing an edge means that the node, previously connected by that edge, stops being a neighbor. Consequently, the representation of this former neighbor does not get mixed with the representation of the node. DE also acts like a data augmenter since we manipulate/change the underlying graph data by randomly dropping edges.

Since the data is now augmented with noise, it is harder for the network to overfit it by "memorizing" rather than learning complex relationships. The combination of DO and DE reaches a better performance in terms of mitigating overfitting in GNNs than DE on its own.

NodeSampling (Chen et al.)

This method of regularization, also known as FastGCN [CMX18], was developed to improve the GCN [KW17] architecture and to address the bottleneck issues of

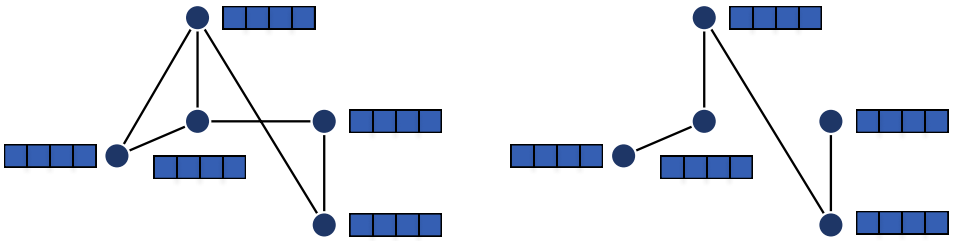

Fig. 2.5.: DropEdge (DE) preserves nodes and all of nodes featurs, but randomly removes edges, leading to a smaller number of neighbors, which results in slower convergence times and allows for architectures with more hidden layers.

GCNs caused by the recursive expansion of neighborhoods. It reduces the expensive computation in batch training of GNN by relaxing the requirement of simultaneous availability of test data. Graphs can be very large and require large computational and processing capacities. By randomly dropping out nodes, we reduce the amount of data so that it alleviates the expensiveness of the computation and reduces the bottleneck issue while preserving important relations.

$$H^{(l+1)} = \sigma(\mathfrak{R}(A)diag(z^{(l)})H^{(l)}W^{(l)})$$

Here, $z^{(l)}$ is a random vector whose elements are drawn from Bernoulli(π). This is a special case of DO since all of the output features are either kept or completely dropped.

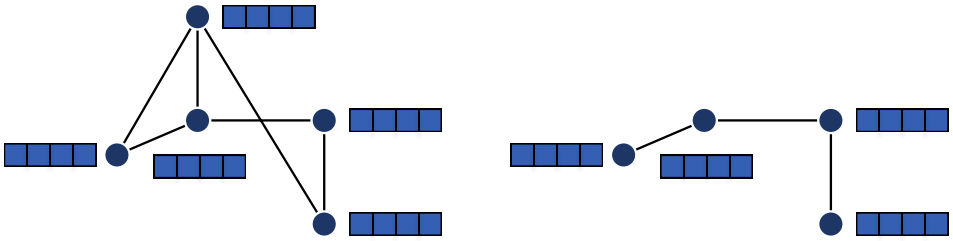

Fig. 2.6.: In NodeSampling (NS), a node is either removed or preserved along with the whole feature vector with a certain probability π.

GraphDropConnect (Hasanzadeh et al.)

Finally, GDC [Has+20], which can be seen as a generalization of all the above-proposed methods, is a stochastic regularization approach, which is [Has+20] the

most effective among all the above and even more effective than the combination of DO and DE. The regularization is done via adaptive connection sampling and can be interpreted as an approximation of Bayesian GNNs.

$$H^{(l+1)}[:,j] = \sigma\left(\sum_{i=1}^{f_l} \mathfrak{R}\left(A \odot Z_{i,j}^{(l)}\right) H^{(l)}[:,i] W^{(l)}[i,j]\right)$$
$$\text{for } j = 1, ..., f_{l+1}$$

where f_l and f_{l+1} are the number of features at layers l and $l+1$, respectively, and $Z_{i,j}^{(l)}$ is a sparse random matrix (with the same sparsity as A) whose non-zero elements are randomly drawn from Bernoulli(π_l), where π_l can be different for each layer. GDC is a regularization technique that combines all of the above by drawing different random masks for each channel and edge independently, which yields better performance results than all of the previous methods or even combinations of them. GDC, as it is expressed in the formula above, has not been implemented and evaluated yet. Instead, a special case of GDC has been implemented:

Under the assumption that $Z_{i,j}^{(l)}$ are the same for all $j \in \{1, 2, \ldots, f_{l+1}\}$, we can omit the indices of the output elements at layer $l+1$ and rewrite the above formula as follows:

$$H^{(l+1)} = \sigma(\sum_{i=1}^{f_l} \mathfrak{R}(A \odot Z_i^{(l)}) H^{(l)}[:,i] W^{(l)}[i,:])$$

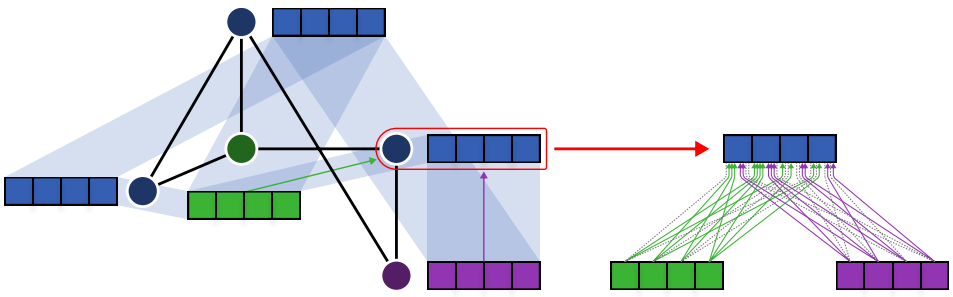

Fig. 2.7.: GraphDropConnect (GDC) can be considered duplicating every existing edge between features of the feature-vectors of existing nodes and then randomly removing every edge with a certain probability π before the convolution.

All the methods are somewhat related and share some similarities [Ron+20]. DropOut (DO) has been shown to be successful in alleviating overfitting by perturbing the feature matrix and setting some entries to zero. The issue of over-smoothing is not affected by this measure. DropEdge (DE) achieved great results in reducing overfitting and over-smoothing.

Intuitively, this makes sense because smoothing comes from the aggregation of the neighbors of a certain node, and by dropping the connections to some neighbors, the feature vectors of those neighbors are no longer aggregated and combined with the hidden representation of the node.

NodeSampling (NS) is a special case of DropOut (DO), as all of the output features for a node are either completely kept or dropped while DO randomly removes some of these related output elements associated with the node. Also, along with the dropped node, the edges of this node are dropped. However, The method is node-oriented, and the edge-drop is a "side-effect".

GraphDropConnect (GDC) generalizes existing stochastic regularization methods for training GNNs and is effective in dealing with overfitting and over-smoothing. GDC regularizes neighbourhood aggregation in GNNs at each chanel separately. This prevents connected nodes in the graph from having the same learned representations in GNN layers; hence better improvement without serious over-smoothing can be achieved [Has+20].

