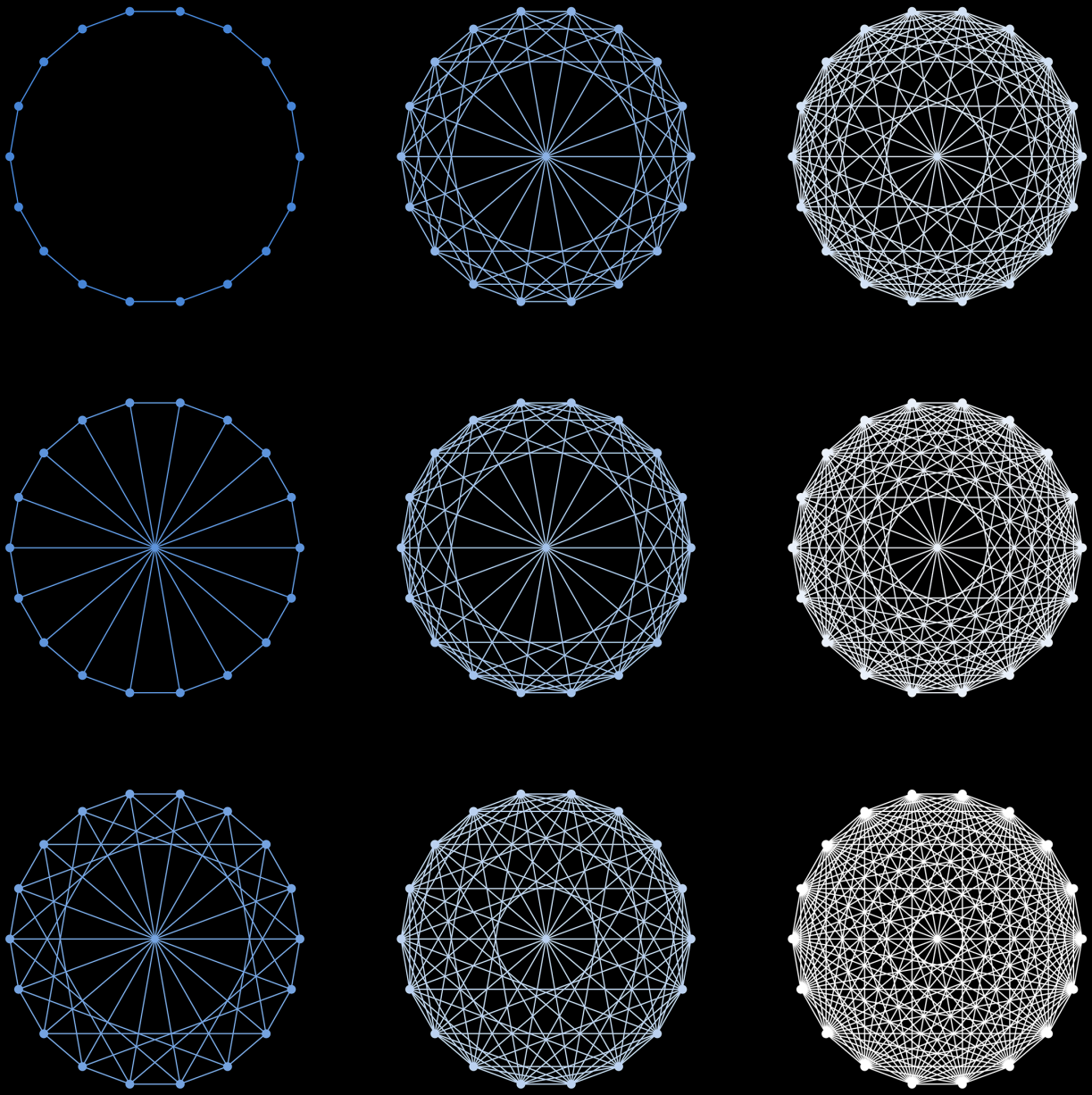
An Experimental Assessment of the Stability of Graph Contrastive Learning

Siert Sebus



Delft University of Technology

TU Delft

An Experimental Assessment of the Stability of Graph Contrastive Learning

by

Siert Sebus

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on the Thursday of January 11th, 2024 at 15:00.

Student number:	4710304	
Thesis committee:	Dr. Hadi Jamali-Rad,	TU Delft and Shell, Daily supervisor
	Dr. Elvin Isufi,	TU Delft, Daily co-supervisor
	Dr. Jan van Gemert,	TU Delft, Advisor
	Dr. Artur M. Schweidtmann,	TU Delft, External committee member
Project duration:	December, 2022 – January, 2024	
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft	

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This report functions as a record of the work I have done for my Master Thesis project. That is, it describes the research questions I asked, the experiments I did to answer those questions, and the literature I consulted along the way. Additionally, it contains a background chapter with the goal of providing the reader with the technical background needed to understand my work. That said, the reader is assumed to have some prior knowledge on deep learning and basic linear algebra.

This project has been one of the hardest things I have done in my life and I think it is safe to say that this was not solely due to challenges in understanding the technical subject matter. My supervisor Hadi and co-supervisor Elvin are not easily pleased and, as a result, collaborating with them has, at times, been difficult. Nonetheless, I want to sincerely thank both of them for sticking with me till the end and supporting me in times that my own hope was dwindling. They both have always been quick to answer my questions and always showed genuine interest in my work during our meetings. Additionally, I want to thank Elvin for taking the wheel at the halfway point and turning my then vague project into something that I could actually graduate on.

Next, I want to thank the remaining members of my Thesis committee, Jan van Gemert and Artur Schweidtmann, for their involvement in assessing the project.

I also want to thank some of the members of the Multimedia Processing research group, Bishwadeep, Jorge, Mohammad, Maosheng, and Tianqi, for the warm welcome they gave me at their floor of the CS building and for the fun lunch talks we have had.

Furthermore, I want to thank my family and my closest friends for their continuous love and support, forming the backbone to any achievement that I make, including this one.

Lastly, I want to give special thanks to the members of “Thesis Town”, Misha and Koen in particular, for embarking on this journey with me. I know for sure that if I had done this entire thesis alone from my tiny room, I would not have been nearly as productive and, frankly, I would have grown depressed. Thanks for making me look forward to come to campus every day!

Siert Sebus
Delft, January 2024

Summary

Deep Neural Networks (DNNs) have become a widely popular machine learning architecture thanks to their ability to learn complex behaviors from data. Standard learning strategies for DNNs however rely on the availability of large, labeled datasets. Self-Supervised Learning (SSL) is a style of learning that allows models to also use *unlabeled* data for training, which is typically much more abundant.

SSL is being applied many different data domains such as images and natural language. One such a domain is the domain of graph data. A graph is a data structure describing a network of nodes connected by edges. Graphs are a natural way of presenting many forms of data such as molecules, social networks, and 3D meshes.

The style of SSL that has found the most success on graphs is Contrastive Learning (CL). In CL, an encoder is trained to produce semantically rich representations from unlabeled input data by smartly separating task-relevant information in the input from task-irrelevant information. The encoder backbone most commonly used for Graph Contrastive Learning (GCL) is the Graph Convolutional Neural Network (GCNN).

While GCNNs are the state of the art on many graph data tasks, they suffer from *underfitting* when made too deep. This is especially a problem for GCL as it prevents encoder complexity to scale with the large availability of unlabeled data.

In this thesis, we investigate this underfitting behavior through the lens of GCNN stability. Stability refers to a model's ability to continue producing consistent outputs, even when its inputs are perturbed slightly. Theoretical work has shown that stability guarantees for GCNNs weaken when their complexity is increased. We confirm experimentally that, in many cases, GCNNs indeed grow less stable when made more complex. This a relevant finding given that learning stable representations is a prerequisite to CL. Additionally, we show in our experiments that, even when trained using CL, stability discrepancies between different GCNN architectures do not disappear. This, in turn, suggests that GCNN architectures with poorer stability may also produce poorer representations. We confirm experimentally that, on at least one dataset, poor stability as a result of architectural complexity can indeed be correlated to a degradation in representation quality.

With this result we provide an additional explanation as to why deeper GCNNs are often found to perform worse in GCL settings. These insights can, in turn, motivate the design of model architectures for GCL that do not suffer from this trade-off between complexity and representation quality.

Contents

1

Introduction

Deep Neural Networks (DNNs) have gained fame for pushing the boundaries of what problems machines can solve. Behaviors previously reserved only to humans, such as object recognition in images and video [3, 42], image generation [7, 38], speech synthesis [35], natural language understanding [22, 51], and strategic decision making in games [12, 57], have all successfully been adopted by DNN models, sometimes even surpassing human performance. The data-hungry nature of DNNs, however, makes it difficult to apply them to domains where gathering labeled data is expensive, unethical or prohibited for other reasons. To overcome this, the Self-Supervised Learning (SSL) paradigm introduces a style of learning that allows the use of *unlabeled* data during training, which is typically much more abundant than labeled data. This makes SSL responsible for many recent successes of DNNs as it allows for training with datasets orders of magnitude larger than was previously possible [31].

A notable style of self-supervised learning is Contrastive Learning (CL). Contrastive learning is done by training an encoder to compress unlabeled input data into semantically rich representations. Specifically, CL uses data augmentations to perturb task-irrelevant information in the input to subsequently train an encoder that is invariant to those augmentations. This allows the encoder to separate task-relevant information in the input from task-irrelevant information.

One prominent data domain to which CL is applied is the domain of graph data. A graph is a data structure that can describe network-like data such as social networks, recommendation networks, traffic networks, point clouds, 3D meshes, molecules, proteins, and knowledge graphs. The links inside such networks are referred to as *edges* and the entities being connected by those links are referred to as *nodes*. For example, when encoding a molecule using a graph, the atoms are the graph's nodes and the atom bonds are the edges.

Applying contrastive learning to the graph domain (see Figure 1.1), referred to as Graph Contrastive Learning (GCL), is not trivial. Strategies that work well for visual data or natural language often cannot directly be applied to graphs. The main reason for this is the free-form nature of graphs: they can describe networks of varying size and with varying connectivity. Additionally, the nodes and edges inside a graph do not have an ordering. As a result, there are multiple equally valid ways of encoding the same graph in computer memory.

The Graph Neural Network (GNN) has achieved state-of-the-art performance on many graph data tasks [16, 71]. GNNs iteratively update node information on the graph by aggregating over the local neighborhood of each node. Such an iterative update is referred to as a layer of the GNN. In particular, Graph Convolutional Neural Networks (GCNNs) are a popular choice of GNN. As the name suggests, this style of GNN relies on a convolution mechanism for updating node information.

GCNNs are often used as the encoder backbone in GCL [32, 61]. However, despite their popularity, we hypothesize that they are a suboptimal choice for GCL. Our motivation for this comes from a collection of work on the stability properties of GCNNs [9, 11, 23, 26], some of which suggest that overly complex GCNNs suffer from instability. Here, "stability" refers to a GCNN's ability to produce consistent outputs when small perturbations are applied to its input. This is relevant to GCL given that, when doing CL, one aims to train an encoder that produces augmentation-invariant representations, i.e. representations that are *stable* with respect to the augmentations [4, 40, 50, 59]. The respective works, [9] and [11], derive upper bounds on the instability of GCNNs. These bounds grow looser as GCNN complexity

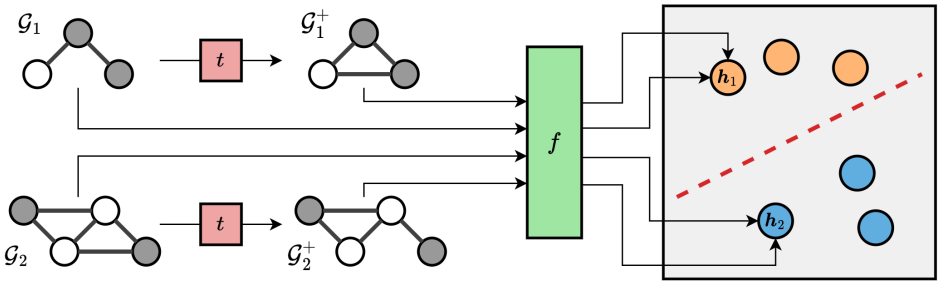


Figure 1.1: Graph Contrastive Learning (GCL) is a style of learning where an encoder f is trained to compress raw graph data (left) into useful representations (right). This is achieved by (1) defining a data augmentation function t that perturbs irrelevant information in the input graphs and (2) training the encoder f to become invariant to such augmentation. This training scheme encourages the encoder f to omit irrelevant information from its encodings.

increases. Thus, we hypothesize that more complex GCNNs have poorer stability. If this hypothesis holds, then overly deep GCNNs can't acquire invariance to the augmentation function, which can in turn harm the quality of learned representations. This would put a hard limit on the complexity of behaviors that GCNNs can learn. Such a limit is especially problematic in the GCL setting, where training is done on abundant unlabeled data and complex models are needed to take full advantage of that abundance. Hence, the main goal of this thesis is answering the following question:

> Do overly complex GCNNs have poor stability properties and, if so, does this explain the poor performance of complex GCNNs in GCL settings?

For clarity, we split this question up in four subquestions. Firstly, To verify that the link between GCNN stability and graph contrastive learning, we formulate the following research question:

RQ1. When a GCNN is pre-trained in a contrastive manner with increasing amounts of augmentation, does its stability to that augmentation improve?

We show that training using GCL can, in certain cases, indeed be linked to an increase in GCNN stability towards the respective augmentations. We will refer to this type of stability that a GCNN acquired through training as *trained stability*.

To verify the theoretically motivated intuition that overly complex are less stable, we formulate the second research question:

RQ2. When varying a GCNN's architecture, does its stability change in accordance to what theoretical stability bounds suggest?

We show that the stability of untrained GCNNs is heavily dependent on GCNN architecture and that, most of the time, more complex GCNNs are indeed less stable than their simpler counterparts. We will refer to this type of GCNN stability that is present in any GCNN, regardless of how it is trained, as *untrained stability*.

Given the two findings, (1) that GCNNs can acquire trained stability through GCL and (2) that GCNNs possess an untrained stability that becomes weaker as their architectures grow more complex, the natural next question is: how do these two types of stability interact? Can one use contrastive training to overcome instability as induced by an overly complex architecture? Or is there something about suboptimal GCNN architectures that prevents them from achieving good stability, even after training? This motivates the third research question:

RQ3. How does a GCNN's *untrained stability* as induced by its specific architecture interact with its *trained stability*, acquired through contrastive pre-training?

We find that improvements in stability through GCL on overly complex GCNNs can, in fact, not fully make up for losses in stability induced by GCNN architecture. An untrained GCNN that is less stable than another untrained GCNN due to a difference in architecture remains proportionally less stable, even after contrastive training. Complex GCNNs struggle to learn invariance to data augmentation and, thus, are prone to underfit on the pre-training task in a GCL setting.

To assess if such underfitting impacts the quality of learned representations, we ask a fourth and final research question:

RQ4. After being pre-trained contrastively using augmentation, is the downstream performance of GCNNs correlated to their stability to that augmentation?

We find at least for one dataset that poorer GCNN stability is indeed tied to worse downstream performance, i.e. degraded representation quality. This is the dataset for which we could confirm that the used data augmentation is appropriate for the dataset. For the three other datasets, we hypothesize that the used data augmentation was not appropriate and, thus, an inability to learn invariance to such augmentation does not affect representation quality negatively. With this, we answer our main question and provide a new explanation for the poor performance of complex GCNN architectures in GCL.

The remainder of this thesis is structured as follows. Chapter 2 gives the reader the necessary background to understand later sections and introduces the mathematical notation. Chapter 3 provides an overview of work done in Graph Contrastive Learning and stability theory as well as a discussion on challenges in GCL. In Chapter 4 we describe and motivate our hypotheses and we outline our experimental method for answering the research questions. The results of the experiments are given in Chapter 5. Lastly, Chapter 6 concludes the thesis by answering the research questions and by providing suggestions for future work.

2 Background

This chapter contains preliminaries to the thesis and introduces mathematical notation. The following topics are discussed: graphs (Section 2.1), graph convolutional neural networks (Section 2.2), stability properties of this style of graph neural network (Section 2.3), self-supervised learning (Section 2.4), contrastive learning (Section 2.5), and, lastly, contrastive learning on graph data specifically (Section 3.2). The reader is expected to have some prior knowledge of deep learning and machine learning.

2.1. Graphs

A **graph** (see Figure 2.1) is a data structure representing a network of **nodes** connected by **edges**. Graphs are useful for to their ability to describe many different types of network-like data. Examples of such data are molecules, proteins, citation networks, social networks, traffic networks, sensor networks, knowledge graphs and communication networks.

A graph can either be **directed** or **undirected**. In a directed graph, all edges have a direction, i.e. an edge going from node i to node j is strictly different from an edge going from j to i. In an undirected graph there is no such distinction. Directed graphs can be useful to describe asymmetric relationships between nodes, such as one paper referencing another in a citation network. Additionally, a graph can either be **weighted** or **unweighted**. Weighted graphs have weight values defined over the edges, describing the strength of any particular connection between two nodes.

Formally, a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$ is defined by a set of nodes $\mathcal{V}$ with $N = |\mathcal{V}|$, a set of edges $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$, and a weight function $w : \mathcal{E} \to \mathbb{R}$. Alternatively, a graph can be described using a weighted adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ defined as

$$[\boldsymbol{A}]_{ij} = \begin{cases} w(e_{ij}) & \text{if } e_{ij} \in \mathcal{E} \\ 0 & \text{if } e_{ij} \notin \mathcal{E} \end{cases}.$$

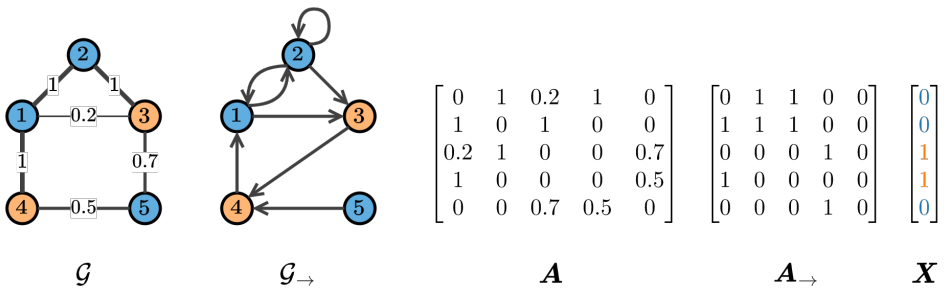


Figure 2.1: Depicted are examples of a weighted, undirected graph $\mathcal{G}$ and an unweighted, directed graph $\mathcal{G}_{\to}$. Their respective adjacency matrices $\boldsymbol{A}$ and $\boldsymbol{A}_{\to}$ are also given. A node's number in the diagram corresponds to its row and column indices in the matrices. Additionally, the nodes in these graphs have a one-dimensional, binary feature, portrayed as a node being colored either blue or orange. These node features are formally described by the given node feature matrix $\boldsymbol{X} \in \{0, 1\}^{N \times 1}$.

For undirected graphs, the adjacency matrix $\boldsymbol{A}$ is always symmetric. Here, each undirected edge is encoded as a pair of directed edges going opposite directions.

Additionally, it is useful for many applications to associate nodes with node-specific information. E.g. in a molecule graph it could be relevant to encode for each node what type of atom it represents. Formally, such node information is encoded as a feature vector $\boldsymbol{x}_i \in \mathbb{R}^F$ for the every node i in a graph. Here F is the dimensionality of the encoded information. The node features of all nodes in a graph can be combined in a feature matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\intercal \in \mathbb{R}^{N \times F}$. In signal processing literature, the node features associated with a graph are sometimes also referred to as the *signal* over that graph.