2.3 Assessment of Graph Regularization Approaches

To make systematic and quantitative statements about the positive effects of over-smoothing by using different regularization techniques, one has to be able to monitor the smoothness of nodes at different execution steps during training. Therefore, choosing a suitable metric is of great importance, as it helps to assess the extent of the effect produced by various regularization techniques and compare them against each other in terms of efficacy.

MAD [Che+20] is a metric for smoothness, the similarity of graph node representations. In that sense, over-smoothness is the similarity of node representations among different classes. While smoothing to some extent is desired (we assume

spatial similarity between nodes), mixing features of nodes with different labels over several iterations leads to over-smoothing.

It is, therefore, important to differentiate between different types of messages between nodes. Signal/information is the messaging of nodes sharing the same class/label, i.e., intra-class communication, and noize denotes intra-class communication. Having too many inter-class edges leads to much noise by incorporating messages from other classes, which results in over-smoothing.

Because of that, measuring the quality of the received messages is crucial. A way to do that is to consider the information-to-noise ratio, i.e., the fraction of intra-class node pairs and all node pairs with interaction through the GNN model. That way, it is possible to differentiate between remote and neighboring nodes and calculate the **MADGap (MADGap)**, positively correlating with a model's accuracy.

MAD is calculated as follows:

Given the graph representation matrix $H \in \mathbb{R}^{n \times h}$ we first obtain the distance matrix $D \in \mathbb{R}^{n \times n}$ for H by computing the cosine distance between each node pair.

$$D_{i,j} = 1 - \frac{H_{i,:} \cdot H_{j,:}}{\mid H_{i,:} \mid \cdot \mid H_{j,:} \mid} \quad i, j \in [1, 2, \ldots, n],$$

where H_k is the k-th row of H. The reason to use cosine distance is that cosine distance is not affected by the absolute value of the node vector, thus better reflecting the smoothness of graph representation. Then we filter the target node pairs by element-wise multiplication D with a mask matrix M^{tgt}

$$D^{tgt} = D \odot M^{tgt},$$

where $\odot$ denotes the element-wise mutliplication: $M^{tgt} \in \{0, 1\}^{n \times n}; M^{tgt}_{i,j} = 1$ only if node pair (i, j) is the target one. Next we access the average distance $\bar{D}^{tgt}$ for non-zero values along each row in D^{tgt} :

$$\bar{D}_t^{tgt} = \frac{\sum_{j=0}^{n} D_{i,j}^{tgt}}{\sum_{j=0}^{n} \mathbb{1}(D_{i,j}^{tgt})}$$

where where 1(x) = 1 if x > 0 otherwise 0. Finally, the MAD value given the target node pairs is calculated by averaging the non-zero values in tgt MAD gives access to the smoothness of a node and pairs of nodes throughout iterations, which makes it easy to "track down" over smoothing. First, the cosine similarity is calculated, showing how similar the corresponding feature vectors are. By subtracting the cosine similarity from one, we get the cosine distance, which tells us the difference between the nodes.

3 Implementation

This section overviews the implementation and aims to motivate the choice of libraries and frameworks. We also deliver an in-depth explanation of the implementation of GDC, as it is the main focus of our work.

3.1 Scope and Limitations

Our research is concerned only with graph-level prediction tasks on two types of GNNs, GCN and GIN. We implemented both networks as described by Kipf and Welling [KW17] and Xu et al. [Xu+19], respectively. We evaluate five different scenarios, among them four different regularization techniques DO, NS, DE, and GDC, and we also train the networks using no regularization. GDC is implemented as described in eq. (3.2) since this allows for better time and space complexity, which was also done by Hasanzadeh et al. [Has+20]. We perform our experiments using five datasets from the OGB dataset collection [Hu+20], all within the molecular realm. Two datasets are for classification, and the rest for regression tasks.

3.2 Sparse Implementation of Graph Dropout Layers

This section is concerned with implementing our regularization techniques and explaining how those have been embedded into a convolutional step. First, we take a look at TensorFlow's gather and scatter operations. Then, we explain how we use these operations to implement sparse graph convolutions. For that, an algorithmic description and a text explanation will be presented. We then explain the implementation of the sparse graph convolutions before we explain how regularization is embedded in a convolutional step.

3.2.1 Gather and Scatter

In GNNs, communication between nodes is achieved via message passing (see section 2.2). For messages to be passed along between nodes, we need first to extract and second to pass information on. For this purpose, we utilize two of TensorFlow's functions, $gather$ and $scatter$ [He+07; DMH20].

Formally, the $gather$ operator takes two inputs: A list X of n row vectors and a list R of m pointers into X. It returns a list X of m row vectors $X[i] = Z[R[i]]$ for $i \in [m]$. The gather operation in TensorFlow extracts specific elements from a tensor along a given axis. Given an input tensor and a list of indices, the gather operation selects elements based on the provided indices from the input tensor. Given a tensor representing node features in a graph and a list of node indices, the gather operation extracts the corresponding node features. The gather operation can extract node features, adjacency information, or other relevant data based on specific graph node indices.

The $scatter_\Sigma$ operator can be understood as the opposite of $gather$. $scatter_\Sigma$ takes a list X of m row vectors and a list R of m pointers from the range $[n]$. It returns a list Z of n row vectors $Z[i] = \sum_{j \in [m] \wedge R[j]=i} X[j]$ for $i \in [n]$. The scatter operation in TensorFlow is the inverse of the gather operation. It updates the elements of an existing tensor based on the provided indices. Given an input tensor, a list of indices, and a tensor containing values, the scatter operation replaces elements in the input tensor at the specified indices with the corresponding values from the values tensor. Gather and scatter operations are crucial for tasks involving graph neural networks GNNs where information aggregation and dissemination across nodes are essential. Information from nodes or subgraphs is gathered and aggregated for graph classification tasks to represent the entire graph. GNNs can use gather operations to pool information from nodes and perform graph-level predictions.

3.2.2 Sparse Implementation of Graph Convolutions

In our study, the fundamental mechanism for information exchange among nodes in both GNNs is implemented through the message-passing mechanism via graph convolutions. GNNs capture complex relationships in convolutional steps within graph-structured data. The detailed algorithm for implementing this message-passing mechanism using graph convolutions is described in Algorithm 2. Algorithm

2 outlines the Sparse Graph Convolution operation, where information is efficiently exchanged between nodes utilizing gather and scatter operations. In this algorithm, the gather operation extracts feature information from neighboring nodes, while the scatter operation aggregates and updates the node representations. The resulting updated node features are the foundation for subsequent graph convolutional layers.