Besides node features, it can sometimes also be useful to encode information over the edges using edge features. Edge features are however not considered in this thesis.

2.2. Graph Convolutional Neural Networks

This section covers Graph Convolutional Neural Networks, how they build on top of Graph Convolutional Filters, and how both of these architectures relate to the graph spectrum. Additionally it covers two commonly used GCNN architectures, GCN and GIN, and gives a general overview of what kinds of machine learning tasks exist on graph data.

2.2.1. Graph Convolutional Filters

The irregular structures and varying sizes of graphs make it tricky to find appropriate, trainable architectures for processing graph data. Simple models such as Multi-Layer Perceptrons (MLPs) are suboptimal because they cannot capture graph structure without imposing an ordering on the nodes in the graph. Additionally, an MLP that directly takes as its input the full node features $\boldsymbol{X}$ or adjacency matrix $\boldsymbol{A}$ is inflexible to varying graph sizes as changing the graph size would change the input dimensionality.

Hence, there exists a special class of models in machine learning designed to effectively process graph data. A simple example of such a model is the **Graph Convolutional Filter (GCF)**, which elegantly solves the problems discussed above. GCFs make effective use of the graph structure, are flexible to varying graph size, and are invariant to node ordering, meaning that they provide consistent outputs for isomorphic graphs.

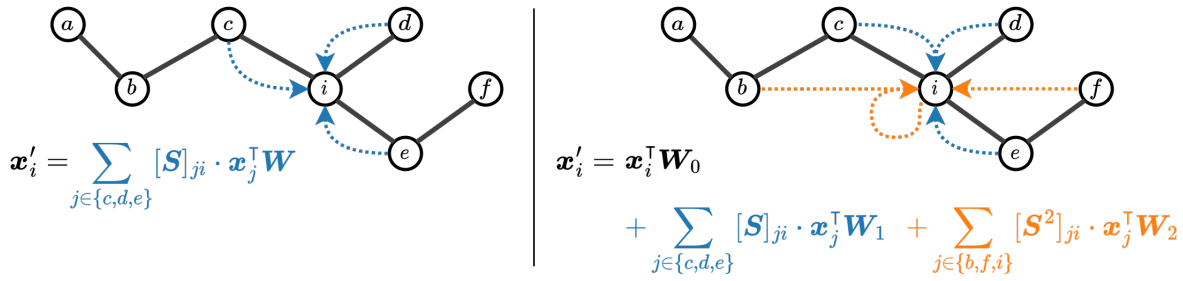


Figure 2.2: Graphical depictions of Graph Convolutional Filters (GCFs). On the left, a basic GCF (2.2) is portrayed. Here, a graph convolution is performed over a node i, taking its direct neighbors into account. On the right, a higher-order GCF (2.3) is given with $K = 2$. This filter additionally also sums over its 2-hop neighbors, i.e. neighbors that are two edges away. Note that node i is included as a 2-hop neighbor since it can be reached with a forward and a backward hop. Node i is also considered to be its own 0-hop neighbor, corresponding to $k = 0$ in (2.3). For both GCFs, node a lies outside the perceptive field of node i.

GCFs use convolutions to process information on a graph. This means that the new representations for any particular node are computed by first applying a linear transformation on the features of their neighbors and then taking the sum over those transformed representations (see Figure 2.2). Formally, a multi-dimensional **Graph Convolutional Filter (GCF)** is defined as

$$\boldsymbol{x}_i' = h(\boldsymbol{X}, \boldsymbol{S}, i) = \sum_{j \in \mathcal{N}_i} [\boldsymbol{S}]_{ji} \cdot \boldsymbol{x}_j^\intercal \boldsymbol{W}, \tag{2.1}$$

where $h(\boldsymbol{X}, \boldsymbol{S}, i) \in \mathbb{R}^{F'}$ is the filter function applied to node i, $\boldsymbol{x}_j$ is the feature vector of node j, $\mathcal{N}_i$ denotes the set of direct neighbors of i, $\boldsymbol{W} \in \mathbb{R}^{F \times F'}$ is a parameterized linear transformation, and

$\boldsymbol{S} \in \mathbb{R}^{N \times N}$ is the **Graph Shift Operator (GSO)**. A GSO is a structure-dependent transformation of the graph's node features, allowing nodes to exchange information with their neighbors. A valid GSO is any square matrix $\boldsymbol{S} \in \mathbb{R}^{N \times N}$ for which holds that $[\boldsymbol{S}]_{ji} = 0$ whenever $i \neq j$ and $(i, j) \notin \mathcal{E}$ [21]. An example of a valid GSO is the adjacency matrix $\boldsymbol{A}$ of a graph. The definition of a GCF (2.1) can also compactly written as:

$$h(\boldsymbol{X}, \boldsymbol{S}) = \boldsymbol{S}\boldsymbol{X}\boldsymbol{W}, \tag{2.2}$$

where $h(\boldsymbol{X}, \boldsymbol{S}) \in \mathbb{R}^{N \times F'}$ is the filter function applied to the entirety of the graph. Performing a matrix multiplication between $\boldsymbol{S}$ and the transformed features has the result of taking a weighted sum over a node's neighbor representations for each node.

Graph filters can be made more expressive by combining higher powers of the GSO $\boldsymbol{S}$ into a single filter. Applying the kth power of a GSO $\boldsymbol{S}$ to a graph signal, allows a node to attend to its k-hop neighborhood as opposed to just its direct neighbors (see Figure 2.2). Hence, combining different powers of $\boldsymbol{S}$ in a single filter greatly increases its expressiveness [21]. A higher order GCF is defined as:

$$h(\boldsymbol{X}, \boldsymbol{S}) = \sum_{k=0}^{K} \boldsymbol{S}^k \boldsymbol{X} \boldsymbol{W}_k. \tag{2.3}$$

Here, K is the maximally considered power, also referred to as the "order" of the graph filter, and $\boldsymbol{W}_k \in \mathbb{R}^{F \times F'}$ is a unique weight matrix associated with a particular k.

Even then, graph convolutional filters can only perform *linear* transformations on the node features, which limits their expressive power. Graph Convolutional Neural Networks extend the expressive power of GCFs with non-linear transformations. These are discussed in the next section.

2.2.2. Graph Convolutional Neural Networks

A **Graph Convolutional Neural Network (GCNN)** is a type of **Graph Neural Network (GNN)** that uses graph convolutional filters for performing node feature transformations. A GCNN consist of a sequence of graph filters, interleaved with non-linearities, resulting in a layered structure. In a GCNN, a single layer is simply defined as a GCF combined with a non-linearity:

$$\boldsymbol{X}^{(l+1)} = f^{(l)}(\boldsymbol{X}^{(l)}, \boldsymbol{S}) = \sigma \left[\sum_{k=0}^{K} \boldsymbol{S}^k \boldsymbol{X}^{(l)} \boldsymbol{W}_k^{(l)} \right], \tag{2.4}$$

where $l \in \{1, \ldots, L\}$ indexes the network layers, $\boldsymbol{W}_k^{(l)} \in \mathbb{R}^{F^{(l)} \times F^{(l+1)}}$ denotes the network weights at layer l for power k, and σ is a point-wise non-linearity. This allows us to define a GCNN $f(\boldsymbol{X}, \boldsymbol{S}) \in \mathbb{R}^{N \times F^{(L+1)}}$ in its totality:

$$\boldsymbol{X}^{(L+1)} = f(\boldsymbol{X}, \boldsymbol{S}) = f^{(L)}(f^{(L-1)}(\ldots f^{(1)}(\boldsymbol{X}, \boldsymbol{S}) \ldots, \boldsymbol{S}), \boldsymbol{S}) \quad \text{[10]}. \tag{2.5}$$

Typically, the non-linearity σ is omitted for the last layer $f^{(L)}$. The output of the final layer L is also commonly written as $\boldsymbol{Z} = \boldsymbol{X}^{(L+1)}$.

2.2.3. The Graph Fourier Transform

When a graph shift operator S is symmetric, which is the case when the graph is undirected, it can be decomposed into its eigenvector basis $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N] \in \mathbb{R}^{N \times N}$ and eigenvalue matrix $\boldsymbol{\Lambda} = \text{diag}([\lambda_1, \ldots, \lambda_N]) \in \mathbb{R}^{N \times N}$:

$$\boldsymbol{S} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\mathbf{H}}. \tag{2.6}$$

This decomposition is useful because $\boldsymbol{V}$ and $\boldsymbol{V}^{\mathbf{H}}$ can be seen as the mappings from and to the **graph frequency domain**, respectively. The eigenvalues $\lambda_1, \ldots, \lambda_N$ represent the different possible frequencies on the graph that the graph signal can be decomposed into. For a specific type of GSO, namely the Laplacian Matrix $\boldsymbol{L}$, these eigenvalues are interpretable as an ordering of frequencies from small to large. Namely, when putting the eigenvalues in ascending order, $\lambda_1 \leq \cdots \leq \lambda_N$, this gives the respective ordering of graph frequencies from low to high (see Figure 2.3).

The **Graph Fourier Transform (GFT)** is defined as the projection of a graph signal $\boldsymbol{X}$ into the graph frequency domain: $\widehat{\boldsymbol{X}} = \boldsymbol{V}^{\mathbf{H}}\boldsymbol{X}$, where $\boldsymbol{V}^{\mathbf{H}}$ denotes the conjugate transpose of eigenbasis $\boldsymbol{V}$ [44].

Projecting a signal $\boldsymbol{X}$ onto the graph spectrum provides useful insights into how that signal changes over a graph. A high $\hat{x}_{ij} = \left[\boldsymbol{V}^{\mathbf{H}}\boldsymbol{X}\right]_{ij}$ indicates that signal feature j strongly varies along the graph according to frequency λ_i. That is, if λ_i is a high frequency, then feature j changes rapidly between neighboring nodes (see $\mathcal{G}_2$ in Figure 2.3). Conversely, if λ_i is a low frequency, then feature j changes slowly between neighboring nodes (see $\mathcal{G}_1$ in Figure 2.3).

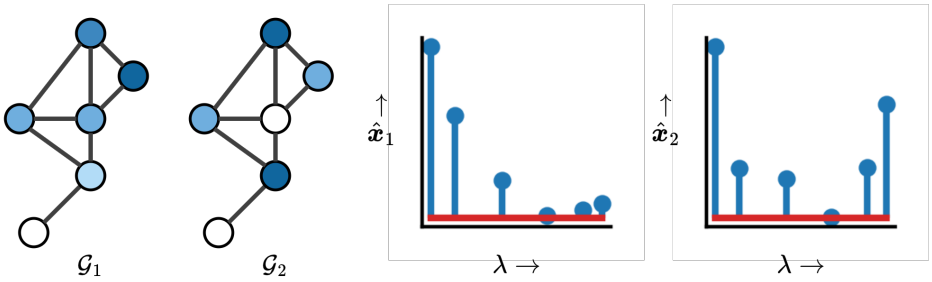


Figure 2.3: Two identically structured graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are shown, each having a distinct graph signal. These signals $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are visualized through varying node colorings. For each graph, a plot is given with their respective features projected onto the graph spectrum, giving $\hat{\boldsymbol{x}}_1$ and $\hat{\boldsymbol{x}}_2$. This was done using the respective Laplacian matrices $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$. Hence, the eigenvalues λ plotted on the horizontal axes correspond to frequencies ordered from low to high. Note that the node features of $\mathcal{G}_1$ are more smoothly distributed than those of $\mathcal{G}_2$. This is reflected by a weaker response on higher frequencies for $\hat{\boldsymbol{x}}_1$ on the graph spectrum.

The graph fourier transform can give additional insights in the workings of graph filters. To show this, let us consider a one-dimensional graph convolutional filter, given by

$$h(\boldsymbol{x}, \boldsymbol{S}) = \sum_{k=0}^{K} w_k \boldsymbol{S}^k \boldsymbol{x}, \tag{2.7}$$

where $\boldsymbol{x} \in \mathbb{R}^N$ is a one-dimensional graph signal. Using (2.6), this can be rewritten as:

$$\boldsymbol{V}^{\mathbf{H}} h(\boldsymbol{x}, \boldsymbol{S}) = \sum_{k=0}^{K} w_k \boldsymbol{\Lambda}^k (\boldsymbol{V}^{\mathbf{H}} \boldsymbol{x}). \tag{2.8}$$

This reveals that graph filters are point-wise operators in the graph frequency domain: the filter's weights w_k can be interpreted as 'filter taps', which individually strengthen and weaken different powers of frequencies $\lambda \in \{\lambda_1, \ldots, \lambda_N\}$ [44].

From this arises the concept of the **graph frequency response** of a filter with respect to a particular frequency λ:

$$h_{\text{gfr}}(\lambda) = \sum_{k=0}^{K} w_k \lambda^k. \tag{2.9}$$

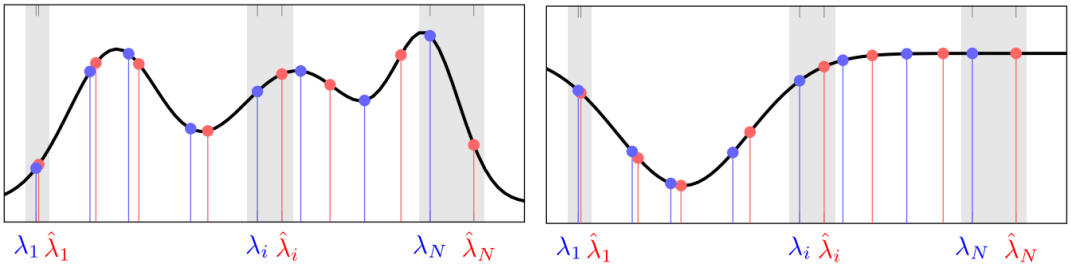
It describes how the strength of any graph frequency λ in an input signal changes as a result of a graph filter with weights w_k being applied [10].

2.2.4. Integral Lipschitz Graph Filters And GCNNs

It has been proven that convolutional graph filters and GCNNs possess innate stability properties towards various types of perturbation. These stability properties are discussed in a later section (2.3.2). Many of these proofs rely on the respective filters and GCNNs being **Integral Lipshitz (IL)**. Effectively, being Integral Lipschitz prevents GCNNs and GCFs from attaining highly unstable weight configurations. Formally, Integral Lipschitz filters are defined as follows: a GCF h is IL if there exists a constant C_{IL} such that for any pair of eigenvalues λ_1 and λ_2 it holds that

$$|h_{\text{gfr}}(\lambda_1) - h_{\text{gfr}}(\lambda_2)| \leq C_{\text{IL}} \frac{|\lambda_1 - \lambda_2|}{|\lambda_1 + \lambda_2|/2}, \tag{2.10}$$

where h_{gfr} is the filter's graph frequency response (2.9). By extension, a GCNN is Integral Lipschitz if its internal graph filters are IL and if its non-linearities are Lipschitz continuous [9]. GCFs that are IL



(a) The graph frequency response h_{gfr} for a Lipschitz filter.

(b) The graph frequency response h_{gfr} for an **Integral Lipschitz** filter.

Figure 2.4: Depicted are the Graph Frequency Response (GFR) plots for a "regular" GCF (left) and a GCF that is Integral Lipschitz (right). The blue λ_i's and the red $\hat{\lambda}_i$'s on the horizontal axes represent the eigenvalues of an original and a perturbed graph, respectively. Note that perturbation causes larger shifts on larger values of λ. As a result, the outputs produced by a regular h_{gfr} can become unstable for frequencies associated with large λ. A filter that is IL, however, is constrained to be mostly flat for large λ's, thus preserving stability under perturbation, even for high λ. These figures are obtained from [9].

are more stable, because it forces its graph frequency response to flatten for high frequencies (see Figure 2.4). This fact becomes evident when observing that (2.10) requires the derivative of the frequency response, h'_{gfr}, to satisfy

$$|\lambda h'_{\text{gfr}}(\lambda)| \leq C_{\text{IL}}. \tag{2.11}$$

Thus, the IL constant C_{IL} effectively puts a limit on how the fast a filter's frequency response can change, where this limit grows tighter for larger λ. For further information on the properties of Integral Lipschitz GCFs and GCNNs, see [9].

2.2.5. Other GNN Architectures

While the Graph *Convolutional* Neural Networks are the focus of this thesis, this is only a subset of all possible types of **Graph Neural Network (GNN)**. In this section, two additional types of GNN are discussed: GCN [24] and GIN [63]. These are by far the two most commonly used styles of GNN in Graph Contrastive Learning literature.

While having a confusingly similar name, the **Graph Convolutional Network (GCN)** [24] describes a much more specific architecture than GCNN. A GCN layer is defined as the degree-normalized summation over the features of a node's direct neighbors as well as its own features:

$$\begin{aligned} f^{(l)}(\boldsymbol{X}^{(l)}, \boldsymbol{A}) &= \sigma\left[\bar{\boldsymbol{D}}^{-1/2}\bar{\boldsymbol{A}}\bar{\boldsymbol{D}}^{-1/2}\boldsymbol{X}^{(l)}\boldsymbol{W}^{(l)}\right], \\ \text{where } \bar{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I} \text{ and } [\bar{\boldsymbol{D}}]_{ij} &= \begin{cases} \sum_m^N [\bar{\boldsymbol{A}}]_{im} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}. \end{aligned} \tag{2.12}$$

Here, $\boldsymbol{A}$ is the adjacency matrix, $\bar{\boldsymbol{A}}$ is the adjacency matrix with added self-loops and $\bar{\boldsymbol{D}}$ is the degree matrix of this $\bar{\boldsymbol{A}}$. Note that GCN is, in fact, a specific case of a GCNN (2.4) where $\boldsymbol{S} = \bar{\boldsymbol{D}}^{-1/2}\bar{\boldsymbol{A}}\bar{\boldsymbol{D}}^{-1/2}$, $K = 1$ and $\boldsymbol{W}_1^{(l)} = \boldsymbol{W}^{(l)}$ for all $l \in 1..L$, $\boldsymbol{W}_0^{(l)} = \boldsymbol{0}$ for all $l \in 1..L$. Despite its simplicity, GCN has seen great success on many node-level tasks and is therefore a widely popular architecture.

The **Graph Isomorphism Network (GIN)** [63], is popular in particular for doing graph-level tasks. A GIN layer is defined by:

$$f^{(l)}(\boldsymbol{X}^{(l)}, \boldsymbol{A}) = \phi^{(l)}\left((\boldsymbol{A} + (1+\epsilon)\cdot\boldsymbol{I})\boldsymbol{X}^{(l)}\right), \tag{2.13}$$

where $\phi^{(l)}$ is a Multi-Layer Perceptron (MLP) of arbitrary depth and ϵ is a trainable scalar parameter. This ϵ effectively controls the strength of the self-loop on every node. The inclusion of a free-form MLP allows a GIN to learn complex transformations without the need for many GNN propagations. Note that GIN containing only single-layer MLPs $\phi^{(l)}$ is also a specific case of GCNN. If we take $\boldsymbol{W}^{(l)}$ to be the weights of the MLP at layer l, then GIN can be defined as a GCNN where $K = 1$, $\boldsymbol{W}_0^{(l)} = (1+\epsilon)\boldsymbol{W}^{(l)}$ for all $l \in 1..L$, $\boldsymbol{W}_1^{(l)} = \boldsymbol{W}^{(l)}$ for all $l \in 1..L$, and $\boldsymbol{S} = \boldsymbol{A}$.

2.2.6. Machine Learning Tasks On Graph Data

There are many useful tasks one could do on graph data. Some examples of such tasks are: detecting adversarial agents in a social network, predicting water levels in a piping network, or recognizing enzymes among a dataset of proteins. Tasks on graph data can be distributed across three categories: **node-level tasks**, **edge-level tasks** and **graph-level tasks** (see Figure 2.5). These involve predicting some properties of the nodes, edges or entire graphs, respectively. For graph-level tasks, a dataset always contains multiple graphs, for node and edge-level tasks, there often is just one.

Graph-level tasks are typically **inductive**, meaning that a model is trained on a set of graphs to then be used and evaluated on newly sampled graphs that are outside the training set. Node-level and edge-level tasks that only involve one graph are **transductive**. Here, the trained model is not expected to generalize beyond this one graph. Note that node and edge-level tasks can also be inductive in the case of a multi-graph dataset.

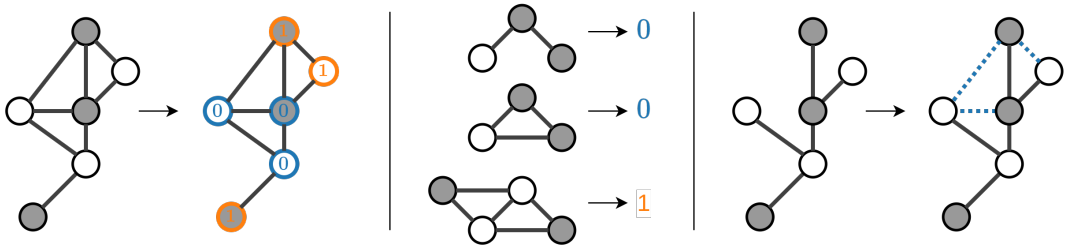


Figure 2.5: Depicted are three examples of tasks on graph data: node classification (left), graph classification (middle), and link prediction (right).

Given that GCNNs only work on node-level features $\boldsymbol{X} \in \mathbb{R}^{N \times F_{\text{node}}}$, the output of a GCNN cannot directly be used to do a graph-level task. For this, the node representations have to be summarized into a single graph representation $\boldsymbol{z} \in \mathbb{R}^{F_{\text{graph}}}$. This is done using a **readout** function $r : \mathbb{R}^{N \times F_{\text{node}}} \to \mathbb{R}^{F_{\text{graph}}}$, producing $\boldsymbol{z} = r(\boldsymbol{Z})$. Such a readout function should be permutation-invariant and compatible with different graph sizes. Examples of simple, yet commonly used readout functions are taking the sum, mean, or maximum over the output node features.

2.3. GCNN Stability

In machine learning on graphs, much like any other data domain, it is desirable to train models that are **stable**. A model is said to be stable if its outputs remain consistent, even if the given input is changed by a little (see Figure 2.6). This is desirable because, e.g. in a node classification problem, small changes to the graph structure should not cause the model to classify a node completely differently.

Changes to the the input are called **perturbations** and, for graphs, such perturbations can be applied to its features $\boldsymbol{X}$ or to its structure $\boldsymbol{S}$. A perturbation can formally be written as function $t : \mathcal{G} \to \mathcal{G}$, taking in a graph and producing another graph. The node features and the graph shift operator of a perturbed graph $\widetilde{\mathcal{G}} = t(\mathcal{G})$ are denoted $\widetilde{\boldsymbol{X}}$ and $\widetilde{\boldsymbol{S}}$, respectively.

Some examples of structural graph perturbations are randomly dropping edges, adding noise to edge weights, randomly dropping nodes, or randomly sampling a connected subgraph. Examples of node feature perturbations are: adding random noise to node features, randomly masking out features, or shuffling features over nodes. Some of these perturbation styles are illustrated in Figure 2.7.

2.3.1. Measuring GCNN Stability

To evaluate the (in)stability of a GCNN f under a particular perturbation t, one uses a distance metric to compare the f's perturbed and unperturbed output. One such a distance metric is given by:

$$\left\| f(\boldsymbol{X}, \boldsymbol{S}) - f(\widetilde{\boldsymbol{X}}, \widetilde{\boldsymbol{S}}) \right\|_{\text{abs}} = \sum_{j=1}^{F_{\text{out}}} \left\| [\boldsymbol{Z}^{\intercal}]_j - [\widetilde{\boldsymbol{Z}}^{\intercal}]_j \right\|_2, \tag{2.14}$$

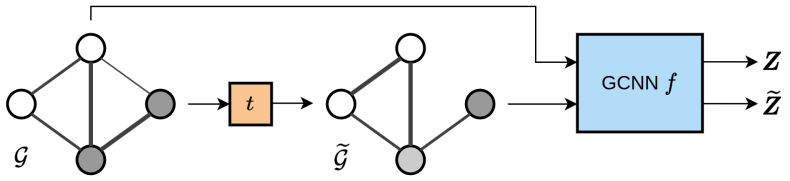


Figure 2.6: Shown are an original graph $\mathcal{G}$ and a perturbed graph $\widetilde{\mathcal{G}} = t(\mathcal{G})$. A GCNN f is stable under the perturbation t if its respective outputs, $\boldsymbol{Z} = f(\mathcal{G})$ and $\widetilde{\boldsymbol{Z}} = f(\widetilde{\mathcal{G}})$ are approximately equal.

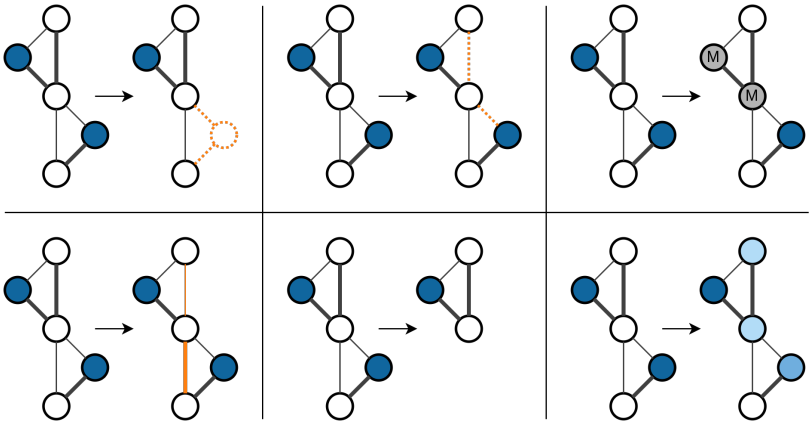


Figure 2.7: Given are six examples of perturbations on graph data: node dropping (top-left), edge dropping (top-center), node feature masking (top-right), edge weight perturbation (bottom-left), subgraph sampling (bottom-center), and applying noise to node features (bottom-right).

where $F_{\text{out}} = F^{(L+1)}$ is the output dimension of f, $\boldsymbol{Z} = f(\boldsymbol{X}, \boldsymbol{S})$ and $\widetilde{\boldsymbol{Z}} = f(\widetilde{\boldsymbol{X}}, \widetilde{\boldsymbol{S}})$ are unperturbed and perturbed outputs, and $[\boldsymbol{Z}^{\intercal}]_j$ denotes indexing of the jth column vector of $\boldsymbol{Z}$. This metric and simple variations of it are used in GCNN stability literature [9, 11].

This particular metric is an *absolute* measure of instability: it scales with the size of the output features of the GCNN. To make stability comparisons between GCNNs with differently scaled outputs, one can normalize by dividing over the size of the unperturbed output features:

$$\left\| f(\boldsymbol{X}, \boldsymbol{S}) - f(\widetilde{\boldsymbol{X}}, \widetilde{\boldsymbol{S}}) \right\|_{\text{n}} = \frac{\sum_{j=1}^{F_{\text{out}}} \left\| [\boldsymbol{Z}^{\intercal}]_j - [\widetilde{\boldsymbol{Z}}^{\intercal}]_j \right\|_2}{\sum_{j=1}^{F_{\text{out}}} \| [\boldsymbol{Z}^{\intercal}]_j \|_2}. \tag{2.15}$$

This a multi-feature variant of the normalized stability metric used in [23]. It also is the primary instability metric we use in our experiments for this thesis.

2.3.2. GCNN Stability Bounds

GCNNs are proven to be stable under certain types of perturbation. In literature, such a stability property is expressed as an upper bound to instability. This section discusses two of such bounds.

Firstly, GCNNs are stable to small variations in the edge weights w of a graph [10]. This style of structural perturbation we refer to as **Edge Weight Perturbation (EWP)** and can be seen in Figure 2.7, bottom-left. This stability is expressed as an upper bound to the instability measure in (2.14):

$$\left\| f(\boldsymbol{X}, \boldsymbol{S}) - f(\boldsymbol{X}, \widetilde{\boldsymbol{S}}) \right\|_{\text{abs}} \leq \Delta L F^{L-1} \epsilon + \mathcal{O}(\epsilon^2), \tag{2.16}$$

where f is a GCNN that is Integral Lipschitz with depth L and width F, Δ is a constant dependent on the IL constant C_{IL} and the style of EWP, ϵ is a variable proportional to the strength of the perturbation, and $\mathcal{O}(\epsilon^2)$ is big O notation characterizing squared growth with respect to ϵ. The value of ϵ depends on the difference between $\boldsymbol{S}$ and $\widetilde{\boldsymbol{S}}$: ϵ is an upper bound to operator norm of their difference matrix, $\|\boldsymbol{S} - \widetilde{\boldsymbol{S}}\|_{\text{op}} \leq \epsilon$. Regarding the Δ constant, for purposes of this thesis, it is defined as $\Delta = 2C_{\text{IL}}$ [10].

Secondly, GCNNs are stable to random **Edge Dropping (ED)** [11], seen in Figure 2.7, top-center. This perturbation is defined as uniform removal of edges:

$$\widetilde{\boldsymbol{S}} = \boldsymbol{S} \odot \boldsymbol{B} \quad \text{where } [\boldsymbol{B}]_{ij} \sim \text{Bernoulli}(p_{\text{remove}}) \; \forall ij. \tag{2.17}$$

Here, p_{remove} is the probability of dropping any particular edge and $\odot$ denotes an entry-wise product. The expected instability, measured using a squared version of (2.14), is upper bounded as follows:

$$\begin{aligned} \mathbb{E}\left[\left\| f(\boldsymbol{X}, \boldsymbol{S}) - f(\boldsymbol{X}, \widetilde{\boldsymbol{S}}) \right\|_{\text{abs}^2}\right] &= \mathbb{E}\left[\sum_{j=1}^{F_{\text{out}}} \left\| ([\boldsymbol{Z}^{\intercal}]_j - [\widetilde{\boldsymbol{Z}}^{\intercal}]_j) \right\|_2^2\right] \\ &\leq C \cdot p_{\text{remove}} \cdot \sum_{j=1}^{F_{\text{out}}} \|[\boldsymbol{Z}^{\intercal}]_j\|_2^2 + \mathcal{O}(p_{\text{remove}}), \end{aligned} \tag{2.18}$$

where f is a GCNN that is IL and C is a constant defined as $C = N\alpha C_{\text{IL}}^2 L^2 C_{\sigma}^{2L} F^{2L-2}$. Here, F is GCNN's width, L is its depth, α is a small constant depending on the style of GSO, C_{IL} is the IL constant (2.10), and C_{σ} is the Lipschitz constant of the non-linearity σ. This bound is a multi-feature version of the one given in [11].

Note that, in the original paper, (2.18) is defined as an upper bound to *stochastic* edge dropping. That is, edges are not just dropped from the input, but can drop at any time during forward propagation [11]. In this thesis, however, we simply consider it to be a loose upper bound to input perturbation.

2.3.3. The Link Between GCNN Stability And Architecture

Both stability bounds discussed in the previous subsection, (2.16) and (2.18), contain the terms L and F, denoting the depth and width of the GCNN, respectively. The theoretical stability of a GCNN thus depends on its architecture. For both bounds, more complex GCNNs, i.e. deeper and wider ones, have worse stability guarantees.

Additionally, both bounds are also affected by the order K of the GCNN. This is because the Integral Lipschitz constant C_{IL} is dependent on K. To see why, note that C_{IL} is an upper bound to the derivative of the graph frequency response (2.11). This derivative h'_{gfr} contains a sum of $K-1$ terms multiplied by $k \in 1 \dots K$:

$$h'_{\text{gfr}}(\lambda) = \sum_{k=1}^{K} k w_k \lambda^{k-1}. \tag{2.19}$$

The order K can be interpreted to scale up C_{IL} as it controls the size of the sum and the scaling of its terms. A particular arrangement of positive and negative weights w_k could however nullify the scaling effect of K. That said, even if the values for w_k are centered around zero, their variance is effectively being scaled by K and, consequently, a larger K does increase the likelihood of a large C_{IL}.

Thus, we conclude that both bounds give weaker stability guarantees for GCNNs with larger L, F, or K. This observation motivates our hypothesis for **RQ2** that GCNN stability under these styles of perturbation grow worse when increasing any of the architectural hyperparameters L, F, or K.

2.4. Self-Supervised Learning

Self-Supervised Learning (SSL) is a specific flavor of unsupervised learning where one trains a model on a synthetic task, called the **pretext task** or **pre-training task**, to subsequently make it easier to learn the actual task of interest, called the **downstream task**. The crux of this two-part setup is that the pretext task does not require any labeled data. It is designed in such a way that the model can supervise itself during pre-training using only unlabeled data. This pre-training then allows the downstream task to be learned with fewer labeled data.

Due to the discrepancy between the pre-text task and the downstream task, a model trained solely on the pre-text task is unlikely to perfectly transfer to the downstream task. Two commonly used methods to bridge this gap are:

1. **Fine-tuning**. Here, training on the pre-trained model is simply continued on the downstream task, typically only adjusting a subset of the model weights.

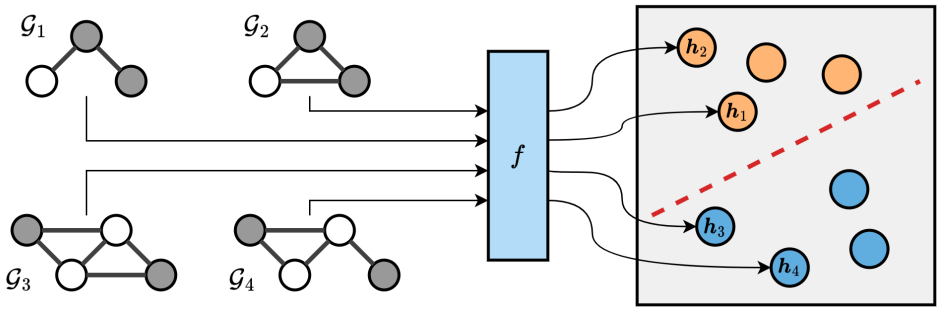


Figure 2.8: The goal of representation learning is to train an encoder f that maps raw inputs to high-level, semantically rich representations. In the case of graph representation learning, as seen here, the elements of semantically similar graph sets, $\{\mathcal{G}_1, \mathcal{G}_2\}$ and $\{\mathcal{G}_3, \mathcal{G}_4\}$, should be clustered together in representation space, while dissimilar graphs should be far apart. Training an f to produce such representations makes it easier for a downstream classifier to distinguish semantically different groups. In the visualized representation space on the left, the dashed red line represents a possible decision boundary used by a trained downstream classifier.

2. **Representation Learning**. This involves pre-training an *encoder* to learn a mapping from the input space to a more semantically rich representation space. Unlike the raw inputs, good representations minimize task-irrelevant information such as noise and redundancy and maximize task-relevant information. After pre-training, the encoder is used as a data pre-processor for training a simple model on the downstream task.

This thesis focuses on the representation learning style of SSL and, hence, this is further discussed in the next section.

2.4.1. Representation Learning

In machine learning, the performance of models strongly depends on the quality of the chosen input features $\boldsymbol{x} \in \mathbb{R}^{F_{\text{in}}}$. Firstly, better performance is typically achieved when datapoints are nicely distributed over the feature space. Here, "nicely distributed" could mean in e.g. a classification setting that datapoints of different classes are well-separated from one another in the feature space. Secondly, the dimensionality F_{in} of the input features also impacts model performance. If the dataset is not suffiently large to match the high dimensionality of the inputs, a model is likely to overfit on the data, harming the model's ability to generalize. This negative effect associated with high-dimensional inputs is known as the **curse of dimensionality**.

Thus, it is desirable to derive features from the input data that (1) nicely spread out datapoints over the feature space and that (2) preserve all task-relevant information in the data, while minimizing feature dimensionality. The goal of representation learning is to train an encoder $f : \mathbb{R}^{F_{\text{in}}} \to \mathbb{R}^{F_{\text{rep}}}$ that achieves exactly this. It transforms high-dimensional, high-redundancy inputs $\boldsymbol{x} \in \mathbb{R}^{F_{\text{in}}}$ into lower-dimensional, low-redundancy representations $\boldsymbol{h} \in \mathbb{R}^{F_{\text{rep}}}$ (see Figure 2.8).

2.4.2. Evaluating Representation Quality

How does one test the quality of an encoder's representations? Trying to directly understand the meaning of the representations is typically not possible since machine-learned representations are often not human-interpretable. And, in general, it is hard for humans to interpret high-dimensional data.

To address this last problem, one can use the **t-distributed Stochastic Neighbor Embedding (t-SNE)** algorithm to reduce the dimensionality of representations to 2D space (see Figure 2.9). This allows one to visualize the spread of datapoints in representation space and to qualitatively assess the representations. To *quantitatively* measure representation quality, one can do a **downstream evaluation**. Here, a simple linear model is trained on the downstream task, taking the learned representations as its input. Its performance on this task then forms a measure for representation quality. In this thesis, downstream performance is the metric of choice for assessing representation quality.

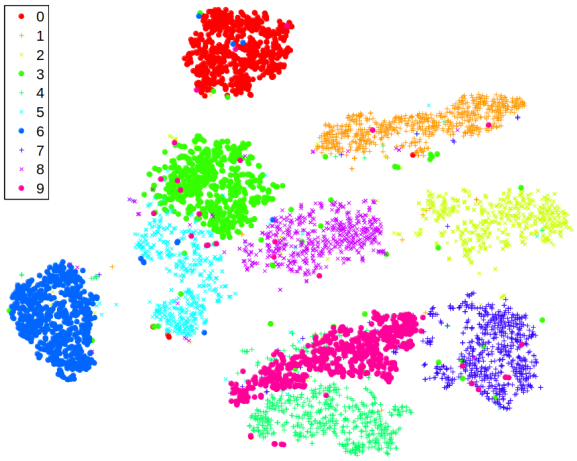


Figure 2.9: An example of a t-SNE plot on the MNIST dataset. MNIST is a collection of 28-by-28 pixel images of handwritten digits. Datapoints in MNIST thus have a feature dimensionality of $28^2 = 784$. The t-SNE algorithm helps transform these high-dimensional features into 2D features, while attempting to preserve relative distances between data points.