The fundamental operation in our GNNs is the graph convolution operation, which allows nodes to aggregate and exchange information with their neighboring nodes. The sparse implementation of graph convolutions can be described algorithmically as follows:

Algorithm 2 Sparse Graph Convolution using Gather and Scatter

1: **function** $SparseGraphConvolution(X \in \mathbb{R}^{n \times d}, R \in [n]^{m \times 2})$
2: $X_a \coloneqq gather(Z, R[:, 0])$ ▷ Gather Operation: $\mathbb{R}^{n \times d} \times [n]^m \to \mathbb{R}^{m \times d}$
3: $X_b \coloneqq gather(Z, R[:, 1])$
4: $X_{\Sigma a} \coloneqq scatter_{\Sigma}(X_a, R[:, 1])$ ▷ Scatter Operation: $\mathbb{R}^{m \times d} \times [n]^m \to \mathbb{R}^{n \times d}$
5: $X_{\Sigma b} \coloneqq scatter_{\Sigma}(X_b, R[:, 0])$
6: $X_{\text{conv}} \coloneqq \sigma\left(X + X_{\Sigma a} + X_{\Sigma b}\right)$ ▷ Result with added self-connections
7: **return** X_{conv} ▷ Result with added self-connections
8: **end function**

Note that algorithm 2 assumes bidirectional edges. Hence, the gather and scatter operations are performed twice. This operation can be efficiently implemented using gather and scatter operations, enabling the network to learn expressive node representations while considering the graph's topology. In graph convolutions, the gather operation extracts feature information from neighboring nodes. For each target node v_i, the gather operation assembles feature vectors from its neighboring nodes in the graph (lines 2 and 3). These gathered features are then scattered into the feature vector of each target v_i. The scatter operation aggregates the gathered features by adding them to each target's feature vector. It takes the computed features and distributes them back to the corresponding nodes in the graph (lines 4 and 5). This step ensures that the updated node representations, enriched with information from neighboring nodes, are propagated throughout the graph.

Lastly, the feature vector of the node itself is added, forming the basis for computation in the subsequent convolutional layer (line 6). By selecting and aggregating features from neighboring nodes using the gather and scatter operations, the model

captures the local neighborhood information critical for graph-based tasks. With each convolution, one additional neighborhood can be captured. This way, both local and global patterns are captured.

3.2.3 Adding Dropout to Sparse Graph Convolutions

Four of our regularization techniques, described in section 2.2.4, have been integrated into this gather/scatter-based graph convolution. We implement those techniques using a unified framework. We are using just two parameters, *row-wise?* and *gather-first?*. We describe all four dropout techniques in section 2.2.4.

The *row-wise?* parameter tells us if the dropping is done for entire rows or just for some values in the rows of a feature matrix. The feature matrix X, where each row represents a node, and each entry in such a row represents a feature of the node. This vector is masked when we perform DO or NS. Each row of the gathered matrices X_a and X_b contains the feature vector of the start node of an edge. We perform DE and GDC by masking those gathered matrices. Masking is implemented as element-wise multiplication with the corresponding vectors.

So, the whole node is dropped when *row-wise?* is true. This is the case for Node-Sampling (NS) and for DropEdge (DE). NS removes the whole node, along with every feature of this node, and DE is concerned with dropping edges, i.e., connections to the entire node, not only to some of the features. So, this type of dropout is also performed row-wise. DropOut (DO) drops just some features of a gíven node, so the *row-wise?* parameter is set to false. This is also the case for GDCs as it draws random masks, i.e., performs drops for each channel independently.

The *gather-first?* parameter tells us whether the mask is applied before or after we gather values. When *gather-first?* is set to true, we gather the values and then apply the mask, meaning that the mask is applied to X_a and X_b. This is the case for DE and GDC. When *gather-first?* is set to false, which is the case for DO and NS, we apply the mask and then gather the values. That means the mask is applied directly to the $X \in \mathbb{R}^{n \times d}$ before we gather values in line 2.

To sum up, *gather-first?* is the parameter that regulates which matrix the randomly drawn mask is applied, and *row-wise?* determines if the “random drop” is valid for the whole row. The permutations of those two parameters result in four of our

regularization techniques section 3.2.3. The different types of regularization are all performed in each gather/scatter-based convolutional step by masking some values from the feature vector X or X_a and X_b.

We implement regularization by integrating the masking of values in each gather/scatter-based convolutional step.

Regularization	**row-wise?**	**gather-first?**
DropOut (DO)	false	false
DropEdge (DE)	true	true
NodeSampling (NS)	true	false
GraphDropConnect (GDC)	false	true

Since our research is first and foremost focused on validating the results of mainly GCN, we aim to elicit a better understanding of this technique. We look at the two proposed variants of GDC and provide an intuition for both. The second, more relaxed version of GDC, as described in fig. 3.1 was implemented for our research, as it allowed for better time and space complexity and was also the one that was originally evaluated by [Has+20].

As stated previously in section 2.2.4, this version of GDC allows drawing different random masks for each channel and edge independently, giving more flexibility and increasing the time and space complexity.

$$H^{(l+1)}[:,j] = \sigma\left(\sum_{i=1}^{f_l} \mathfrak{R}\left(A \odot Z_{i,j}^{(l)}\right) H^{(l)}[:,i] W^{(l)}[i,j]\right) \quad \text{for } j = 1, ..., f_{l+1} \tag{3.1}$$

Here, we calculate the new feature matrix $H^{(l+1)}$ by stacking the column vectors of each iteration. One can think of the calculations that are being performed as a 4-dimensional matrix with the dimensions $n \times n \times f_l \times f_{l+1}$

To understand what is being done, we can look at what is being performed when calculating one column of the resulting matrix. First, a random mask is applied to the connection from the i-th to the j-th node. Regarding node features communicating with each other, we look at the edge between the i-th and j-th features between connected nodes. By applying the random mask, we drop those connections selectively. Because we sample the random binary mask Z f_{l+1} times, one time for

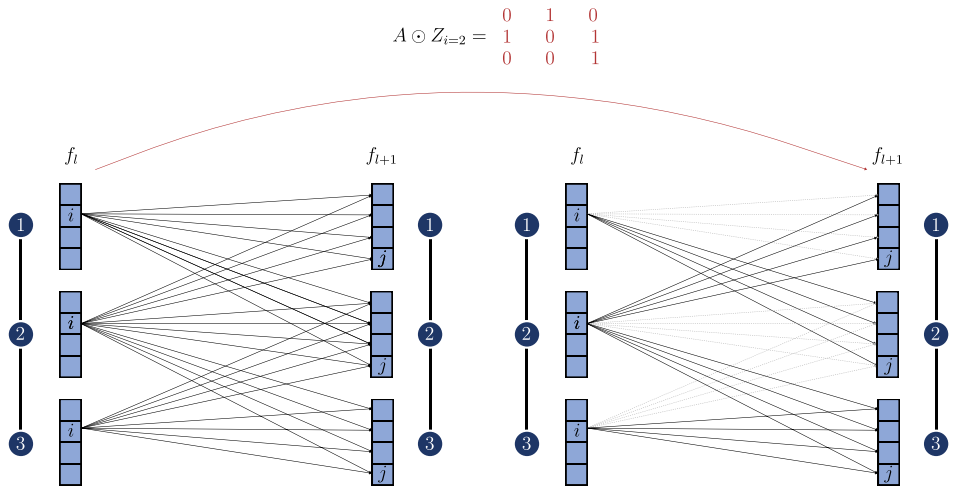

Fig. 3.1.: Originally proposed GDCNote: self connection are assumed

every feature in the $l+1$-th iteration, we differentiate between the connection $i \to j$ between two nodes and the connection $j \to i$ between the same two nodes. Thus, the same edge can be dropped as a connection and remain as a connection in the opposite direction. The masked adjacency is then multiplied by the corresponding column and weight. One may think of it as performing a random sampling across each channel since in each iteration from $i = 1$ to f_{l+1}, we perform multiplication with the $i - th$ column of H.