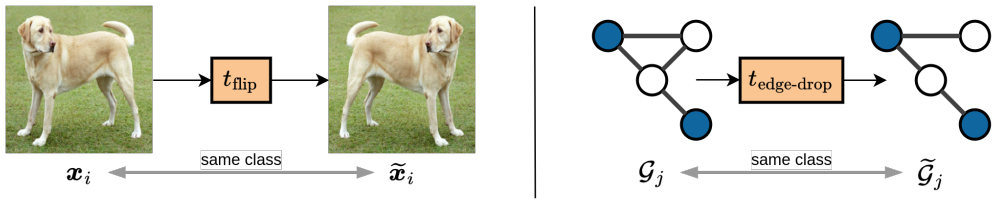


Figure 2.10: In contrastive learning, augmentations are used for creating positive pairs. A good augmentation should significantly perturb the data while preserving task-relevant information in the data. E.g. in the case of image classification (left), a sensible data augmentation is horizontal flipping, because the flipped image can be assumed to still belong to the same class; a flipped picture of a dog is still a picture of a dog. In the case of graph classification (right), a sensible augmentation could be edge dropping. Though, unlike for images, for graphs it is often hard to intuitively verify if an augmentation indeed preserves task-relevant information due to their abstract nature.

2.5. Contrastive Learning

The SSL technique central to this thesis is **Contrastive Learning (CL)**. In CL, representations are learned by pulling similar datapoints together and pushing dissimilar datapoints apart in the representation space. Of course, without access to labels, it is hard to define a similarity metric that rhymes with the actual semantics of the data. CL overcomes this problem by synthetically creating pairs of similar datapoints. It does so by pairing up original datapoints with perturbed versions of themselves. For example, on image data one could flip the original image horizontally as the perturbation. Here, a good perturbation maximally distorts task-irrelevant information while preserving task-relevant information (see Figure 2.10). Then, when pushing the perturbed and original data together in representation space, the encoder learns to encode task-relevant information in its representations, while omitting irrelevant information.

Creating such a perturbed version $\widetilde{x}$ of an input x is referred to as **augmenting** x or creating a **view** of x. In this context, the original datapoint x is called the **anchor** and the generated augmentations of that datapoint are called **positive samples** for that original datapoint, written here as x^+. Together with the anchor, a positive sample forms a **positive pair**: a pair of semantically similar datapoints. Two different augmentations $x^{+,1}$ and $x^{+,2}$ of the same anchor x are in some settings also considered to be a positive pair.

By using augmentations to create a set of positive pairs $\mathcal{D}^+ \in \mathcal{P}(\mathbb{R}^{F_{\text{in}}} \times \mathbb{R}^{F_{\text{in}}})$ from a dataset $\mathcal{D} \in \mathcal{P}(\mathbb{R}^{F_{\text{in}}})$, one can construct a training target to find an optimal f^*:

$$f^* = \arg\max_f \bigoplus_{(\boldsymbol{x},\boldsymbol{x}^+)\in\mathcal{D}^+} \text{sim}(f(\boldsymbol{x}), f(\boldsymbol{x}^+)). \tag{2.20}$$

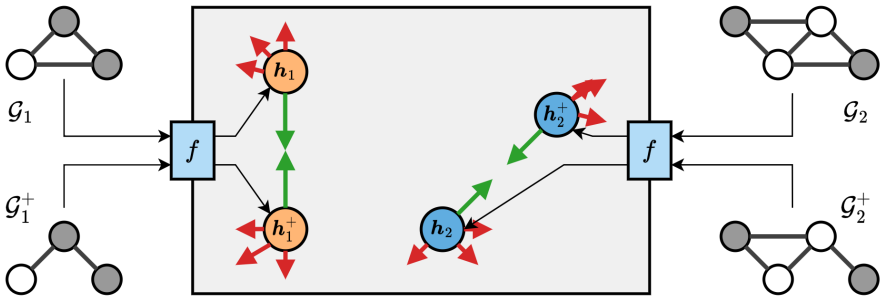


Figure 2.11: In contrastive learning, representations are learned by pulling positive data pairs closer to one another in the representation space. Illustrated are two positive pairs, $(\mathcal{G}_1, \mathcal{G}_1^+)$ and $(\mathcal{G}_2, \mathcal{G}_2^+)$, and their respective representations as produced by f, $(\boldsymbol{h}_1, \boldsymbol{h}_1^+)$ and $(\boldsymbol{h}_2, \boldsymbol{h}_2^+)$. This pulling force is visualized with green arrows. To avoid a representational collapse, e.g. all representations being pulled to the same point, another force is added that makes representations of dissimilar data points to repel each other. These forces associated with negative pairs are visualized with red arrows. In practice, this repelling force is often applied to all possible data pairs, including the positive ones.

This states that an optimal encoder f^* maximizes similarity between positive pairs. And thus, if all positive pairs are truly semantically similar, then an optimal f^* will nicely distribute datapoints over the representation space. Here, $\text{sim}(\cdot)$ is a simple similarity metric and $\oplus$ is an operator that combines the similarities of all positive pairs into a single maximization problem. A commonly used similarity metric is the cosine similarity: $\text{sim}(\boldsymbol{h}_1, \boldsymbol{h}_2) = \boldsymbol{h}_1^\intercal \boldsymbol{h}_2 / \|\boldsymbol{h}_1\| \|\boldsymbol{h}_2\|$ and a natural choice for $\oplus$ would be some sort of summation.

This simple optimization target however has a crucial problem: an encoder f can learn to always return the same value for a pair of datapoints, regardless of their actual semantic (dis)similarity. Such an encoder would still maximize (2.20). This phenomenon is called a **collapse** of the representation space. Such a collapse can be avoided by also considering so-called **negative pairs** during optimization. A sample $\boldsymbol{x}^-$ forms a negative pair with an anchor $\boldsymbol{x}$ if the two have *dissimilar* semantics. A method that works well in practice is simply pairing up random samples from the dataset $\mathcal{D}$. Note that this method may create false negatives by accidentally pairing up two semantically similar datapoints. With these negative pairs $\mathcal{D}^-$, we can define a training target that resists collapse:

$$f^* = \arg\max_f \left[\bigoplus_{(\boldsymbol{x}, \boldsymbol{x}^+) \in \mathcal{D}^+} \text{sim}(f(\boldsymbol{x}), f(\boldsymbol{x}^+)) \quad - \bigoplus_{(\boldsymbol{x}, \boldsymbol{x}^-) \in \mathcal{D}^-} \text{sim}(f(\boldsymbol{x}), f(\boldsymbol{x}^-)) \right]. \tag{2.21}$$

Here, an optimal f^* maximizes similarity between positive pairs while minimizing similarity between negative pairs. A visualization of how this affects the representation space is given in Figure 2.11.

2.5.1. The InfoNCE Loss

The maximization targets in (2.20) and (2.21) provide a generic framework for contrastive learning. An example of a more concrete training loss, i.e. a *minimization* target, is the **InfoNCE loss** [36]. This loss is based on a categorical cross-entropy loss for identifying a positive sample among a set of unrelated noise samples. Effectively, training a model to minimize InfoNCE is equivalent to maximizing the probability that this model, combined with the similarity metric $\text{sim}(\cdot)$, correctly identifies the positive sample in a mixed set $\{\boldsymbol{x}_i^+\} \cup \mathcal{D}^-$. The InfoNCE loss is mathematically defined as:

$$\begin{aligned} \mathcal{L}_{\text{InfoNCE}} &= \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_i^+) \in \mathcal{D}^+} \mathcal{L}_{\text{InfoNCE},i}(\boldsymbol{x}_i, \boldsymbol{x}_i^+) \\ \mathcal{L}_{\text{InfoNCE},i}(\boldsymbol{x}_i, \boldsymbol{x}_i^+) &= -\log \frac{\text{sim}(f(\boldsymbol{x}_i), f(\boldsymbol{x}_i^+))}{\text{sim}(f(\boldsymbol{x}_i), f(\boldsymbol{x}_i^+)) + \sum_{(\boldsymbol{x}_i, \boldsymbol{x}^-) \in \mathcal{D}^-} \text{sim}(f(\boldsymbol{x}_i), f(\boldsymbol{x}^-))} \\ &\text{where } \text{sim}(\boldsymbol{h}_1, \boldsymbol{h}_2) = e^{\boldsymbol{h}_1^\intercal \boldsymbol{h}_2 / \tau}. \end{aligned} \tag{2.22}$$

Here, τ is the temperature hyperparameter, controlling the "sharpness" of the similarity metric $\text{sim}(\cdot)$. For τ's closer to zero, the encoder f is encouraged to output representations that produce more confident (dis)similarities and, for larger τ's, more uncertainty is tolerated.

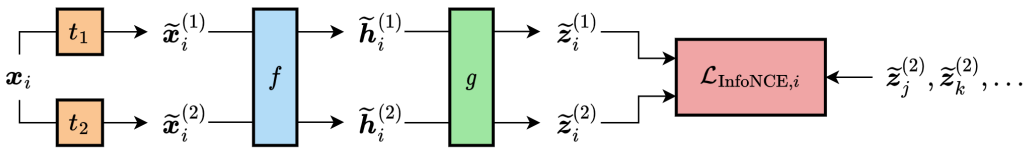


Figure 2.12: A graphical overview of the SimCLR architecture. During a forward pass, for each sample $\boldsymbol{x}_i$ in the batch, two views are generated, $\widetilde{\boldsymbol{x}}_i^{(1)}$ and $\widetilde{\boldsymbol{x}}_i^{(2)}$. The encoder f transforms them into representations $\widetilde{\boldsymbol{h}}_i^{(1)}$ and $\widetilde{\boldsymbol{h}}_i^{(2)}$ and these are subsequently transformed by the projector g into projections $\widetilde{\boldsymbol{z}}_i^{(1)}$ and $\widetilde{\boldsymbol{z}}_i^{(2)}$. These projections are then used as a positive pair in the InfoNCE loss (2.23). Negative pairs obtained by pairing up the first view $\widetilde{\boldsymbol{x}}_i^{(1)}$ with the second view $\widetilde{\boldsymbol{x}}_j^{(2)}$ of every sample $\boldsymbol{x}_j$ in the batch.

2.5.2. Weak Data Augmentations

The choice of data augmentation function $t : \mathbb{R}^{F_{\text{in}}} \to \mathbb{R}^{F_{\text{in}}}$ is crucial to the ability of contrastive methods to learn good representations. An augmentation function t can't be too disruptive as the view $\widetilde{\boldsymbol{x}} = t(\boldsymbol{x})$ should preserve task-relevant semantics of the anchor $\boldsymbol{x}$ in order for the pair $(\boldsymbol{x}, \widetilde{\boldsymbol{x}})$ to be considered a positive pair. At the same time, an augmentation function t also can't be too weak. When training an encoder f with overly weak augmentation, it cannot learn to distinguish task-relevant and task-irrelevant information. As a result, there will be no clustering of semantically similar data in representation space; the encoder will simply learn to distribute all datapoints evenly over the representation space.

What this demonstrates is that there exists no one-size-fits-all augmentation that works well for all types of data. To get a good trade-off between disruption of irrelevant information and preservation of relevant information, one needs to carefully design augmentations around the particular data domain of interest. An overly generic augmentation, e.g. one that only adds random noise to the features, is likely to destroy task-relevant information when it is applied with sufficient strength to create challenging positive pairs.

2.5.3. SimCLR

A well-known CL method in Computer Vision (CV) and one relevant to this thesis is **SimCLR** [4]. It creates positive pairs by generating two views $\widetilde{\boldsymbol{x}}^{(1)}$ and $\widetilde{\boldsymbol{x}}^{(2)}$. These are then passed through an encoder f to get two representations $\widetilde{\boldsymbol{h}}^{(1)}$ and $\widetilde{\boldsymbol{h}}^{(2)}$. Before passing these representations into the InfoNCE loss, however, they are mapped to projections $\widetilde{\boldsymbol{z}}^{(1)}$ and $\widetilde{\boldsymbol{z}}^{(2)}$ by a **projector** $g : \mathbb{R}^{F_{\text{rep}}} \to \mathbb{R}^{F_{\text{proj}}}$. This gives the following SimCLR loss:

$$\begin{aligned}
\mathcal{L}_{\text{SimCLR}} &= \sum_{\widetilde{\boldsymbol{x}}_i^{(1)} \in \widetilde{\mathcal{B}}_1, \widetilde{\boldsymbol{x}}_i^{(2)} \in \widetilde{\mathcal{B}}_2} \mathcal{L}_{\text{SimCLR},i}(\widetilde{\boldsymbol{x}}_i^{(1)}, \widetilde{\boldsymbol{x}}_i^{(2)}) \\
\mathcal{L}_{\text{SimCLR},i}(\widetilde{\boldsymbol{x}}_i^{(1)}, \widetilde{\boldsymbol{x}}_i^{(2)}) &= -\log \frac{\text{sim}\left(g(f(\widetilde{\boldsymbol{x}}_i^{(1)})), g(f(\widetilde{\boldsymbol{x}}_i^{(2)}))\right)}{\sum_{\widetilde{\boldsymbol{x}}_j^{(2)} \in \widetilde{\mathcal{B}}_2} \text{sim}\left(g(f(\widetilde{\boldsymbol{x}}_i^{(1)})), g(f(\widetilde{\boldsymbol{x}}_j^{(2)}))\right)} \\
&\text{where } \text{sim}(\boldsymbol{z}_1, \boldsymbol{z}_2) = e^{\boldsymbol{z}_1^\intercal \boldsymbol{z}_2 / \tau}.
\end{aligned} \tag{2.23}$$

Here, $\widetilde{\mathcal{B}}_1 = \{t_1(\boldsymbol{x}_i) \mid \boldsymbol{x}_i \in \mathcal{B}\}$ and $\widetilde{\mathcal{B}}_2 = \{t_2(\boldsymbol{x}_i) \mid \boldsymbol{x}_i \in \mathcal{B}\}$ denote the first and second augmented batches of the inputs $\boldsymbol{x}$ in the original batch $\mathcal{B}$. A visualization of the SimCLR pipeline is given in Figure 2.12. Note that the omitted similarity term in the denominator compared to (2.22) is accounted for by the fact that $\widetilde{\boldsymbol{x}}_i^{(2)}$ is also a member of $\widetilde{\mathcal{B}}_2$. The augmentations t_1 and t_2 are randomly sampled from a set of image transformations, including e.g. cropping, rotation, and color distortion.

The key difference between the encoder f and the projector g is that g is discarded when pre-training is over, while f is kept for use in downstream tasks. Projectors have become a commonly included component in modern CL architectures. Typically, a shallow Multi-Layer Perceptron (MLP) is used as the learnable backbone for the projector [4, 13, 67, 68, 72]. In the case of SimCLR, g is a 2-layer MLP. The intuition behind including such a projector is that it may not be desirable for the representations $\boldsymbol{h}$ to have complete invariance to the perturbations. The projector g is a way of further detaching pretext and downstream task.

2.5.4. Other Methods Of Avoiding Collapse

A drawback of relying on negative samples to avoid representational collapse is that it is memory inefficient. Due to the inclusion of negative pairs, it takes $\mathcal{O}(|\mathcal{B}|^2)$ space to calculate the InfoNCE loss. Here, $|\mathcal{B}|$ denotes the batch size. Simply using smaller batch sizes is not a solution as it weakens the theoretical guarantees of InfoNCE and, consequently, leads to degraded representation quality. To overcome this issue, new strategies were developed for preventing collapse that do not use negative samples. One such a strategy is **redundancy reduction**. Here, a term is added to the loss that encourages the output features $\boldsymbol{z}$ to have a non-zero variance across samples, while discouraging correlation between different features. This ensures that each individual feature encodes distinctly different information and forces different samples to produce different representations, thus preventing collapse [1, 68].

Another strategy for avoiding collapse is **self-distillation**. To do this, self-distillation relies on a dual, asymmetric encoder. Specifically, it employs two different networks in parallel, an online network and a target network, through which two positive samples $\boldsymbol{x}^{+,1}$ and $\boldsymbol{x}^{+,2}$ are passed. The goal is to have both networks output the same representation, despite the asymmetry. Different self-distillation methods have different ways of making the architecture asymmetric [5, 13, 49]. An intuition for why this asymmetry helps against collapse is that it creates a discrepancy between the two networks that never quite disappears throughout training and, hence, they can't converge to a collapsed representation.

2.6. Graph Contrastive Learning

Contrastive learning on graph data, referred to as **Graph Contrastive Learning (GCL)**, is of particular interest to this thesis. Due to the special properties of graph data, applying CL strategies from other domains to the graph domain is often non-trivial. Extra considerations have to be made to make CL work for graphs. Some of these considerations are discussed here.

2.6.1. Representation Scope

Firstly, when doing representation learning on graphs using GCL, one has to decide whether to learn **node representations** or **graph representations**. Node and graph representations are each useful for their own set of downstream tasks (see 2.2.6). As a consequence of this, one can't directly compare node-level and graph-level GCL methods through downstream evaluation. One could possibly also learn edge-level representations, but this is rarely done in GCL literature.

2.6.2. Contrastive Mode

Secondly, one needs to consider what "**contrastive mode**" to use when doing GCL, which is a term coined by [62]. The contrastive mode determines how learned representations/projections are compared to one another in the contrastive loss. Many GCL methods symmetrically contrast either two node representations, the local-to-local (L2L) contrastive mode [49, 69, 72], or two graph representations, the global-to-global (G2G) mode [6, 58, 67]. Additionally, some methods create positive/negative pairs *across* scopes: e.g. contrasting node-level to graph-level, the local-to-global mode (L2G) [16, 45, 54]. When doing data augmentation to create positive pairs, the used mode also determines what styles of data augmentation are appropriate. E.g. node dropping and subgraph sampling augmentations do not make sense in a node-level setting.

2.6.3. Encoder And Projector Backbone

Lastly, contrastive learning on graph data requires model architectures that can handle graph data for the encoder and, possibly, a projector. In GCL literature, a GCNN is typically the architecture of choice for the encoder and a shallow Multi-Layer Perceptron (MLP) for the projector. GCN [24] and GIN [63] are by far the most popular styles of GCNN employed as the encoder. GCN is typically used for node representation learning and GIN for graph representation learning.

3

Related Work

This chapter consists of three parts. First, we discuss related work on GCNN stability (Section 3.1) and, after that, related work on Graph Contrastive Learning (GCL) (Section 3.2). Lastly, we identify some challenges in the current-day GCL landscape and how the challenge of scaling GNN architectures can be related to GCNN stability (Section 3.3).

3.1. GCNN Stability

Theoretical research on Graph Convolutional Neural Network (GCNN) stability is rather niche and only started after the recent surge of interest in GNNs [24]. Because of this, there is not much work yet in this area. The few works that do exist are discussed here.

In their paper, the authors of [9] theoretically derive stability properties of Graph Convolutional Neural Networks (GCNNs). They prove that GCNNs can simultaneously be stable and discriminative, whereas for linear Graph Convolutional Filters (GCFs) there is always a trade-off between stability and discriminability. Due to the non-linearities present in GCNNs, high-frequency signal can leak into lower frequencies during forward propagation, which makes high-frequency information easier to discriminate without sacrificing stability. Additionally they prove that GCFs and GCNNs are stable to both relative and absolute edge weight perturbation. They derive a theoretical upper bound to the instability of both GCFs and GCNNs, the latter of which is central to this thesis and given in (2.16). Notably, this bound loosens as a GCNN's depth L, width F, or order K increase.

The authors of [11] consider the stability of *stochastic* graph convolutional neural networks: these are GCNNs where edges in the network can drop out at all times, even during forward propagation. An example of a stochastic network in a practical setting is the communication network between drones in a swarm. They show that there is an upper bound to the expected instability of such stochastic GCNNs. This bound is also considered in our work and given in (2.18). In this thesis however, we use it as loose bound for edge dropping perturbation to the input graph. Like the bound in [9], this bound loosens with larger L, F, and K.

The authors of [23] derive an *interpretable* upper bound on the instability of spectral graph filters under random edge dropping and adding. GCFs belong to the family of spectral graph filters and, with that, this upper bound also relates to GCNNs for which GCFs form the backbone. As opposed to the previously mentioned bounds, this bound is interpretable in the sense that it is defined in terms of node drops and additions around particular nodes. For measuring stability, this work utilizes a normalized instability metric of which the one used in this thesis (2.15) is a multi-feature variant.

In [26], it is shown that spectral graph filters – which include GCFs – and spectral graph neural networks – which include GCNNs – give consistent outputs to perturbed input as long as the input graphs discretize the same "continuous graph space". Notably, this transferability applies even to graph pairs of completely different sizes and topologies. This transferability in turn implies that these architectures are stable to structural perturbations, including upscaling and downscaling graphs while preserving global structure.

3.2. Graph Contrastive Learning

This section discusses recent work on Graph Contrastive Learning (GCL). While GCL is the primary strategy employed to do Self-Supervised Learning (SSL) on graphs, there also exists work on non-contrastive methods. Most notably, this includes a family of generative methods using graph autoencoders [19, 20, 27, 37, 47, 48, 53, 70]. The scope of this thesis and, hence, also this section is limited to GCL. In the following sections we discuss various GCL methods, grouped in six different categories for clarity.

3.2.1. Methods Inspired By Deep InfoMax

The earliest set of contrastive methods for graphs were inspired by Deep InfoMax [18] from Computer Vision (CV). The driving principle behind Deep InfoMax is that good representations should maximally share information with the input data. Hence, the training target for the encoder is Mutual Information (MI) maximization, which is achieved by using a contrastive loss with negative samples. The GCL method DGI [54] trains an encoder f to produce node-level representations using a G2L contrastive mode. That is, it maximizes the MI between individual node representations in the output and a mean-pooled graph representation of that same output. For this it trains a simple discriminator on the binary cross-entropy loss to distinguish the true node representations from negative samples. For negative samples it uses node representations of a corrupted graph. This corrupted graph created by shuffling the node representations between nodes.

InfoGraph [45] closely follows DGI but focuses on learning graph representations. And, instead of contrasting in the G2L mode, it maximizes MI between graph-level and "patch"-level representations by concatenating the node features from every layer of the GNN encoder. Due to its focus on graph classification, it can use simple random sampling from the batch to obtain negative samples.

GMI [39] again is similar to DGI but aims to maximize the mutual information between the learned representation of a node and the input features of the k-hop neighborhood around that node. Additionally, it tries to capture edge weight information by weighing those k-hop neighbors using scalars that are in turn trained to share maximal MI with the original edge weights in $\boldsymbol{A}$. Negative samples are obtained by taking nodes outside the k-hop neighborhood of the anchor node.

Lastly, GCC [41] employs a similar strategy but with the goal of learning node representations that encode the *structure* information of the graph. These structural representations are meant to generalize to graphs from unseen datasets. Hence, it does not actually use the graph's input features $\boldsymbol{X}$ as input for f. As for the contrasting strategy, it considers the mean-pool of random walks around the same node as positive pairs and those around different nodes as negative pairs.

3.2.2. Methods Inspired By SimCLR

Following the success of the SimCLR framework [4] in CV, similar methods were developed to be applied to graph data. GraphCL [67] faithfully applies SimCLR-style CL to graph representation learning, using the NT-Xent loss, a variant of InfoNCE (2.22). It introduces the four most commonly used graph augmentation functions: node dropping, node feature masking, edge dropping/adding, and subgraph sampling (see Figure 2.7).

GRACE [72] also closely follows SimCLR, but instead applies it to node representation learning. Contrasting is done between nodes of the same graph instead of graphs sampled from a multi-graph dataset as in GraphCL. To generate positive views, GRACE limits itself to edge dropping and node feature masking.

MVGRL [16] brings back the concept of cross-scope contrasting by pairing up node representations with mean-pooled graph representations of sampled subgraphs. This allows MVGRL to be applied to both graph-level and node-level downstream tasks. To generate a positive samples, it uses a new graph *diffusion* augmentation, which "diffuses" the graph structure encoded in GSO $\boldsymbol{S}$ by taking a weighted sum of different powers of $\boldsymbol{S}$.

3.2.3. Methods Using Self-Distillation Or Redundancy Reduction

In computer vision, BYOL [13] revolutionizes CL by avoiding representational collapse through a self-distillation mechanism instead of negative samples. The GCL method BGRL [49] is the first to apply

this to the graph domain. Similar to GRACE, it focuses on node representation learning and uses edge dropping and node feature masking for creating views of nodes. Notably, in contrast to BYOL, BGRL's architecture does not include a projector as the authors found that it did not empirically improve downstream performance.

SUGRL [34] also adopts the self-distillation strategy but applies it to an unusual, asymmetric architecture where the encoder f is a Multi-Layer Perceptron (MLP) and it is trained with positive samples generated by a target network based on a GNN. The authors empirically show that this allows SUGRL to remove the need for explicit graph augmentations while gaining a speedup in training time without sacrificing downstream performance. They however provide no theoretical motivation for this particular style of asymmetric architecture.

Another self-distillation method, InfoGCL [62], attempts to automize the human-driven process of picking the right graph augmentations, encoder backbone, and contrastive mode for a particular graph dataset. It does this by theoretically deriving a separate optimization problem for each of these three aspects of GCL design.

Lastly, CCA-SSG [69] brings the redundancy reduction paradigm from Barlow Twins [68] to the graph domain, specifically for node representation learning. It generates positive views by augmenting the input graph with edge dropping and feature masking and, like Barlow Twins, prevents representational collapse not through negative samples but by penalizing redundancy, i.e. inter-feature correlation, in the node representations.

3.2.4. Negative Sample Mining Strategies For GCL

Some GCL methods aim to improve on standard methods by reducing the performance impact of false negatives or an overemphasis on easy negative samples. CuCo [6] does this by gradually increasing the "hardness" of negative samples during training. Here, the hardness of a negative sample j is scored by the cosine distance between its projection z_j and the projection of the anchor node z_i.

The node representation learning method ProGCL [60] exploits label information to estimate the probability distributions of true and false negatives conditioned on the cosine similarity with the anchor. They show that, for *graph* data in particular, true negatives and false negatives have significantly different distributions. Hence, by fitting curves to them during training, true and false negatives can be better separated. They demonstrate that putting extra emphasis on high-probability true negatives in the loss leads to improvements in downstream performance. Note that, due to its use of label information, ProGCL is semi-supervised, not fully self-supervised.

HSAN [33], another node representation learning method, aims to avoid false negatives by clustering nodes into pseudo classes. Nodes with the same pseudo label are assumed to belong to the same class and, hence, are not combined into negative pairs. To do the clustering, HSAN calculates similarity between nodes using an unusual dual MLP structural encoder and dual MLP attribute encoder architecture. Similarity is computed with a linear combination of the cosine distances between the outputs of the two structural and attribute encoders, respectively. This asymmetric architecture also allows HSAN to function without any data augmentation. The authors provide no theoretical motivation for this particular architecture.

3.2.5. Smarter Graph Augmentation Strategies

Because good data augmentations are so important for effective CL, many GCL works focus on finding smarter ways to augment graph data. The simple graph data augmentations introduced in [67] often fail to perturb task-irrelevant information without also destroying task-relevant information.

GCA [73], an extension of GRACE, is one of the first attempts at doing smarter augmentation. It makes use of an *adaptive* augmentation method, based on basic edge perturbation and node feature masking. These standard perturbations are made adaptive by performing less perturbation on and around important nodes and more perturbation on and around unimportant nodes. Node importance is determined by considering three deterministic measures of node centrality, building on the assumption that the most central nodes in the graph are the most important.

The authors of [66] propose another smart augmentation method. This one builds on top of GraphCL but, instead of using standard augmentations, it uses two independent generative models to create two views of an input graph. For this, it employs two variational graph auto encoders trained on their own contrastive loss based on the information bottleneck principle.

LG2AR [17] tries to eliminate the labor-intensive step of picking good augmentations for a particular dataset. It does so by training a network that outputs a categorical distribution over five basic augmentation functions, from which one is sampled on each forward pass. Additionally, each specific augmentation is adaptively parameterized with its own neural network. E.g. node dropping is parameterized by a network that gives a categorical distribution over the nodes. These networks are trained jointly with the encoder, but not in an adversarial manner. This last design choice is surprising, given that it allows the trainable augmentor to learn "easy" augmentations to boost the contrastive optimization, effectively bypassing important penalties in the loss.

The remaining smart augmentation methods are grouped subsections for clarity. Each subsection corresponds to one of three categories: (1) augmentations that use the graph spectrum, (2) adversarial augmentations, and, lastly, (3) GCL methods that omit augmentation entirely.

Spectral Graph Augmentations. The authors of [30] introduce a way of enhancing any GCL method by applying a structural graph augmentation that uses the graph spectrum. To motivate their design, they performed a preliminary analysis of how commonly used graph augmentation methods affect the graph spectrum. They found that good augmentations perturb high-frequency signal while preserving lower frequencies. This is likely a consequence of the homophilic property of the graph datasets they used for this experiment. They demonstrate the effectiveness of augmentation under this assumption by implementing an edge dropping/adding scheme that optimally perturbs the graph structure to achieve a maximal change of high-frequency eigenvalues and a minimal change of low-frequency eigenvalues.

SPAN [28] introduces a similar spectral augmentation scheme, but this time built on the principle that an augmentation should maximally perturb *all* frequencies, not just the high ones. Again, this is achieved by finding edge perturbations that optimize for this target. A drawback of both of these methods is that, due to their dependence on the graph spectrum, they are required to do a costly eigendecomposition of graph structure matrix. This can however be done as a one-time precomputation step.

Adversarial Graph Augmentations. AD-GCL [46] aims to learn better node and graph representations by performing adversarial edge dropping. For this, it uses an adversarial GNN, trained to maximize the contrastive loss, that produces a categorical distribution from which edges to drop are sampled. They demonstrate that this technique of maximizing the difficulty of augmentations can boost downstream performance. To prevent the adversarial augmentation from being too disruptive they use a regularization term in the loss.

ArieL [8], a node-level method, uses standard augmentation alongside a style of adversarial augmentation similar to AD-GCL. Ariel's augmentation smartly perturbs both edges and node features to maximize the contrastive loss of a particular positive pair. It prevents unbounded distortion to the graph, not with a penalty in the loss, but by limiting the augmented structure and features of graph to a finite ball around those of the original graph.

Lastly, GASSL [64] limits its adversarial augmentation similarly to ArieL, but focuses solely on perturbing node features. Additionally, the authors also considered a variant that perturbs not the input features on the graph, but the hidden representations produced by the first layer of the encoder. The goal of this is to augment higher-level representations conditioned on the graph structure. They demonstrate the effectiveness of their method on graph classification downstream tasks.

Augmentation-Free Methods. AFGRL [25] does node-level GCL without any augmentation and without any negative samples. It achieves this by adopting a BYOL-style self-distillation architecture and by considering the k-nearest neighbors of the anchor node in a node-to-node similarity matrix as positive samples. As the similarity metric they use the cosine similarity between representations produced by the online and target networks. To aid training, they filter out potential false positives by only considering direct neighbors and by exploiting a clustering algorithm to find dissimilar node pairs.

AF-GCL [56] obtains positive node pairs by taking the n most similar nodes in a k-hop neighborhood around the anchor node. Similarity is again computed using the cosine similarity, this time between the projected representations of different nodes. A core goal of this method is learning good node representations for both homophilic and heterophilic graphs.

A critique towards both of these methods is that they use learned node representations to inform the positiveness and negativeness of a sample. These positives and negatives in turn inform how to

update representations. The authors don't provide strong arguments for why such a cyclic dependency would not cause the algorithm to converge to a poor solution.

3.2.6. Domain-Specific GCL

MolCLR [58] is a GCL method solely focusing on molecular data. They employ a straightforward SimCLR-style approach for graph representation learning but additionally motivate their choices for augmentations by relating them to molecular properties.

The authors of [52] also advocate for the use of more domain-driven augmentations. Their paper delves into flaws in GCL research and, specifically, how domain-agnostic graph augmentations form a bottleneck for GCL performance. They argue that such augmentations often lead to the loss of task-relevant information while insufficiently perturbing task-irrelevant data. They demonstrate that domain-tailored graph augmentations indeed produce better representations than domain-agnostic ones.

3.3. Discussion

In this section, we take a more global look at GCL research. This is done in three parts. Firstly, we discuss the challenge of designing effective graph data augmentations for doing GCL. Then, we cover the surprising observation that randomly initialized GNNs can produce near-state-of-the-art representations in GCL. Lastly, we provide a motivation for the thesis by discussing the oversmoothing problem and subsequently arguing why GCNN stability theory can provide a new perspective on the poor performance of deep GCNNs in GCL.

3.3.1. The Challenge Of Designing Graph Data Augmentations

As is evident from the large number of GCL methods being developed, adapting CL strategies to graphs is a non-trivial endeavour. One reason for this is that graphs are universal data structures and, as a result, graph datasets can cover an extremely wide set of domains. In particular, it is challenging to design good, general-purpose augmentations for graphs. The four basic graph augmentations introduced by GraphCL [67], node dropping, edge perturbation, node feature masking, and subgraph sampling, have seen some success but don't work equally well across datasets. Unlike augmentations for image data, graph data augmentations often lack clear intuitions to justify their design. As a result, they often either destroy too much task-relevant information or perturb too little task-irrelevant information, which can be detrimental to CL performance [4, 40, 50, 59]. For example, in a molecular setting, dropping an atom or a bond can completely change a molecule's properties. The usage of such an augmentation in this setting will produce many false positive pairs and, with that, produce a faulty training signal for the encoder [52].

To combat this, many strategies are being developed for doing smarter augmentation on graphs. These strategies can roughly be divided into three camps: (1) heuristic-driven approaches, possibly informed by the graph spectrum [28, 30, 73], (2) approaches using adversarial training [8, 46, 64], and (3) domain-specific approaches [52, 58].

While (1) and (2) can improve over basic graph augmentations, they remain highly generic and, with that, their potential is limited. This is most clearly demonstrated by the fact that even image data can be represented using graphs and, clearly, these strategies would in no way be able to compete with augmentations specifically designed for images. More specifically, for (1), the augmentation is only as good as the employed heuristic and, typically, the stronger a heuristic, the narrower is its scope. Regarding (2), we argue that good augmentations should not only be maximally disruptive, but should simultaneously preserve task-relevant information. This last quality is not achieved by the adversarial training strategies proposed in [8, 46, 64].

Hence, we believe that the most potential is found in (3), augmentation strategies that are tailored to specific graph domains. That means: special augmentations for molecules, social networks, citation networks, 3D meshes, etc. Such augmentations can be hand-designed or possibly obtained through smarter adversarial strategies. Even though this is not directly relevant to this thesis, we deem this a promising direction for future GCL research in general.

3.3.2. Randomly Initialized Encoders In GCL

As Trivedi et al. point out in [52], *randomly initialized* GCNNs already perform remarkably well in GCL settings. Specifically, they show that GraphCL [67] and InfoGraph [45] struggle to improve over the already impressive downstream performance of randomly initialized encoders on multiple datasets. We also observed this behavior when attempting to reproduce GraphCL [67], CCA-SSG [69], BRGL [49], and GRACE [72] as part of our preliminary research to this thesis. This begs the question: to what extent can the success of GCL methods be attributed to the contrastive pre-training they employ and to what extent to the inductive bias that GCNNs possess? If the GCNN inductive bias is indeed so strong that it can produce good representations on its own, then it may also be sufficiently strong to form an obstacle for improving performance beyond that base level. This is, aside from oversmoothing and the work presented in this thesis, yet another sign that GCNNs may be a suboptimal choice for GCL. So far, little work is done on investigating this phenomenon in GCL literature [52].

3.3.3. Effects Of GCNN Architecture On GCL

Another reason for the difficulty of adapting generic CL methods to the graph domain is that the behavior of a GCNN – the most commonly employed encoder backbone in GCL – is strongly tied to its specific architecture. For many popular GCNNs such as GCN and GIN, nodes can only interact with their direct neighbors during forward propagation of a layer. For a GCNN of L layers, this limits the perceptive field of any particular node to its L-hop neighborhood. At the same time, these models can't be made too deep as overly deep models typically perform worse. Consequently, one has to strike a awkward balance between a small receptive field and an overly deep architecture when using GCNNs. In practice, this balance tips in the favor of shallow GCNNs, as is also the case in most GCL research (see Table 3.1). This reliance on shallow architectures likely forms a bottleneck in the GCL setting as simple architectures will underfit on large unlabeled datasets. The authors of [65] demonstrate with their custom transformer-based architecture that GCNNs indeed leave a lot of headroom when it comes to downstream performance on large-scale datasets.

One possible explanation for this degraded performance of deep GCNNs is "oversmoothing". Smoothing is a behavior that many styles of GCNN possess, where the information over the graph grows more uniform as it is passed through the layers of the GCNN. Smoothing is useful to many tasks when done in moderate amounts, but too much smoothing, referred to as oversmoothing, can hurt performance; oversmoothing can make the representations of semantically dissimilar nodes so similar that they can no longer be distinguished [2, 29, 67].

Oversmoothing provides only one perspective on the potential poor applicability of deep GCNNs to GCL. In this thesis we suggest another perspective, which we derive from theoretical work on the stability properties of GCNNs [9, 11, 23]. These studies provide upper bounds on the instability of GCNNs under certain styles of perturbation. Such stability guarantees are relevant to any machine learning setting that uses GCNNs but are especially relevant to the contrastive learning setting. A crucial part of CL is learning augmentation-invariant representations, i.e. representations that are *stable* to those augmentations [4, 40, 50, 52, 59].

Most notably, the theoretical upper bounds provided in [9, 11] suggest that there is a link between a GCNN's stability and its architecture. The respective upper bounds are dependent on the GCNN's depth, width, and order. As a result, these bounds loosen when a GCNN's architecture grows more complex. This loosening of the bound suggests that GCNNs grow less stable when made more complex. Note however this is merely a suggestion: a looser bound does not necessarily *imply* worse stability. In fact, the considered upper bounds are all rather loose to begin with.

Nonetheless, if complex GCNNs are indeed less stable, then that can negatively affect their ability to learn augmentation-invariance, which in turn can lead to degraded downstream performance in GCL. In that case, this forms an alternative explanation for the poor performance of overly deep GCNNs in GCL. Hence, in this thesis we experimentally investigate how a GCNN's architecture influences its stability and, consequently, its downstream performance in a GCL setting.

Table 3.1: The encoder backbone and encoder depth, denoted L, for the discussed GCL methods that use a GNN as their encoder. Notably, many GCL methods employ rather simple GNN architectures. If the encoder architecture is dependent on whether applied to node-level tasks or graph-level tasks, this is indicated in the table by (N) and (G), respectively.