As for the implementation of GDC, we decided to implement the less complex version, as shown below, since this implementation reduces the runtime completely and is also the one that was originally implemented for testing the efficacy of GDC.

$$H^{(l+1)} = \sigma(\sum_{i=1}^{f_l} \mathfrak{N}(A \odot Z_i^{(l)}) H^{(l)}[:, i] W^{(l)}[i, :]) \tag{3.2}$$

Here, we compute the new feature matrix in one go instead of doing f_l iterations for all f_l columns.

3.3 Choice of Libraries and Frameworks

Below, we briefly overview the datasets and frameworks used and motivate the choice. Even though machine learning on graph-structured data is carried out

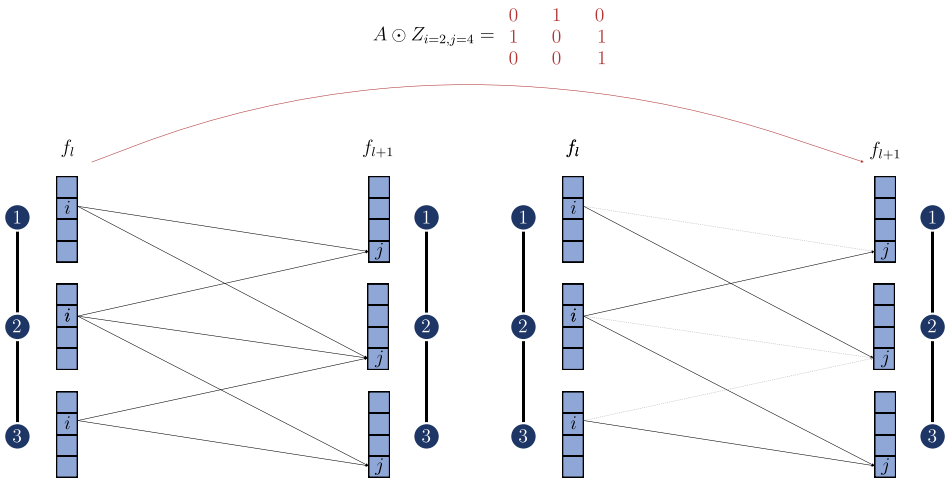

Fig. 3.2.: GDCNote: self connection are assumed

in many areas and has many interesting use cases ranging from social networks to molecular graphs, manifolds, and source code [Hu+20], there is no unified framework for working with graph-structured data. Furthermore, commonly used datasets and evaluation procedures suffer from multiple issues that negatively affect the quality of predictions and the reliability of evaluations of models. Machine learning algorithms rely heavily on data. For a GNN to be able to make accurate predictions, there is a need for a sufficient amount of properly prepared training data. Standardized splitting and evaluation methods are needed to compare different models against each other.

Today, most of the frequently used graph datasets are extremely small compared to graphs found in real applications. The same datasets, such as Cora, CiteSeer, and PubMed, are used repeatedly to train various models, leading to poor scalability in most cases. Small datasets also make it hard to rigorously evaluate data-hungry models, such as graph neural networks (GNNs). The performance of a GNN on these datasets is often unstable and nearly statistically identical due to the small number of samples the models are trained and evaluated on [KW17; Xu+19; Hu+20].

OGB offers a wide range of different-sized graph datasets from different domains for a variety of varying classification tasks and provides a unified pipeline for working with the datasets in ML tasks. The unified experimental protocol with standardized dataset splits, evaluation metrics, and cross-validation protocols makes it easy to compare performance reported across various studies [Hu+20].

Working with OGB consists of following steps:

1. OGB provides realistic, different-scaled graph benchmark datasets that cover different prediction tasks from diverse applications.

2. Dataset processing and splitting is fully automated. OGB data loaders automatically download and process graphs and further split the datasets in a standardized manner. This is compatible with multiple libraries and provides a library-agnostic option.

3. This step includes developing an ML model to train on the OGB datasets.

4. OGB evaluates the model in a dataset-dependent manner and outputs the model performance appropriate for the task at hand.

5. OGB provides public leaderboards to keep track of recent advances.

Fig. 3.3.: Overview of the standardized OGB pipeline adapted from [Hu+20]

For machine learning tasks, TensorFlow, a powerful and versatile open-source machine learning framework, was a fundamental tool for developing and training intricate machine learning models. TensorFlow's extensive set of libraries and tools simplifies the process of building, training, and deploying machine learning models, making it a preferred choice. The framework's adaptability to various tasks, ranging from image recognition to natural language processing, underscores its universal applicability, positioning it at the forefront of modern machine learning research. In addition to TensorFlow, this study leveraged NetworkX, a Python package designed to create, manipulate, and analyze complex networks. NetworkX was used as a tool for graph representation.

4 Evaluation

In this section, we delve into the critical evaluation of the machine learning experiments conducted as part of this research. The main postulated question of this study was to determine whether GDC is effective in solving the problem of overfitting and over-smoothing for graph-level prediction tasks, as there is already a wide range of conducted studies, which answer the question of various regularization techniques for node level prediction tasks. Other regularization techniques have also been evaluated for this type of task. The investigation encompassed classification and regression tasks, with a comprehensive analysis of two types of neural networks, GCN and GIN. The datasets of choice were all molecular datasets. In the evaluation, we will mainly focus on two manipulated parameters: the number of layers and the dropout rate, since the number of layers is important in concluding overfitting, especially the issue where additional layers do not make the network perform better. The dropout rate since this parameter indicates the efficacy of various types of dropouts and if higher rates have an impact at all.

4.1 Datasets and Metrics Overview

Before proceeding to the experimental findings, we present a quick overview of the datasets and metrics. We have used five datasets, all from the molecular realm. Molhiv and molpcba are small and medium-size classification datasets, respectively. The other three, OGB-molesol, -mollipo, and -molfreesolv are regression datasets. They contain 1128, 4200, and 642 molecular structure graphs, respectively. The regression task is to predict the solubility of a molecule in different substances. To evaluate the performance of molhiv, we used ROC-AUC for molpcba Average Precision (ap) was used, and we used Mean Absolute Error (MAE) for all the regression datasets.

4.2 Experimental Setup

This section outlines the experimental setup for training the GCN and the GIN network architectures. We did not have to deal with data preprocessing steps and data preparation techniques because we use the OGB datasets [Hu+20].

The training of the GNN models followed a systematic methodology to ensure the robustness and reliability of the results. The following key parameters were considered: The training process spanned $N_{epochs} = 200$ to ensure the model can learn the underlying patterns within the data. Training with each configuration was repeated three times to account for variability in the training process. Early stopping was used to prevent overfitting and enhance the generalization ability of the models. The training process monitored the validation loss, and the training was halted if the validation loss did not improve for a consecutive number of epochs. The 'patience' parameter was set to 50, indicating that the training process was terminated early if the validation loss did not decrease over 50 consecutive epochs. The selection of the best hyperparameter configuration was based on the validation loss. The configuration yielding the lowest validation loss was chosen as the optimal setting among the evaluated hyperparameter combinations. The Adam optimizer was employed during the training process.

4.2.1 Parameter Grid

In this study, a comprehensive exploration of the model's hyperparameters was conducted to optimize the performance of the graph neural network. The hyperparameter search space was defined through a parameter grid encompassing various configurations. It was designed to contain many possibilities, enabling a systematic investigation of the model's behavior under different settings.