Method	Backbone	L
DGI [54]	GCN or GraphSAGE-GCN [14]	1 or 3
GMI [39]	GCN	1 or 2
InfoGraph [45]	GIN	4, 8, or 12
GCC [41]	GIN	5
GraphCL [67]	GIN	2
MVGRL [16]	GCN	1, 2, or 4
GRACE [72]	GCN	2
GCA [73]	GCN	2
BGRL [49]	GCN	2 or 3
CuCo [6]	GIN	3
ProGCL [60]	GCN or GraphSAGE-GCN [14]	2 or 3
CCA-SSG [69]	GCN	1 or 2
SPAN [28]	GCN (N); GIN (G)	2 (N); 5 (G)
InfoGCL [62]	GCN, GIN or GAT	1 (N); 2, 4, 8, or 12 (G)
AF-GCL [56]	GCN	2
AFGRL [25]	GCN	1
MolCRL [58]	GCN or GIN	5
AD-GCL [46]	GIN	unknown
You et al. [66]	GIN	2
ArieL [8]	GCN	2
LG2AR [17]	GCN	2
GASSL [64]	GCN or GIN	2

4

Hypotheses And Experimental Method

In this section, we repeat the research questions from the introduction (Section 4.1), describe our hypotheses to those research questions (Section 4.2), and motivate our experimental method based on those hypotheses (Section 4.3).

4.1. Research Questions

To reiterate, the goal of this thesis is to find further explanation as to why deeper GCNN architectures perform worse for contrastive learning. We aim to do this by gaining insights into the interplay between *untrained* stability of GCNNs and *trained* stability through contrastive pre-training. The research questions we formulated for this were:

RQ1. When a GCNN is pre-trained in a contrastive manner with increasing amounts of augmentation, does its stability to that augmentation improve?

RQ2. When varying a GCNN's architecture, does its stability change in accordance to what theoretical stability bounds suggest?

RQ3. How does a GCNN's *untrained stability* as induced by its specific architecture interact with its *trained stability*, acquired through contrastive pre-training?

RQ4. After being pre-trained contrastively using augmentation, is the downstream performance of GCNNs correlated to their stability to that augmentation?

4.2. Hypotheses

For **RQ1**, we formulate the following hypothesis:

H.1.1. GCNNs trained in a contrastive manner with stronger augmentation are more stable to said augmentations than those trained with weaker augmentation.

This is motivated by the fact that practically every contrastive loss function contains a term that encourages the encoder f, potentially combined with a projector g, to become stable to augmentations (see Section 2.5). E.g. in the InfoNCE loss, this is the similarity term $\text{sim}(\boldsymbol{h}, \boldsymbol{h}^+)$ in the enumerator. We aim to test **H.1.1** as follows. We pre-train a set of GCNNs using a standard CL method, while varying the augmentation strength. We then measure how the stability of these GCNNs changes as the augmentation strength changes.

RQ2 is motivated by the fact that the theoretical stability guarantees for GCNNs grow weaker as their complexity, i.e. any one of the hyper parameters L, F, and K, grows. This is the case under specific types of perturbation: edge weight perturbation and edge dropping (see 2.3.3). This suggests that more

complex GCNNs are less stable than simpler ones. Note however that this is merely a suggestion: a weaker bound does not necessarily imply that the respective GCNN is indeed less stable. An additional caveat is that the bounds only apply to GCNNs that are Integral Lipschitz (IL). Being IL puts constraints on the weights that a GCNN can attain (see 2.2.4). These constraints are not present real-world training regimes. Nonetheless, in line with the intuitions derived from the theory, we formulate the following hypothesis:

H.2.1. Having a larger depth L, width F, or order K all contribute to a GCNN's instability. Thus, more complex GCNNs are less stable than a less complex ones.

We aim to test **H.2.1** by instantiating GCNNs with varying L, F, and K and then evaluating their stability on a number of graph datasets. To eliminate the effect of any particular training regime on stability, we do not train the GCNNs at all. Their weights are simply randomly initialized.

RQ3 builds on the assumption that both **H.1.1** and **H.2.1** hold. If they do, then GCNNs both possess *untrained* stability properties as a result of their architecture and the ability to acquire *trained* stability through pre-training in a contrastive manner. The next question is then: how these two types of stability interact? We don't have a strong hypothesis on what happens in this case as the GCNN stability theory [9, 11] does not give clear insights on how network weights obtained through a particular training regime can affect GCNN stability. The only term in the considered stability bounds [9, 11] that depends on network weights is the IL constant C_{IL} (see 2.3.2).

On the one hand, one can argue that untrained and trained stability should be expected to have a multiplicative effect, given that performing contrastive pre-training would only affect the IL constant C_{IL} and nothing else about the theoretical bound. If contrastive pre-training a GCNN manages to successfully reduce its C_{IL}, then that reduced amount would still have to be multiplied with terms containing L, F, and K. The contrastive pre-training would effectively work as a scalar to the already present stability properties of the GCNN as induced by architecture. This motivates our first hypothesis for **RQ3**:

H.3.1. Trained stability behaves as a scalar to the untrained stability properties of a GCNN and, as a result, cannot fully negate instability of complex GCNN architectures.

On the other hand, both theoretical bounds are loose bounds. It could be possible that complex GCNNs are be capable of learning weights that make them just as stable as their simpler counterparts. A complex GCNN architecture is namely, to some extent, a superset of simpler architectures. It could e.g. learn dummy weights to artificially reduce its architectural complexity and, with that, increase its stability. If this is indeed possible, then the contrastive training could, theoretically, negate the effects on stability associated with architectural complexity. This motivates our second hypothesis for **RQ3**:

H.3.2. Trained stability overrides the effects of untrained stability properties and, as a result, complex GCNNs can become just as stable as simpler architectures through contrastive training .

We aim to investigate these **H.3.1** and **H.3.2** by, again, creating a set of GCNN models with varying L, F, and K. This time, however, we do not only consider untrained models. Additionally we consider GCNNs trained using CL with varying augmentation strengths.

The last research question, **RQ4**, is motivated by the fact that learning augmentation-invariant representations is a crucial part of contrastive learning. If the augmentations are indeed appropriate for the respective downstream task, i.e. they maximally perturb task-irrelevant information while preserving task-relevant information, then representations learned from those augmentations produce good downstream performance. Thus, when doing GCL, an increase in a GCNN's stability to the augmentations should positively affect its downstream performance, on the condition that those augmentations are appropriate. Hence, we formulate the following hypothesis for **RQ4**:

H.4.1. When doing GCL, there is a positive correlation between a GCNN's stability to the augmentations and its downstream performance, on the condition that those augmentations are appropriate for the respective downstream task.

We test this hypothesis by considering contrastively trained GCNNs with varying L, F, and K. Subsequently, we assess whether or not there is a positive correlation between their stability and their downstream performance.

4.3. Method

This section gives an overview of the experimental setting used to answer the research questions. This concerns the used contrastive learning method, perturbation/augmentation functions, stability metric, and datasets.

4.3.1. Graph Contrastive Learning With GRACE

Many of our experiments involve performing CL on GCNNs to observe how such training affects their stability. The particular contrastive learning method we use for this is GRACE [72]. This is a simple SimCLR-style contrastive method for graphs suited to node-level downstream tasks such as node regression and node classification. It specifically focuses on transductive tasks where the dataset contains only a single graph. This aligns with the relevant GCNN stability literature [9, 11], in which a single-graph setting is also assumed.

GRACE by default uses GCN [24] as its encoder. See (2.12) for a definition of GCN. In our experiments we swap GCN out for the more expressive GCNN, which is necessary to be able to vary the order K. We do however still use the same graph shift operator $\boldsymbol{S}$ as in GCN: the degree-normalized adjacency matrix, given by $\bar{\boldsymbol{D}}^{-1/2}\bar{\boldsymbol{A}}\bar{\boldsymbol{D}}^{-1/2}$.

GRACE also includes a projector $\boldsymbol{z}_i = g(\boldsymbol{h}_i)$ which is a 2-layer MLP applied to each individual node representation $\boldsymbol{h}_i \in \boldsymbol{H} = f(\boldsymbol{X}, \boldsymbol{A})$. These projections are then fed into a symmetric variant of the InfoNCE loss, contrasting node feature pairs:

$$\begin{aligned}
\mathcal{L}_{\text{GRACE}} &= \frac{1}{2}\sum_{i=1}^{N} \mathcal{L}_{\text{GRACE},i}(\widetilde{\boldsymbol{z}}_i^{(1)}, \widetilde{\boldsymbol{z}}_i^{(2)}) + \mathcal{L}_{\text{GRACE},i}(\widetilde{\boldsymbol{z}}_i^{(2)}, \widetilde{\boldsymbol{z}}_i^{(1)}), \\
\mathcal{L}_{\text{GRACE},i}(\widetilde{\boldsymbol{z}}_i^{(1)}, \widetilde{\boldsymbol{z}}_i^{(2)}) &= -\log \frac{\text{sim}\left(\widetilde{\boldsymbol{z}}_i^{(1)}, \widetilde{\boldsymbol{z}}_i^{(2)}\right)}{\text{sim}\left(\widetilde{\boldsymbol{z}}_i^{(1)}, \widetilde{\boldsymbol{z}}_i^{(2)}\right) + \sum_{k=1}^{N} \text{sim}\left(\widetilde{\boldsymbol{z}}_i^{(1)}, \widetilde{\boldsymbol{z}}_k^{(2)}\right) + \sum_{k=1}^{N} \text{sim}\left(\widetilde{\boldsymbol{z}}_i^{(1)}, \widetilde{\boldsymbol{z}}_k^{(1)}\right)}, \\
&\text{where } \text{sim}(\boldsymbol{z}_1, \boldsymbol{z}_2) = e^{\boldsymbol{z}_1^\intercal \boldsymbol{z}_2 / \tau}.
\end{aligned} \tag{4.1}$$

Here, $\widetilde{\boldsymbol{z}}_i^{(1)}$ and $\widetilde{\boldsymbol{z}}_i^{(2)}$ denote the two projections of node i corresponding to the two augmentation paths of SimCLR-style contrastive learning. That is, they are obtained by sequentially applying f and g on the perturbed graphs $\widetilde{\boldsymbol{S}}^{(1)}$ and $\widetilde{\boldsymbol{S}}^{(2)}$, respectively. The constant τ is the temperature hyperparameter.

4.3.2. Perturbation

We perform two types of perturbation: edge weight perturbation and random edge dropping, each respectively belonging to one of the considered stability bounds, namely (2.16) and (2.18).

1. For **Edge Weight Perturbation (EWP)**, we use the same stochastic perturbation function t_{EWP} as the synthetic experiment in [9]:

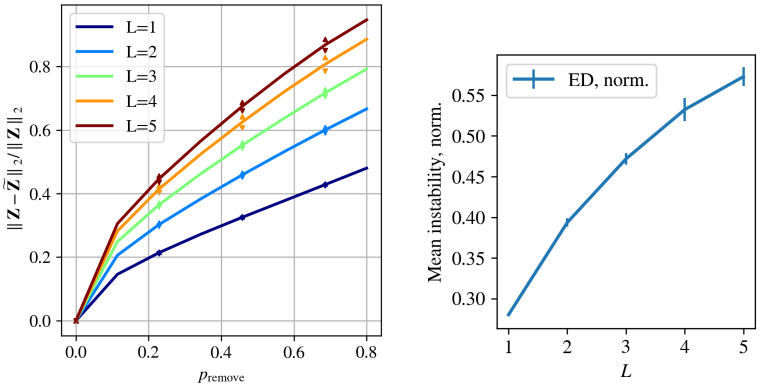
$$\widetilde{\boldsymbol{S}} = t_{\text{EWP}}(\boldsymbol{S}) = \boldsymbol{S} + \boldsymbol{S}\boldsymbol{E} + \boldsymbol{E}\boldsymbol{S} \tag{4.2}$$

   Here, $\boldsymbol{E}$ is a diagonal matrix constructed by sampling its diagonal entries from a uniform distribution in the range $[(1-\epsilon)\epsilon, \epsilon]$ [10]. This perturbation scheme ensures that $\|\boldsymbol{E}\|_{\text{op}} \leq \epsilon$ and allows us to use $\Delta = 2C_{\text{IL}}$ in the stability bound (2.16). In our results we use $\epsilon \in [0, 1]$ as an indication of perturbation strength.

2. For **Edge Dropping (ED)**, we use the same style of uniform edge dropping t_{ED} as is used in the original paper [11]:

$$\widetilde{\boldsymbol{S}} = t_{\text{ED}}(\boldsymbol{S}) = \boldsymbol{S} \odot \boldsymbol{B} \quad \text{where } [\boldsymbol{B}]_{ij} \sim \text{Bernoulli}(p_{\text{remove}})\ \forall ij. \tag{4.3}$$

   Here, $\odot$ denotes an entry-wise product between $\boldsymbol{S}$ and $\boldsymbol{B}$. For ED we use $p_{\text{remove}} \in [0, 1]$ as an indication of perturbation strength.



(a) The "raw" instability values.

(b) The mean instability values over perturbation intensities.

Figure 4.1: Both left and right plots visualize the same underlying data: normalized instability of untrained GCNNs on CiteSeer under ED while varying depth L. The right plot, though, collapses the values over the ED intensity dimension, p_{remove}, into scalar values by taking the mean over the instability measurements.

4.3.3. Measuring Stability

To measure stability or, more specifically, *instability*, we consider both a normalized and an absolute instability metric. These are given in (2.15) and (2.14), respectively. We will predominantly use the normalized metric as it allows for comparing models across datasets and training settings. However, we also consider absolute instability given that the two stability bounds, (2.16) and (2.18), are defined only for absolute instability.

To do a complete analysis of GCNN stability under the two respective perturbations, we evaluate it over a range of perturbation strengths. Specifically, for EWP, we consider values for ϵ in the full range of $[0, 1]$ and, for ED, we consider values for p_{remove} in the range $[0, 0.8]$. The somewhat arbitrary upper bound to p_{remove} is motivated by the fact that values larger than 0.8 would remove unreasonable amounts of the graph structure.

This style of stability evaluation can be visualized using plots as seen in Figure 4.1a, giving a full overview of how GCNN instability changes over the range of perturbation intensities. Though, for visual clarity and given the unimportance of the exact curves to our conclusions, we opt to instead use a **mean instability** metric for the plots in the results section (chapter 5). Here, the mean is taken of the instability measurements over all perturbation intensities. An example of such a plot is given in Figure 4.1b. For completeness, full instability plots of all experimental results are given in Appendix A.

4.3.4. Datasets

The experiments are done on four datasets: *Cora*, *CiteSeer* [43], a custom node regression dataset based on the *MovieLens-1M* dataset [15], and a custom anomaly detection dataset. Cora and CiteSeer are included because they are real-world datasets, commonly used in GCL performance evaluation. The MovieLens dataset is included because, unlike the other datasets, it is a weighted graph. We should include at least one such dataset in our tests given that we are evaluating the effects of EWP (4.2). Lastly, the "AnomalyDetection" dataset is specifically designed to highly resistant to ED (4.3) and thus is a good benchmark for downstream performance under ED augmentation. Some general statistics of all four datasets are shown in Table 4.1 and further details on each are given below:

- **Cora** and **CiteSeer** are both citation networks in which nodes represent scientific publications and links represent references by one publication to another. The input features are 0/1 vectors denoting the presence of words from a predefined dictionary in the respective article. These datasets don't have edge weights so for these experiments all edge weights are set to 1.
- The **MovieLens-1M** dataset is a collection of 1 million discrete 1-to-5 ratings from 6000 users on 4000 movies. The absence of a rating by a user for a movie is encoded as a zero. Loosely inspired by [9], we construct a custom node regression task from this in which nodes represent users and the node features are the respective user's ratings for various movies. The goal is to

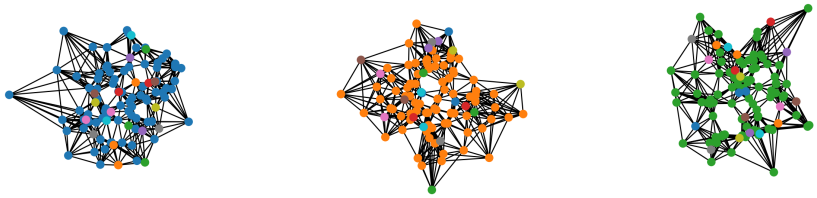


Figure 4.2: Three out of the ten communities in AnomalyDetection. Each community is associated with a distinct color. Most of its member nodes have that color. The anomaly nodes have colors belonging to different communities.

Table 4.1: Statistics on the used datasets. The "#Edges" metric refers to the number of *directed* edges. Here, undirected edges are represented by two directed edges with opposing directions. The "#Targets" column denotes the number of classes for the classification datasets and the regression target dimension for MovieLens. The "Weighted" column indicates whether the dataset includes edge weights. The "Task" column indicates whether the dataset's respective task is classification (C) or regression (R).

Dataset	#Nodes	#Edges	#Features	#Targets	Weighted	Task
Cora	2708	5429	1433	7	✗	C
CiteSeer	3327	4732	3703	6	✗	C
MovieLens	2991	51524	911	1	✓	R
AnomalyDetection	1000	12544	10	2	✗	C

predict the user ratings for the movie *Star Wars*. Only users that actually rated Star Wars are included and we only consider the most densely rated 25% of the movies rated by those users, excluding Star Wars itself. Using the Pearson correlation between node features, we generate an undirected graph structure using the 10-nearest neighbours algorithm. The edge weights are set to the respective Pearson correlation scores.

- The **AnomalyDetection** dataset consists of 10 disjoint communities of 100 nodes. Each community is associated with a distinct color and contains a large majority of "regular" nodes (82%) that have the community's color. The remaining nodes, the "anomaly" nodes, have different colors (see Figure 4.2). The task is to detect the anomaly nodes among regular nodes. Each community was generated by sampling node locations from a 2D normal distribution and subsequently using the 10-nearest neighbor algorithm to build the graph. This intentional abundance of edges allows for the following simple, yet effective classification strategy: classify a node as an anomaly if and only if it has a different color than most of its neighbors. Thanks to the importance of *relative* neighbor information as opposed to *absolute* neighbor counts, node semantics are mostly unaffected by edge dropping. This is verified by the fact that a procedural 2-hop neighbor vote algorithm maintains a $> 95\%$ classification accuracy, even under ED with $p_{\text{remove}} = 0.75$.

4.3.5. Hyperparameters

For Cora and CiteSeer, we use the default hyperparameters given by GRACE. These may however be suboptimal given that we use GCNN as the encoder backbone and not GCN. For MovieLens and AnomalyDetection, we use a set of hyperparameters that provide reasonably good downstream results on standard GRACE. Ultimately, it is not of crucial importance to our experiments that the hyperparameters are optimal for the respective dataset as our experiments focus on relative changes to stability. The exact hyperparameters for each dataset can be found in Table 4.2.

Table 4.2: The hyperparameters used for each dataset during experiments. Here, L, denotes the GCNN encoder's depth, F_{hidden}, F_{rep}, and F_{proj} denote its hidden, representation, and projection feature dimensionalities, σ is the activation, τ is the temperature parameter, and "lr" denotes the learning rate. Note that, in the GRACE architecture, F_{hidden} is always twice F_{rep}.

Dataset	L	F_{hidden}	F_{rep}	F_{proj}	K	σ	τ	lr
Cora	2	256	128	128	1	ReLU	0.4	$5 \cdot 10^{-4}$
CiteSeer	2	512	256	256	1	PReLU	0.9	$1 \cdot 10^{-3}$
MovieLens	2	64	32	8	1	ReLU	0.1	$5 \cdot 10^{-4}$
AnomalyDecection	2	512	256	32	1	ReLU	0.5	$5 \cdot 10^{-3}$

5 Results

This chapter contains the experimental results that, ultimately, lead to our answers for the research questions. It is divided into five sections: one for each of the four research questions (Section 5.1, 5.2, 5.3, and 5.4) and, lastly, a section that globally discusses the results (Section 5.5).