This parameter grid contained several layers ranging from 1 to 6, representing the depth of the neural network architecture. The learning rate alternated between the three different values 0.0001, 0.001, and 0.01 for the optimization of the model. We chose the Adam optimizer exclusively for its robust performance in optimizing complex neural networks. As mentioned previously, we looked at four regularization techniques described in section 2.2.4, DO, NS, DE, GDC, and also looked at the performance when no regularization was applied. The drop probability for each

performed regularization was set to 0.3, 0.5, or 0.7. We also considered three activation functions: 'relu', 'sigmoid', and 'tanh'. The number of units in the hidden layer units, determining the neural network's capacity, was alternated between 32 and 64. All this was done for two different network architectures, GCN and GIN. All possible combinations of these parameters were generated to comprehensively explore the hyperparameter space, resulting in a list denoted as `param_combos`. Each configuration within `param_combos` represented a unique set of hyperparameters for the graph neural network. By exhaustively evaluating these combinations, this study aimed to identify the most suitable hyperparameter configuration, providing valuable insights into the optimal setup for graph-based machine learning tasks.

4.2.2 Finding the Best Set of Hyperparameters

For hyperparameter optimization, we use grid search grid search (GS) [Lor+17; YS20; ZH21]. GS is a model-free method of automated hyperparameter selection, which systematically explores the configuration space by performing an exhaustive search. Grid search has two major drawbacks:

1. Poor scalability for large configuration spaces due to its exponential complexity in the number of hyperparameters and corresponding values. Assuming that there are k parameters, and each of them has n distinct values, its computational complexity is $O(n^k)$.

2. Lack of consideration of the hierarchical hyperparameter structure leads to many redundant configurations.

Despite its two major drawbacks, GS is well-suitable for small search spaces and can easily be implemented and parallelized.

4.3 Experimental Results

As seen in table 4.1, the best results are achieved using no regularization techniques for graph-level prediction tasks on both types of networks GCN and GIN. This holds for both datasets- classification, and regression- indicating that any regularization type is unsuitable for both graph-level prediction tasks independent of the network

Tab. 4.1.: Experimental results for graph-level prediction tasks. With ROC-AUC metric for OGB-molhiv, AP for -molpcba and MSE for the three remaining regression datasets.

		OGB-molhiv	-molpcba	-molesol	-molfreesolv	-mollipo
GCN	**None**	0.51 ± 0.12	0.11 ± 0.00	1.67 ± 0.20	10.01 ± 1.71	0.69 ± 0.02
	DropOut	0.47 ± 0.03	0.06 ± 0.00	3.97 ± 0.31	13.84 ± 0.77	1.11 ± 0.01
	NodeSampling	0.51 ± 0.03	0.07 ± 0.00	2.96 ± 0.12	12.54 ± 0.59	1.06 ± 0.02
	DropEdge	0.48 ± 0.10	0.10 ± 0.00	1.92 ± 0.07	8.79 ± 1.08	0.82 ± 0.05
	GDC	0.54 ± 0.03	0.08 ± 0.00	2.88 ± 0.13	13.29 ± 1.19	1.02 ± 0.03
GIN	**None**	0.70 ± 0.01	0.10 ± 0.02	1.74 ± 0.10	8.36 ± 0.70	0.75 ± 0.05
	DropOut	0.50 ± 0.03	0.07 ± 0.00	3.46 ± 0.24	20.32 ± 1.15	1.10 ± 0.02
	NodeSampling	0.55 ± 0.04	0.08 ± 0.00	3.02 ± 0.59	13.10 ± 1.80	0.94 ± 0.04
	DropEdge	0.52 ± 0.04	0.11 ± 0.00	2.16 ± 0.21	7.94 ± 0.17	0.78 ± 0.05
	GDC	0.52 ± 0.03	0.09 ± 0.00	2.54 ± 0.20	20.10 ± 3.50	1.06 ± 0.04

of choice. For both classification datasets, the variance is very small; the network performance is stable. Out of the three regression datasets, only the mollipo dataset has low variance in performance for both types of GNNs, and we have rather a high variance on the remaining datasets. The high variance of molesol and mollipo is an interesting trend, which would be nice to investigate further. Despite achieving the best result when using no regularization, we can point out a clear second-place winner among the different regularization methods on both networks and across all datasets. DE performs the second best in all cases, with the second place being GDC for classification tasks and NS for regression tasks, apart from one exception on the mollipo dataset where the second best performing regularization is GDC. However, as all the results are very close in range, we cannot point out any notable advantages of using GDC above other regularization methods, as their performance varies depending on the task and dataset. Despite the GIN network being more powerful than GCN and as powerful as the 1-dim. WL test, the performance on both networks is very similar, with only a significant difference in performance on the molhiv dataset.

4.4 Detailed Investigation of Change in Number of Layers and Probability

Since we could not detect any benefits of using regularization for graph-level prediction tasks, indicating that the potential reduction of over-smoothing shown by [Has+20] is not relevant for graph-level prediction tasks. Therefore, we focus on

overfitting and investigate how the model performance changes when we increase the number of layers. Also, we want to make sure that our findings are consistent with and can be attributed to the use of regularization, which is why we take a look at the change of performance with regards to drop probability.

4.4.1 Effect of the Number of Layers

This analysis aims to gain insights into the phenomenon of over-smoothing. To achieve this, we investigate the variations in performance corresponding to changes in the number of layers within the network architecture. Upon analyzing five distinct datasets, discernible patterns in performance relative to the number of network layers become evident. Except for the molhiv and molfreesolv datasets, a weak positive correlation between increased layers and improved performance is observable. This trend is consistent across both GCN and GIN models. Typically, optimal performance is achieved at the 5th or 6th layer. However, it is noteworthy that the overall differences in performance are marginal.

This observed trend persists across various regularization techniques, including scenarios without regularization. Interestingly, there is insufficient evidence for the hypothesis that regularization effectively mitigates over-smoothing for graph-level classification tasks on GCNs, specifically concerning these datasets. Additionally, substantial fluctuations are observed in the molhiv dataset, so no clear trends could be found.

In regression datasets, the molfreesolv dataset presents a unique case where optimal performance is attained with a single layer, contrary to the general trend where 5-6 layers yield high performance. Importantly, consistent trends emerge across all regularization techniques and without regularization, suggesting that regularization does not induce unexpected network behavior.

Regarding variance, apart from substantial variance and high fluctuations observed in the molhiv dataset, across both networks and all types of regularization, including scenarios without any regularization, we can observe somewhat clear trends most of the time. Overall, higher variance is evident in instances without regularization, whereas regularization techniques tend to stabilize performance, resulting in lower variance. This finding underscores the stabilizing influence of regularization techniques on model performance in graph-level classification tasks.

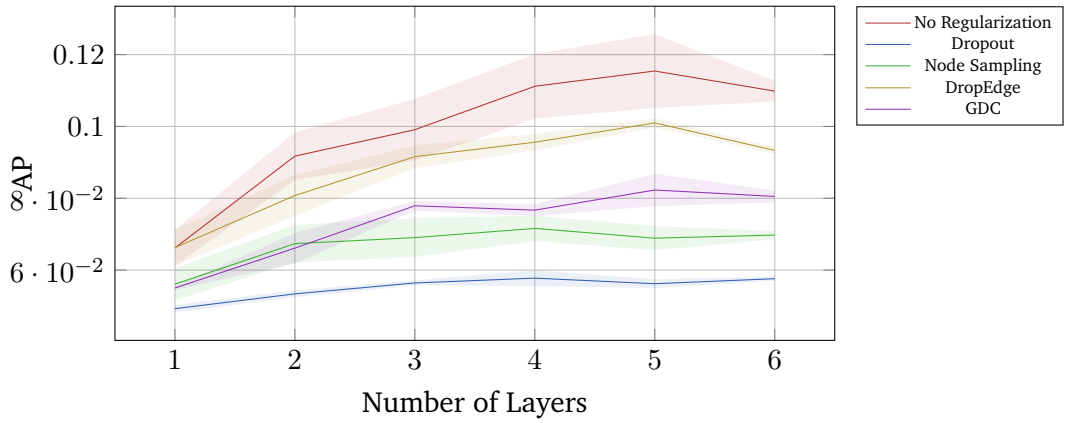

Fig. 4.1.: molpcba (GCN Model)

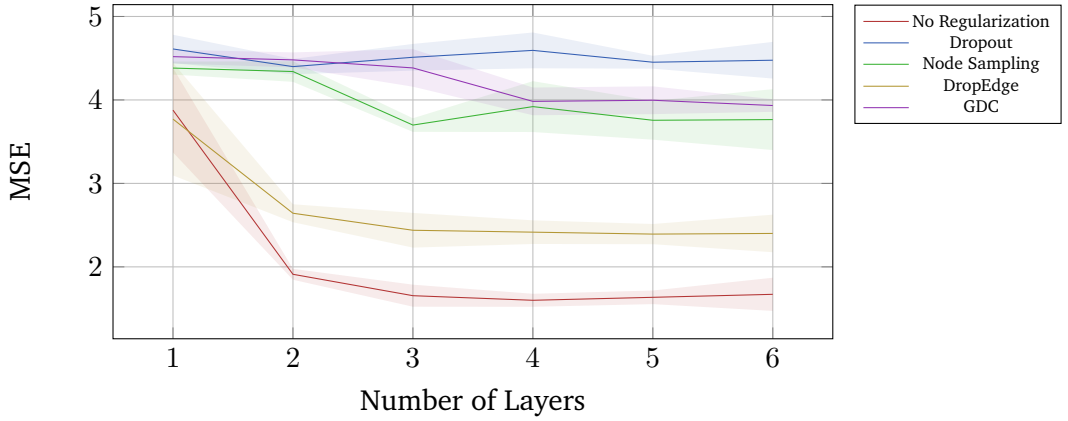

Fig. 4.2.: molesol (GCN Model)

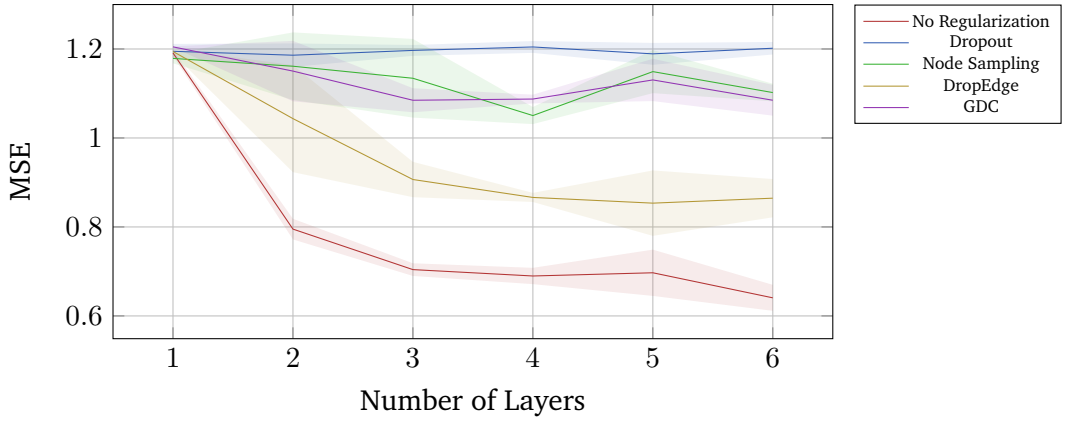

Fig. 4.3.: mollipo (GCN Model)

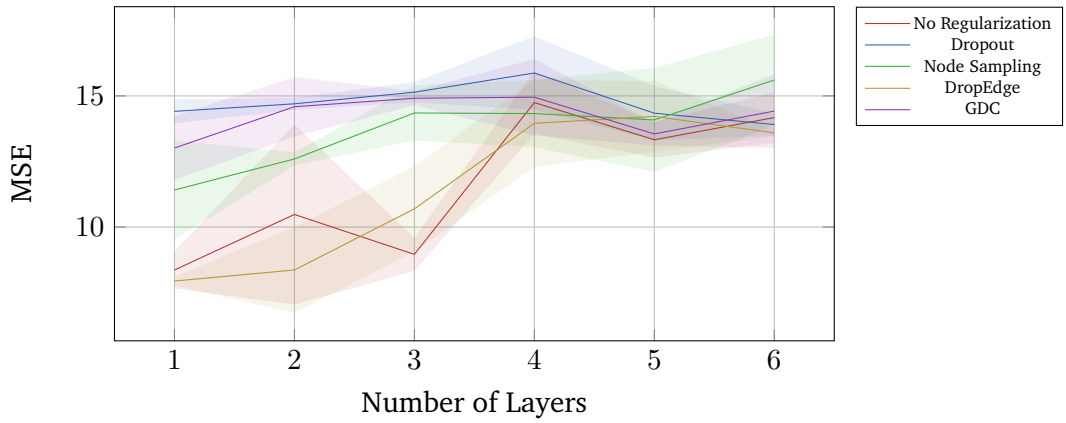

Fig. 4.4.: molfreesolv (GIN Model)

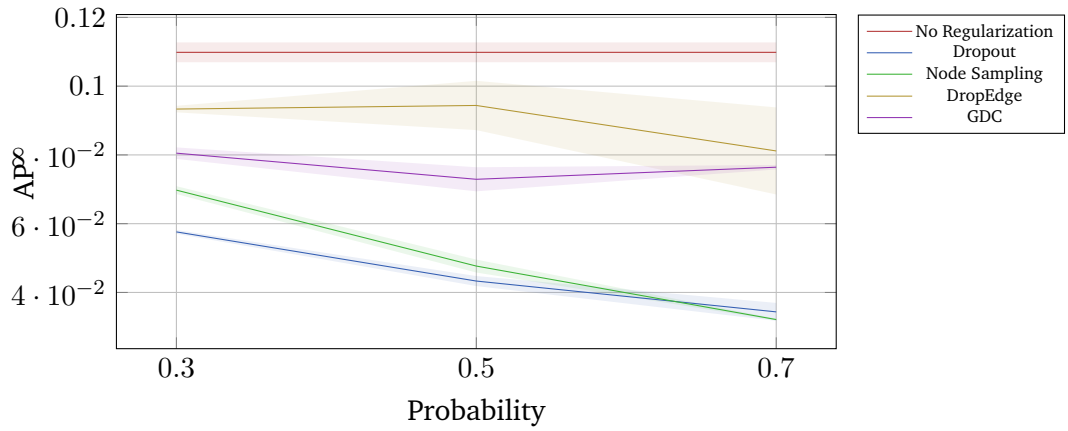

Fig. 4.5.: molpcba (GCN Model)

4.4.2 Effect of the Probability of Regularization

We also take a closer look at how a change in the drop probability affects the performance of the networks. Regarding this, we have found that the model performs worse on average as we approach higher probabilities, i.e., a lower dropout probability leads to better performance. This makes sense, as the best performance is achieved using no regularization. This trend can be observed in the regression datasets on the GCN and the GINnetwork. This trend also holds for the classification dataset molpcba on both networks. On the molhiv dataset, the dropout probability does not lead to any substantial differences in performance, which is why we do not show it.