5.1. Experimental Results For RQ1

The goal of this first set of experiments is to either corroborate or contradict **H.1.1** as an answer to **RQ1**. We do so by training a set of GCNN encoders with GRACE, while varying the strength of the augmentation they are trained with. As data augmentations, we consider both Edge Weight Perturbation (EWP) and Edge Dropping (ED) since these are the perturbations used in the relevant graph stability theory. For both augmentation styles, instability is measured under the respective perturbation style. For significance of the results, each individual experiment is repeated on eight times, i.e. eight repeats of each training setting.

The results are summarized in Figure 5.1. These show the mean normalized instability (2.15) of pre-trained GRACE encoders on the vertical axis against the augmentation strength on the horizontal axis. A more precise explanation and a motivation for using this particular instability metric is given in 4.3.3. The full stability plots can be found in Figure A.1.

5.1.1. Only Under ED GCNNs Become More Stable.

The hypothesis **H.1.1** states that GCNNs should become more stable to perturbation when pre-trained with more stronger augmentation. Clearly, this is the case for ED but not for EWP. For ED, instability to the respective augmentation decreases as the models are trained with higher p_{remove}. For EWP, however, the instability remains constant, only appearing to slightly decrease for a few datasets. The weak effect of training using EWP augmentation can be explained by the fact that instability to this style of perturbation is extremely low to begin with. If GCNNs are already so stable to EWP, then constrastive training may struggle to improve over that. Notably, for EWP, there is also no significant difference between the weighted graph dataset, MovieLens, and the the other, unweighted datasets.

Due to **H.1.1** only holding for edge dropping, only this style perturbation will be considered in the later experiments aimed at answering **RQ3** and **RQ4**.

5.2. Experimental Results For RQ2

This set of experiments aims to answer **RQ2** by either corroborating or contradicting **H.2.1**. The hypothesis states that more complex GCNNs do not only have looser upper bounds to instability, but they are also less stable than simpler GCNNs in practice. To test this, we measure the stability of GCNNs under EWP and ED while varying either one of the following architectural hyperparameters: the GCNN's depth L, its width F, and its order K. Note that the width F here refers to F_{rep}, which is the dimensionality of the output representations. The actual encoder width, F_{hidden}, is two times F_{rep}.

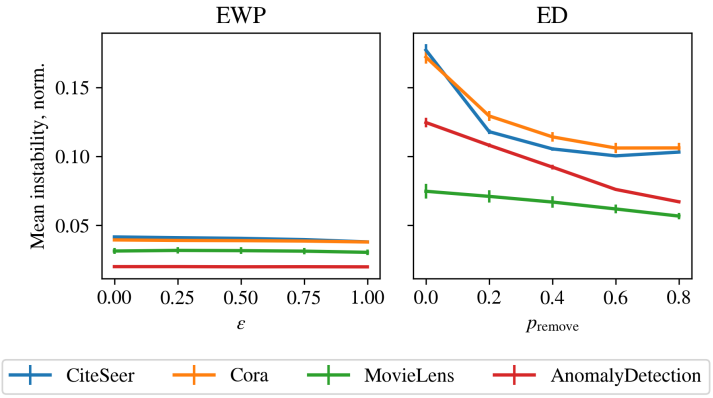


Figure 5.1: The mean normalized instability (2.15), plotted on the vertical axis, of GCNNs trained with increasing amounts of augmentation, plotted on the horizontal axis. The left plot shows this for EWP and the right plot for ED. In the plots, each line corresponds to one of the four datasets. The standard deviation over eight repeats of each setting are visualized using error bars. For EWP, the models do not become significantly more stable as they are trained with increasingly stronger EWP. For ED, they do become more stable.

Table 5.1: The base values for L, F, and K used for the three datasets.

Dataset	L	F	K
Cora	2	128	1
CiteSeer	2	256	1
MovieLens	2	32	1
AnomalyDetection	2	256	1

Only one of these architectural hyperparameters is varied at a time. The other two are kept at a base value. The base values used for each dataset are shown in Table 5.1. These base values correspond to the hyperparameters settings in which the other experiments were run (see Table 4.2).

As was shown in the previous set of experiments, a particular training regime can influence the stability of a GCNN. Thus, to eliminate the effect of a GCNN's training on its stability, we only consider *untrained* GCNNs in these experiments.

The results of the experiments are summarized in Figure 5.2 in a similar style to the previous set of experiments. Here, however, the horizontal axis denotes variation in architecture, i.e. variation in either L, F, or K. In addition to normalized instability, the effects of architecture on *absolute* stability is also given. This is relevant, firstly, because the considered stability bounds, (2.16) and (2.18), are defined only for absolute instability and, secondly, because there are notable discrepancies between the normalized and absolute results.

5.2.1. The Impact Of Normalization When Varying F

In most cases, varying the architecture has a significant impact on GCNN stability. The exception to this is varying F, when measuring *normalized* instability. Across datasets and for both EWP and ED, normalized instability stays more or less constant as F changes. At the same time, when considering *absolute* instability, we found it consistently grows as F increases. This discrepancy between the two metrics makes sense when considering that calculating the absolute instability metric (2.14) involves a sum over the output feature dimension $F_{\text{out}} = F$. Naturally, when F_{out} grows, the absolute instability values will grow too. For calculating the normalized instability measure (2.15), this sum over F_{out} terms is divided out and, presumably, because of that, the stability differences between GCNNs disappear. This suggests that the dependence of GCNN stability on F in the theoretical bounds can be explained by the fact that these bounds are defined for absolute instability and not so much by the fact that wider GCNNs are inherently less stable.

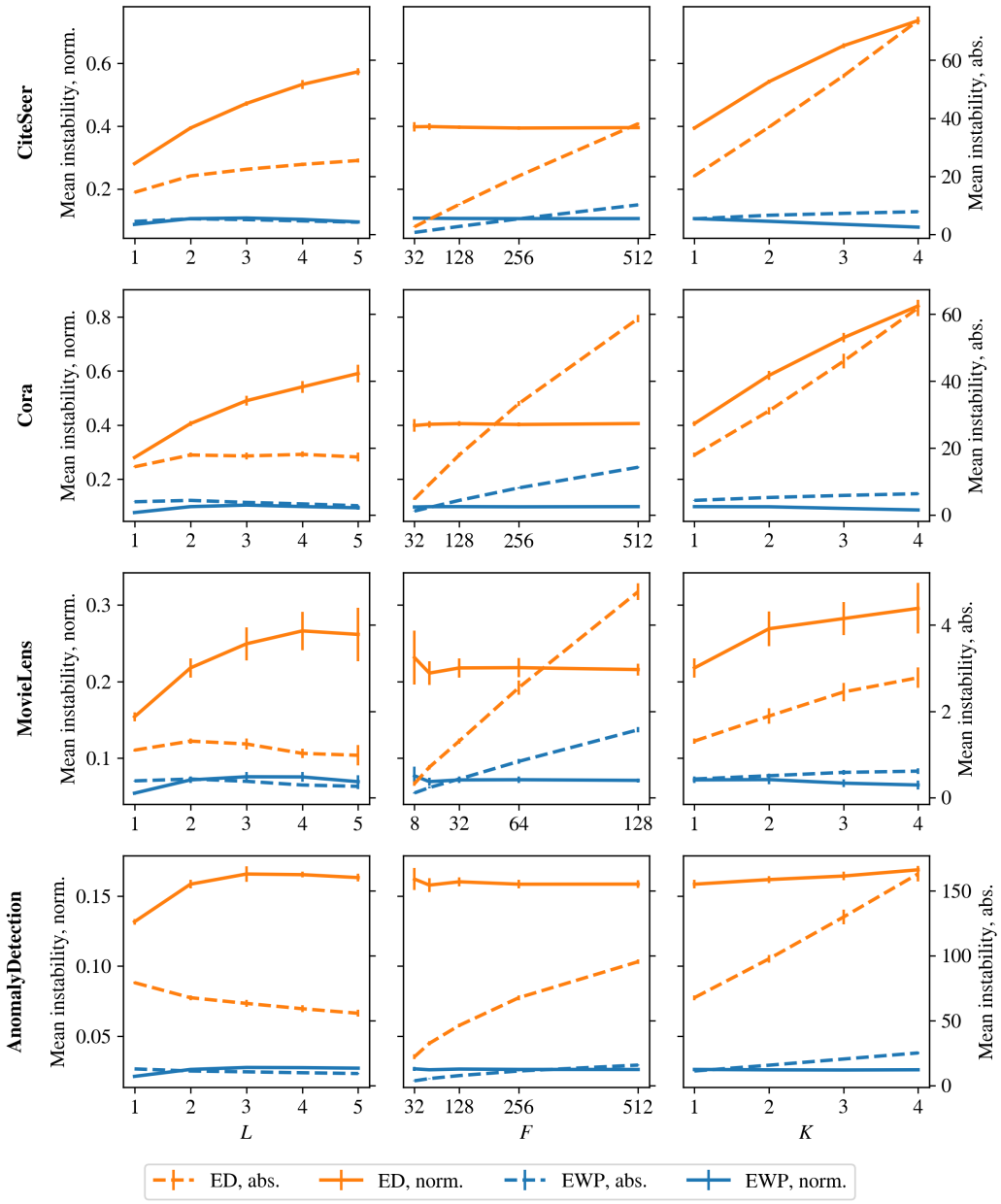


Figure 5.2: The mean instability, plotted on the vertical axis, of GCNNs with varying L, F, or K, plotted on the horizontal axis. Normalized instability is plotted using solid lines and is quantified on the left. Absolute instability is plotted using dashed lines and is qualified on the right. The plots contain lines for both EWP and ED. Note that, across datasets, normalized instability is unaffected by varying F, as is illustrated by the horizontal, solid lines in the middle column. This suggests that changing F does not in fact affect the stability of individual output features. At the same time, for ED, normalized stability consistently grows as either L or K grows. The results for EWP, however, are weak overall and do not show a confident relation between architecture and stability in most cases.

5.2.2. Inconsistency Between Absolute And Normalized Metrics

In general, the results between absolute and normalized instability are often inconsistent with one another, not just when varying F. This discrepancy is explained by the fact that the term in the denominator in (2.15) is not constant as the GCNN architecture is varied. This term describes the magnitude of the unperturbed output signal. As a result, the absolute instability effectively represents a mix of two phenomena: (1) GCNN stability changes as architecture is varied and (2) the output signal's magnitude changes as architecture is varied. In contrast, the normalized instability solely measures (1). We therefore deem the normalized instability to be the most informative metric out of the two, especially in experiments where GCNN architecture is varied. Though, for the sake of completeness, we still included absolute instability in the plots for this experiment.

5.2.3. Comparison To Theoretical Intuitions

In many experiments, GCNN stability changes in the way that the theoretical bounds would suggest: more complex architectures are less stable. For example, normalized instability consistently rises with L or K and absolute instability consistently rises with F or K. There are clearly also exceptions. Most notably, for Cora and MovieLens, the absolute instability of the GCNNs does not follow a clear trend when L is varied. For AnomalyDetection in the same setting, there is even a pattern inverse to our theoretical intuitions: the deeper GCNNs are *more* stable than their shallower counterparts. This inverse pattern occurs as well, to a lesser extent, for normalized instability when varying K under EWP on CiteSeer, Cora, and MovieLens. That said, for EWP, changes to GCNN stability induced by architecture are generally much weaker and less consistent than for ED.

Overall, we can conclude that, for ED, normalized instability grows as L and K increase. This is in line with what the theory suggests and thus supports our hypothesis. Varying F however does not truly affect stability, unlike what theory would suggest, since it does not affect normalized instability. The respective changes in absolute instability can be explained as an artifact of how absolute distances between GCNN outputs are calculated. For EWP, no confident conclusions can be drawn as to how GCNN architecture affects stability, since the results are too inconsistent. Again, this may be due to EWP with $\epsilon \in [0, 1]$ being too weak of a perturbation.

5.3. Experimental Results For RQ3

From the previous two sets of experiments we concluded that, at least for ED, (1) GCNNs are indeed capable of improving their stability through contrastive pre-training and (2) deeper and higher order GCNNs are indeed inherently less stable. The third research question, **RQ3**, concerns the interaction between these two types of stability.

We formulated two hypotheses for this research question: either contrastive training uniformly scales down the instability of GCNNs across architectures (**H.3.1**) or the training overrides the effects of architecture and, hence, achieves good stability even for complex GCNNs (**H.3.2**). To investigate these possible outcomes, we vary the architecture just as in Section 5.2, but this time on GCNN models that are trained in a contrastive regime. We consider three "training states" for the GCNN models: (1) the untrained setting, (2) a setting in which models are trained with GRACE without any augmentation and (3) a setting in which they are trained with a high level of augmentation ($p_{\text{remove}} = 0.8$). We include the latter two to gain insights in how the enumerator and the denominator of the loss (4.1) individually impact stability. When training *without* any augmentation, the contrastive loss simply maximizes the distance between representations of different nodes. Training *with* augmentation additionally encourages the model produce representations that are invariant to the augmentations. These two settings may affect stability in different ways.

Since the results in Section 5.1 show that contrastive training using EWP does not make GCNNs significantly more stable to this style of perturbation, only ED is considered in this set of experiments. The results of these experiments are summarized in Figure 5.3.

5.3.1. GCL Without Augmentation Already Improves Stability

In line with intuitions gained from the first set of experiments, GCNN instability progressively goes down when moving from the untrained setting, to the trained setting without augmentation, and, lastly, to the

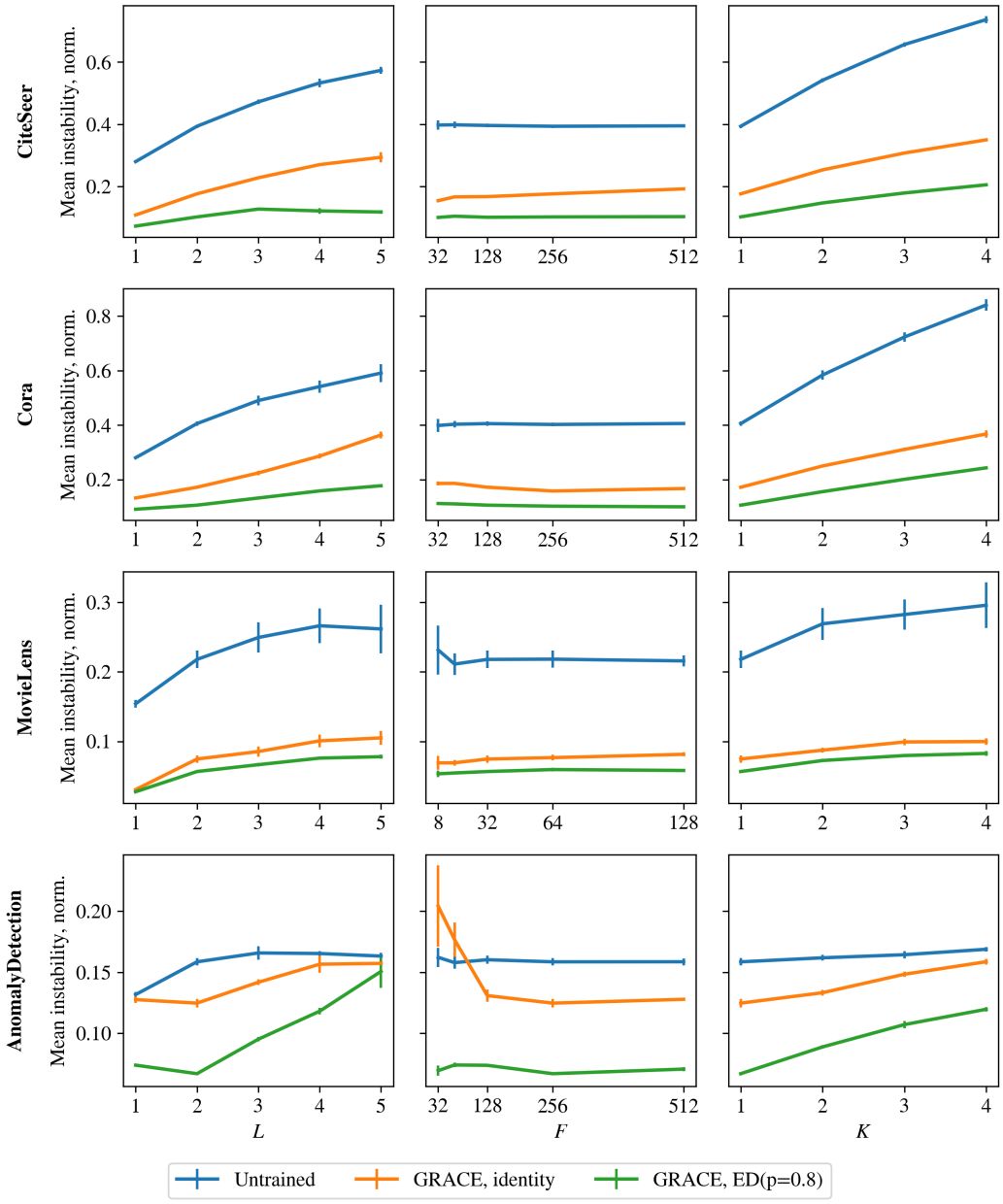


Figure 5.3: The mean normalized instability, plotted on the vertical axis, of GCNNs with varying L, F, or K, plotted on the horizontal axis. Each plot contains three lines corresponding to one of three training settings: (1) untrained, (2) trained using GRACE but without augmentation (3) trained using GRACE with augmentation. In almost every case the GCNNs become more stable as we progress through these training settings. As in the previous set of experiments, F does typically not affect stability. For CiteSeer, Cora, and MovieLens, the respective curves move downwards but keep more or less the same shape: relative distances between architectures are preserved. For AnomalyDetection, in the cases of varying L and K, the differences between architectures become *more* pronounced. Thus, on all datasets, but especially AnomalyDetection, stability remains strongly tied to the architectural hyperparameters L and K, even after contrastive training.

trained setting with augmentation. This is consistent across all combinations of dataset and architecture, with the exception of the AnomalyDetection dataset with small F.

Notably, for all datasets except AnomalyDetection, the largest gain in stability is achieved simply by applying the contrastive loss without any augmentations. This is remarkable because, in this setting, the contrastive loss pushes all node representations to become maximally dissimilar, even those of nodes that have similar input features and neighborhoods. Thus, it would be reasonable to expect that this causes representations to become *less* stable as a result of training, not more so. Nevertheless, the results of our experiments do not support this intuition.

For AnomalyDetection, the largest improvement in stability across architectures comes from adding augmentation to the training. This is possibly explained by the fact that this dataset is specifically designed around ED, and hence encoders can easily achieve good stability to ED on this dataset.

5.3.2. Trained Stability Remains Affected By Architecture

For CiteSeer, Cora, and MovieLens, GCNN instability is effectively scaled down uniformly across architectures, when progressively moving through the training settings. Relative differences between architectures that were present in the untrained setting, are mostly preserved after training, in both the augmented and non-augmented settings. Thus, the experiments on these datasets support our first hypothesis **H.3.1**.

For AnomalyDetection, GCNN stability also improves across the board when moving through the training settings, but, here, the stability differences between architectures progressively grow *more* pronounced. In the case of K, stability differences uniformly grow larger, illustrated by a steeper curve. In the case of L, the respective curve in Figure 5.3 even changes shape, shifting its optimal point to $L = 2$. The results for AnomalyDetection thus do not align with either **H.3.1** or **H.3.2**.