Among the datasets, the classification dataset molhiv and the regression dataset molfreesoolv

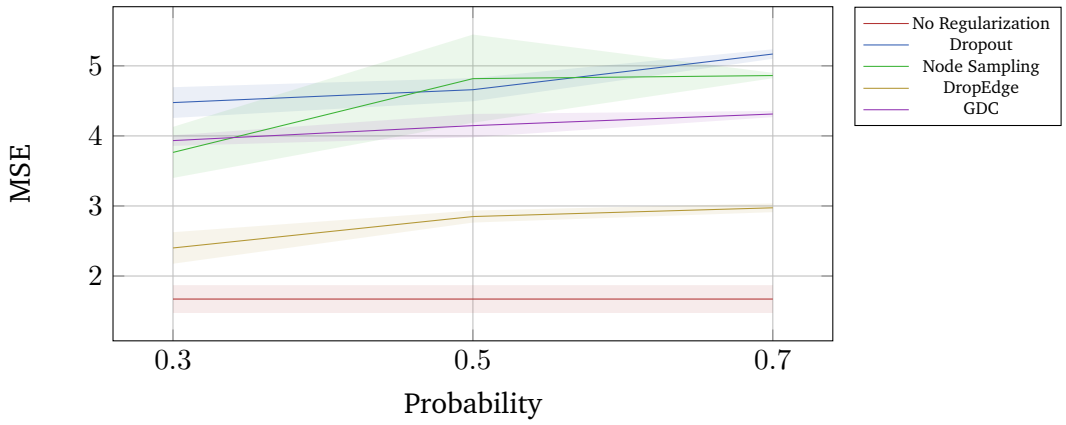

Fig. 4.6.: molesol (GCN Model)

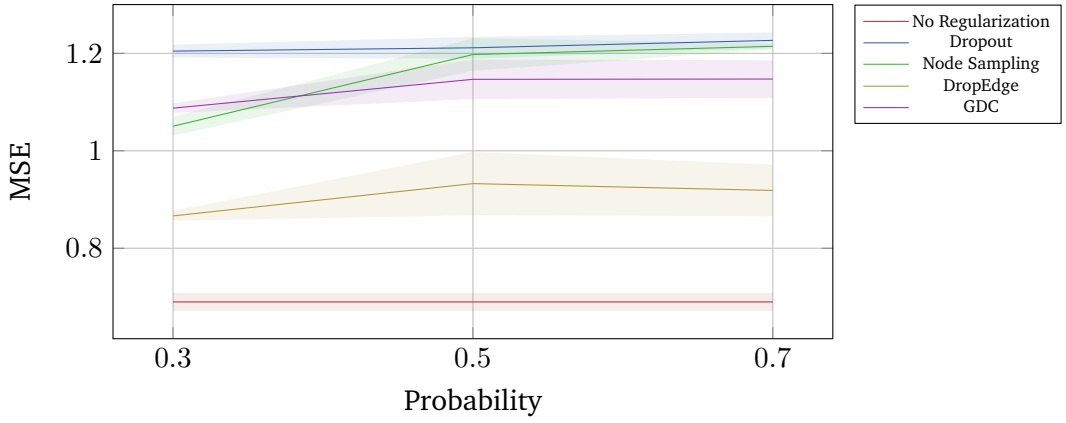

Fig. 4.7.: mollipo (GCN Model)

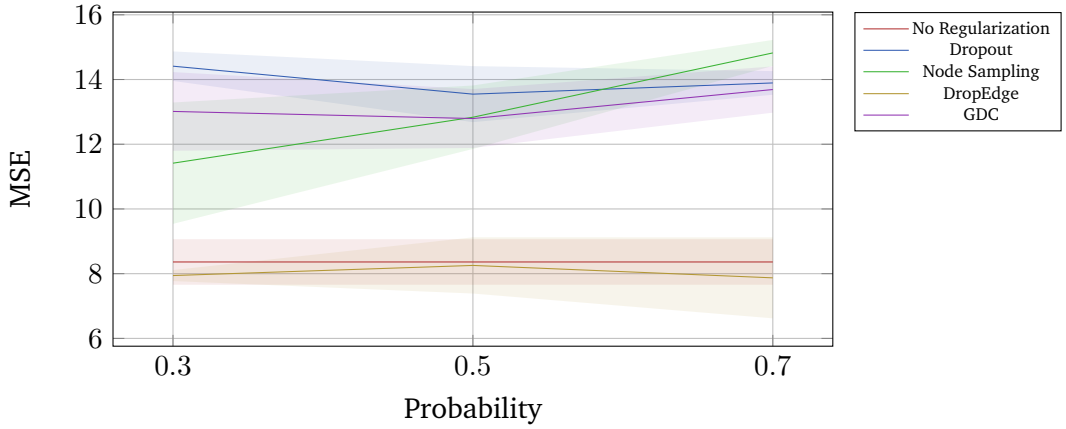

Fig. 4.8.: molfreesolv (GIN Model)

As mentioned earlier, the results are very close in range, and together with the limited number of datasets- all from the same molecular realm- we cannot draw any general conclusions. However, we can describe some emerging trends from our findings: The trends: Overall, regularization seems to smoothen the line, i.e., to reduce the differences in performance depending on the number of layers. Also, the performance variance is much higher when we use no regularization.

5 Conclusion

5.1 Review

Our study examined the impact of various regularization techniques, particularly GDC, on graph-level prediction tasks across two different network types. Specifically, we postulated the following three research questions, which we will now answer.

RQ1: Is regularization, specifically GDC, effective in solving the problem of overfitting and over-smoothing for graph-level prediction tasks? Since both our networks had no problem with overfitting, even if no regularization was used, we cannot say anything about the effectiveness of regularization in mitigating this problem. We can only say that the model is not negatively affected by regularization in terms of over-smoothing.

RQ2: Is there a difference in performance between GCN and GIN architectures regarding performance with regularization techniques? We found no difference in performance between both network architectures GCN and GIN. Both networks perform best when we use no regularization at all, and on both networks, DE is the second-best performance most of the time. This holds for both classification and regression datasets.

RQ3: Are there similarities and differences between different regularization techniques in terms of performance? We tried to verify or refute whether any of the four regularization techniques DO, NS, DE, and GDC is more effective. In terms of performance, the different regularization techniques are very close. DE performs best among the different datasets and GDC is second-place, hinting at the close relatedness between those two techniques, as already has been described by [Has+20].

5.2 Future Work

Our investigation observed that regularization might not offer significant advantages for graph-level prediction tasks, opening avenues for further exploration, which leads to intriguing questions that warrant deeper investigation. While our study focused on two widely employed GNN architectures, GCN and GIN, it is crucial to acknowledge the existence of many other interesting types of GNNs. Furthermore, our experiments were conducted on a limited number of datasets; the generalizability of our findings to other datasets remains an open question, and we encourage researchers to extend our work to explore potential variations in results across diverse datasets.

In our study, we stumbled upon an interesting metric MAD, which measures the similarity between nodes and allows us to gain deeper insights into how and when over-smoothing occurs. Regrettably, due to the expansive nature of this research, we could not comprehensively evaluate MAD within the scope of this study. We advocate for further research to assess the efficacy of MAD in graph-level prediction tasks. Additionally, our exploration hints at alternative regularization techniques that may yield promising results. `Noisy Nodes`, a particularly intriguing regularization approach, in which the input graph is corrupted with noise and a noise-correcting node-level loss is added [God+22].

In conclusion, while our study sheds light on the limited efficacy of traditional regularization methods, it also lays the groundwork for future research directions. We hope that our findings stimulate further inquiry into the nuances of graph-level prediction tasks, focusing on evaluating MAD and exploring alternative regularization techniques, such as `Noisy Nodes`, to advance the field and deepen our understanding of graph-based machine learning models.

Appendix A

Bibliography

[BK79] László Babai and Ludek Kucera. “Canonical Labelling of Graphs in Linear Average Time”. In: *20th Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 29-31 October 1979*. IEEE Computer Society, 1979, pp. 39–46 (cit. on p. 9).

[Che+20] Deli Chen, Yankai Lin, Wei Li, et al. “Measuring and Relieving the Over-Smoothing Problem for Graph Neural Networks from the Topological View”. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 3438–3445 (cit. on pp. 2, 14, 19).

[CMX18] Jie Chen, Tengfei Ma, and Cao Xiao. “FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018 (cit. on p. 16).

[DMH20] Clemens Damke, Vitalik Melnikov, and Eyke Hüllermeier. “A Novel Higher-order Weisfeiler-Lehman Graph Convolution”. In: *Proceedings of The 12th Asian Conference on Machine Learning, ACML 2020, 18-20 November 2020, Bangkok, Thailand*. Ed. by Sinno Jialin Pan and Masashi Sugiyama. Vol. 129. Proceedings of Machine Learning Research. PMLR, 2020, pp. 49–64 (cit. on pp. 7, 24).

[God+22] Jonathan Godwin, Michael Schaarschmidt, Alexander L. Gaunt, et al. “Simple GNN Regularisation for 3D Molecular Property Prediction and Beyond”. In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022 (cit. on p. 42).

[HYL17] William L. Hamilton, Rex Ying, and Jure Leskovec. “Representation Learning on Graphs: Methods and Applications”. In: *IEEE Data Eng. Bull.* 40.3 (2017), pp. 52–74 (cit. on p. 6).

[Has+20] Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, et al. “Bayesian Graph Neural Networks with Adaptive Connection Sampling”. In: *CoRR* abs/2006.04064 (2020). arXiv: 2006.04064 (cit. on pp. 1, 14, 17, 19, 23, 27, 34, 41).

[He+07] Bingsheng He, Naga K. Govindaraju, Qiong Luo, and Burton Smith. “Efficient gather and scatter operations on graphics processors”. In: *Proceedings of the 2007 ACM/IEEE conference on Supercomputing - SC ’07*. ACM Press, 2007 (cit. on p. 24).

[Hu+20] Weihua Hu, Matthias Fey, Marinka Zitnik, et al. “Open Graph Benchmark: Datasets for Machine Learning on Graphs”. In: *CoRR* abs/2005.00687 (2020). arXiv: 2005.00687 (cit. on pp. 23, 29, 30, 32).

[HV22] Ningyuan Huang and Soledad Villar. “A Short Tutorial on The Weisfeiler-Lehman Test And Its Variants”. In: *CoRR* abs/2201.07083 (2022). arXiv: 2201.07083 (cit. on p. 7).

[KW17] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017 (cit. on pp. 10, 11, 16, 23, 29).

[KGC17] Jan Kukacka, Vladimir Golkov, and Daniel Cremers. “Regularization for Deep Learning: A Taxonomy”. In: *CoRR* abs/1710.10686 (2017). arXiv: 1710.10686 (cit. on p. 14).

[LYJ22] Meng Liu, Haiyang Yu, and Shuiwang Ji. “Your Neighbors Are Communicating: Towards Powerful and Scalable Graph Neural Networks”. In: *CoRR* abs/2206.02059 (2022). arXiv: 2206.02059 (cit. on p. 10).

[Lor+17] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sánchez Ramos, and José Ranilla Pastor. “Particle swarm optimization for hyper-parameter selection in deep neural networks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017*. Ed. by Peter A. N. Bosman. ACM, 2017, pp. 481–488 (cit. on p. 33).

[Ron+20] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. “DropEdge: Towards Deep Graph Convolutional Networks on Node Classification”. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020 (cit. on pp. 16, 19).

[Sri+14] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1929–1958 (cit. on p. 15).

[WL68] Boris Weisfeiler and Andrei A. Lehman. “A reduction of a graph to a canonical form and an algebra arising during this reduction”. In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968), pp. 12–16 (cit. on p. 7).

[Xu+19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. “How Powerful are Graph Neural Networks?” In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019 (cit. on pp. 6, 11, 12, 23, 29).

[YS20] Li Yang and Abdallah Shami. “On hyperparameter optimization of machine learning algorithms: Theory and practice”. In: *Neurocomputing* 415 (2020), pp. 295–316 (cit. on p. 33).

[Zha+19] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. “Graph Convolutional Networks: A Comprehensive Revew”. In: *Computational Social Networks* (2019) (cit. on pp. 5, 6).

[Zho+20a] Jie Zhou, Ganqu Cui, Shengding Hu, et al. “Graph neural networks: A review of methods and applications”. In: *AI Open* 1 (2020), pp. 57–81 (cit. on p. 6).

[Zho+20b] Kaixiong Zhou, Xiao Huang, Yuening Li, et al. “Towards Deeper Graph Neural Networks with Differentiable Group Normalization”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc’Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020 (cit. on p. 14).

[ZH21] Marc-André Zöller and Marco F. Huber. “Benchmark and Survey of Automated Machine Learning Frameworks”. In: *J. Artif. Intell. Res.* 70 (2021), pp. 409–472 (cit. on p. 33).

List of Figures

List of Tables

Colophon

This thesis was typeset with LaTeX 2_ε. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Declaration

Ich, Olga Yakobson (Matrikel-Nr. 11591478), versichere, dass ich die Masterarbeit mit dem Thema SerialExperimentsOlga selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen der Arbeit, die ich anderen Werken dem Wortlaut oder dem Sinn nach entnommen habe, wurden in jedem Fall unter Angabe der Quellen der Entlehnung kenntlich gemacht. Das Gleiche gilt auch für Tabellen, Skizzen, Zeichnungen, bildliche Darstellungen usw. Die Bachelorarbeit habe ich nicht, auch nicht auszugsweise, für eine andere abgeschlossene Prüfung angefertigt. Auf § 63 Abs. 5 HZG wird hingewiesen.

München, 26. November 2023

Olga Yakobson