The most important finding here is that **H.3.2** does not hold in any setting. A contrastive training regime is, in fact, *not* capable of negating inherent differences in GCNN stability as induced architecture. Thus, even though a GCNN with high L or K has more parameters than a simpler GCNN, the more complex GCNN encoder will nonetheless be worse at learning augmentation-invariant representations. This cannot be a result of overfitting: evaluation and training are done on the same data and on the same style of perturbation. We thus have to conclude that overly complex GCNNs *underfit* on the contrastive pretext task as a result of being overly complex.

5.4. Experimental Results For RQ4

The final research question, **RQ4**, concerns the relation between GCNN stability and downstream performance. We expect that representations with good stability to a particular augmentation produce good downstream performance, on the condition that the respective augmentation is appropriate for the downstream task (**H.4.1**). We experimentally verify this by evaluating the downstream performance of GCNNs trained using GRACE with strong ED augmentation ($p_{\text{remove}} = 0.8$). Downstream performance is evaluated by fitting a simple linear model to the representations. Specifically, for CiteSeer, Cora, and AnomalyDetection, we evaluate an optimal logistic classifier using the F1 micro-averaged score and, for MovieLens, we evaluate an optimal linear regressor using the negative Mean-Squared Error (MSE). The results are summarized in Figure 5.4.

5.4.1. Negative Correlation Only Present For AnomalyDetection

As per our hypothesis **H.4.1**, we expect a positive correlation between GCNN stability and downstream performance, i.e. a negative correlation between *instability* and performance, on datasets for which ED with $p_{\text{remove}} = 0.8$ is an appropriate augmentation. We find that our experimental results support this hypothesis. Most notably, in Figure 5.4, on the AnomalyDetection dataset, this negative correlation is clearly present for L (left) and K (right). Performance goes down when instability goes up and vice versa. On the other datasets, this negative correlation is typically not visible. In many cases, there is even a positive correlation, indicating that a weaker invariance to the augmentation actually benefits downstream performance.

The presence of this correlation on AnomalyDetection and not on the other datasets can be explained by the fact that, for those other datasets, ED with $p_{\text{remove}} = 0.8$ is not an appropriate augmenta-

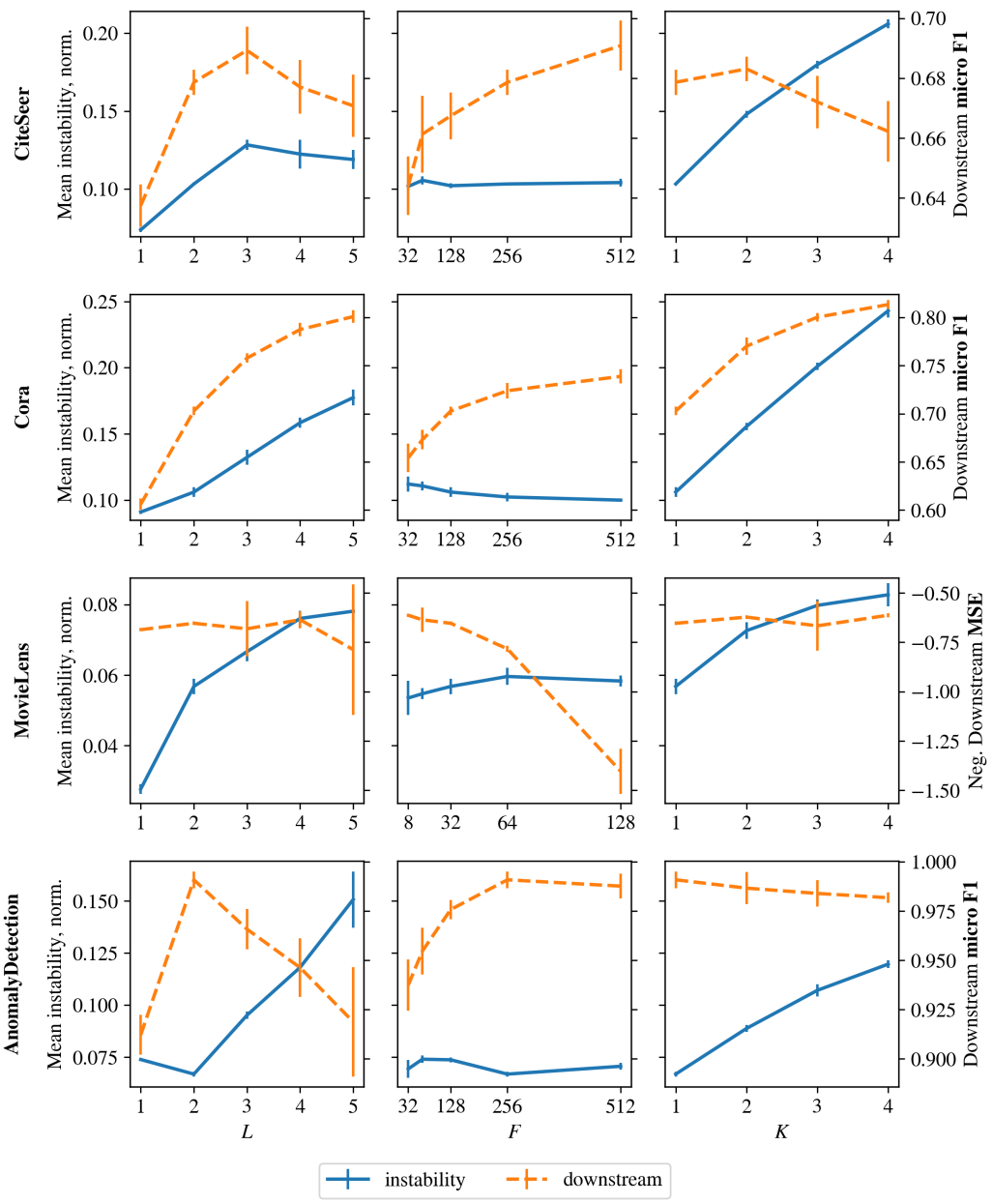


Figure 5.4: The mean normalized instability plotted as solid lines together with downstream performance as dashed lines on the vertical axis over varying GCNN architectures on the horizontal axis. Instability and downstream performance are quantified on the left and right, respectively. For CiteSeer, Cora, and AnomalyDetection downstream performance is measured in the micro-averaged F1 score and for MovieLens in negative Mean Squared Error (MSE). For both of these metrics, higher is better. On the AnomalyDetection dataset for L and K (bottom row, left and right columns), one can see a negative correlation between instability and downstream performance. This suggests a causal relationship between the two, which is supported by theory: representations with better stability to appropriate augmentations perform better on downstream tasks. On other datasets and for F, no such negative correlation is consistently present.

tion, where, for AnomalyDetection, it is. We verified in a preliminary experiment that the node semantics of AnomalyDetection are largely unaffected by edge dropping (see 4.3.4). Under the assumption that ED is indeed unfit for all datasets but AnomalyDetection, all results align with our hypothesis. Though, for CiteSeer, Cora, and MovieLens they only do so weakly.

5.4.2. Effect Of Varying F On Downstream Performance

As concluded earlier, variation in F barely affects GCNN stability. Nonetheless, the results show that downstream performance improves on CiteSeer, Cora, and AnomalyDetection and degrades on MovieLens as F grows. This difference in performance cannot be explained as a result of models learning better or worse augmentation-invariance as that would be reflected in the stability. Instead, for CiteSeer, Cora, and AnomalyDetection the benefit of larger F can be attributed to the fact that a large representation dimensionality $F_{\text{rep}} = F$ allows for a better spread of dissimilar nodes across representation space. Conversely, for MovieLens, a low representation dimensionality may already be sufficient for this purpose and, thus, larger F_{rep} may only hinder a linear regressor, due to the curse of dimensionality.

5.5. Discussion

As the experiments have shown, particular hyperparameter choices in a GCNN's architecture can have a great impact on its stability. We found that, in particular for ED, GCNNs with a higher depth L or of a higher order K are inherently less stable than their simpler counterparts. This is demonstrated by the fact that these stability differences are already present in untrained GCNNs. Additionally, we showed that, when GCNNs are trained using a contrastive loss, the relative differences in stability between architectures do not vanish, even though such a loss encourages stability. Moreover, we found that, for one dataset, differences in stability become *more* pronounced after contrastive training. This all suggests that overly complex GCNNs, despite their complexity, are prone *underfit* on the pretext task in contrastive learning.

Underfitting on the pretext task can, in turn, hurt downstream performance. We experimentally demonstrated that this is the case on at least one dataset: we found that, for AnomalyDetection, an increase in instability to the ED augmentation is matched with a decrease in performance, and vice versa. Notably, this behavior is not found on other datasets, but this can reasonably be explained by the fact that underfitting does not necessarily degrade downstream performance if the respective pretext task is inappropriate for a dataset. This is an oversight on our part but also not necessarily atypical for GCL: in GCL literature, augmentation functions often are inappropriate [52].

With our experiments, we provide an explanation for the poor performance of overly complex GCNNs in GCL. Namely, an overly high depth L or order K of a GCNN's architecture causes the GCNN to have worse inherent stability properties. As a consequence, the GCNN underfits on the GCL pretext task and, if the pretext task was appropriate to begin with, it achieves a worse downstream performance. This is an alternative explanation to oversmoothing [2, 29], which can also explain underfitting behavior of overly deep GCNNs for a class of smoothing GNN architectures. Our work expands on this by also directly relating the underfitting behavior to the order K of a higher-order GCNN. That said, the main contribution of this work is that it provides a new perspective on the already known phenomenon that overly complex GCNNs perform worse at GCL than simpler ones.

5.5.1. Limitations

The work presented in this thesis also comes with limitations. Firstly, most of the drawn conclusions are relevant only to the ED style of perturbation and augmentation. The other considered style of perturbation, EWP, did not provide strong results in the first two sets of experiments and, thus, was not even included in the following experiments. To extend our conclusions to other perturbation styles, similar experiments should be done on those perturbations. Specifically for evaluating the correlation between GCNN stability and downstream performance in GCL, it would be beneficial to consider other augmentation functions that are more appropriate for the respective datasets. In our experiments, ED was verified to be an appropriate augmentation only for one out of four datasets.

Secondly, while there is a link between our work and existing work on the oversmoothing problem, this link remains experimentally and theoretically unexplored. It is unclear as to whether stability degra-

dation as measured in our experiments and degradation due to oversmoothing are both tied to the same underlying phenomenon, or if these are completely disjoint mechanisms. Further studies have to be conducted to shed light on this.

6

Conclusion

In this final chapter, we answer the research questions based on the experimental results (Section 6.1) and, after that, we discuss possible directions for future work (Section 6.2).

6.1. Answering The Research Questions

In this thesis, we formulated the following research questions:

RQ1. When a GCNN is pre-trained in a contrastive manner with increasing amounts of augmentation, does its stability to that augmentation improve?

RQ2. When varying a GCNN's architecture, does its stability change in accordance to what theoretical stability bounds suggest?

RQ3. How does a GCNN's *untrained stability* as induced by its specific architecture interact with its *trained stability*, acquired through contrastive pre-training?

RQ4. After being pre-trained contrastively using augmentation, is the downstream performance of GCNNs correlated to their stability to that augmentation?

The goal of these research questions was to investigate why deeper GCL architectures tend to perform worse than their shallower counterparts. To answer **RQ1**, we considered a basic SimCLR-inspired GCL method "GRACE" for learning node representations on graphs [72]. We measured on four datasets how this style of pre-training affects the stability of the GCNN encoders to data augmentation. As augmentation functions, we chose Edge Weight Perturbation (EWP) and Edge Dropping (ED), due to their relevance to the theoretical stability bounds [9, 11]. For ED, we found that GCNNs indeed become increasingly more stable as we increased the intensity of the augmentation during training. For EWP, however, the contrastive learning had no significant effect on stability. Our hypothesis is that this is because EWP is too subtle of a perturbation, motivated by the fact instability induced by EWP was extremely low across the board.

To answer **RQ2**, we eliminated the influence of training on stability by only considering untrained, i.e. randomly initialized, GCNNs. To measure the effect of architecture, we varied the depth L, width F, and order K of the GCNNs. These experiments led to two important findings:

1. Varying F only significantly affects absolute instability. The normalized instability, which we deem to be the more informative metric of the two, remains mostly constant when varying F. This means that increasing F does not increase the instability of *individual* features, despite the looser theoretical bounds.

2. Increasing either L or K increases a GCNN's instability. This is in line with theoretical intuitions that more complex GCNNs are less stable. Though, this behavior was only consistently found for ED and when using the normalized instability metric (2.15).

The findings in the experiments for **RQ1** and **RQ2** paved the way for a third set of experiments, aimed at answering **RQ3**. Here, we considered three training settings: untrained GCNNs, GCNNs trained using GRACE without any augmentation, and GCNNs trained with GRACE with strong augmentation. We found that, even though training under progressively stronger GCL improves stability across architectures and on all datasets, the relative differences in stability between architectures do not diminish, even after training. In other words, simply training under stronger GCL cannot make up for poor stability as induced by architecture.

Lastly, we answered **RQ4** by measuring the downstream performance of GCNN encoders trained with GRACE under strong ED. We found that, for three out of four datasets, there is no clear link between instability and downstream performance. We hypothesize that this is because ED is not an appropriate augmentation for these datasets. For one dataset, AnomalyDetection, the one specifically designed around ED augmentation, we did find that instability and downstream performance are indeed strongly correlated: GCNNs with overly high L and K are both less stable and, consequently, perform worse.

This last finding is relevant to the overall goal of this thesis: to shed light on the poor performance of deeper GCNNs in GCL. We conclude that overly complex GCNNs can underfit on the contrastive pre-text task and, as a result, produce worse representations. These new insights can aid in the design of new model architectures for graphs that have better scaling properties and therefore better accommodate for the ever increasing availability of unlabeled graph data.

6.2. Future Work

We conclude this thesis with some recommendations for future work. We identify three different avenues for potential future work, each discussed in their respective section.

6.2.1. Evaluate More Types Of Perturbation

In our experiments, we only considered Edge Weight Perturbation (EWP) and Edge Dropping (ED) as perturbations and augmentations. Since the results for EWP were mostly inconclusive, our main conclusions only apply to ED. And, while ED is a commonly used augmentation in GCL, it is not the only one. Thus, repeating the same experiments on popular augmentations such as edge adding, node feature masking, and subgraph sampling would be of great value. Here, it would be best to focus on graph datasets for which the chosen style of augmentation is, in fact, appropriate, unlike most datasets considered in this thesis.

6.2.2. Evaluate Different GNN Architectures

Our experiments focused solely on the convolutional subset of GNNs: GCNNs. Our motivation for this is that the relevant theoretical work on stability applies only to GCNNs. GCNNs are however not the only style of GNN used in GCL. The Graph Attention Network (GAT) [55], for example, is also occasionally used. Hence, it would be useful to evaluate other GNN architectures from the perspective of stability to investigate if their behaviors align with that of GCNN. Specifically, it would be interesting to redo our experiments on architectures that are designed to overcome the oversmoothing issue such as the one presented in [29]. Additionally, it could also be insightful to evaluate the stability properties of non-GNN architectures for graph data such as Graphormer [65].

6.2.3. Relate GCNN Stability To Oversmoothing

We found that GCNNs can underfit on the GCL pretext task as a result of overly complex architecture, which can ultimately harm downstream performance. The oversmoothing problem provides an alternative explanation for the poor performance of deep GNNs [2, 29], though it is not applied to GCL specifically, unlike our work. As stated in the limitations section in Chapter 5, our current work does not shed light on whether or not oversmoothing and instability of overly complex GCNNs are related problems. Hence, further empirical experiments are needed to investigate this. A starting point could be to look into whether there is a correlation between GCNN stability and the Mean Average Distance metric for measuring smoothness as introduced in [2].

References

[1] Adrien Bardes, Jean Ponce, and Yann Lecun. “VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning”. In: (2021).

[2] Deli Chen et al. “Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View”. In: *CoRR* abs/1909.03211 (2019). arXiv: 1909.03211. URL: http://arxiv.org/abs/1909.03211.

[3] Leiyu Chen et al. “Review of image classification algorithms based on convolutional neural networks”. In: *Remote Sensing* 13.22 (2021), p. 4712.

[4] Ting Chen et al. “A Simple Framework for Contrastive Learning of Visual Representations”. In: (2020). URL: https://github.com/google-research/simclr..

[5] Xinlei Chen and Kaiming He. “Exploring Simple Siamese Representation Learning”. In: (2021).

[6] Guanyi Chu et al. *CuCo: Graph Representation with Curriculum Contrastive Learning*. Tech. rep. Main Track. Aug. 2021, pp. 2300–2306. DOI: 10.24963/ijcai.2021/317. URL: https://doi.org/10.24963/ijcai.2021/317.

[7] Florinel-Alin Croitoru et al. “Diffusion Models in Vision: A Survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.9 (2023), pp. 10850–10869. DOI: 10.1109/TPAMI.2023.3261988.

[8] Shengyu Feng et al. “Adversarial Graph Contrastive Learning with Information Regularization”. In: *WWW 2022 - Proceedings of the ACM Web Conference 2022*. Association for Computing Machinery, Inc, Apr. 2022, pp. 1362–1371. ISBN: 9781450390965. DOI: 10.1145/3485447.3512183.

[9] Fernando Gama, Joan Bruna, and Alejandro Ribeiro. “Stability Properties of Graph Neural Networks”. In: (May 2019). DOI: 10.1109/TSP.2020.3026980. URL: http://arxiv.org/abs/1905.04497%20http://dx.doi.org/10.1109/TSP.2020.3026980.

[10] Fernando Gama et al. “Graphs, Convolutions, and Neural Networks”. In: November (2020).

[11] Zhan Gao, Elvin Isufi, and Alejandro Ribeiro. “Stability of Graph Convolutional Neural Networks to Stochastic Perturbations”. In: (June 2021). URL: http://arxiv.org/abs/2106.10526.

[12] Scott R Granter, Andrew H Beck, and David J Papke Jr. “AlphaGo, deep learning, and the future of the human microscopist”. In: *Archives of pathology & laboratory medicine* 141.5 (2017), pp. 619–621.

[13] Jean-Bastien Grill et al. “Bootstrap Your Own Latent A New Approach to Self-Supervised Learning”. In: (2020). URL: https://github.com/deepmind/deepmind-research/tree/master/byol.

[14] William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216. URL: http://arxiv.org/abs/1706.02216.

[15] F Maxwell Harper and Joseph A Konstan. “The movielens datasets: History and context”. In: *Acm transactions on interactive intelligent systems (tiis)* 5.4 (2015), pp. 1–19.

[16] Kaveh Hassani and Amir Hosein Khasahmadi. “Contrastive Multi-View Representation Learning on Graphs”. In: (June 2020). URL: http://arxiv.org/abs/2006.05582.

[17] Kaveh Hassani and Amir Hosein Khasahmadi. “Learning Graph Augmentations to Learn Graph Representations”. In: (Jan. 2022). URL: http://arxiv.org/abs/2201.09830.

[18] R Devon Hjelm et al. “Learning deep representations by mutual information estimation and maximization”. In: (Aug. 2018). URL: http://arxiv.org/abs/1808.06670.

[19] Zhenyu Hou et al. “A Decoding-Enhanced Masked Self-Supervised Graph Learner”. In: (Apr. 2023). DOI: 10.1145/3543507.3583379. URL: http://arxiv.org/abs/2304.04779.

[20] Zhenyu Hou et al. “GraphMAE: Self-Supervised Masked Graph Autoencoders”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, Aug. 2022, pp. 594–604. ISBN: 9781450393850. DOI: 10.1145/3534678.3539321.

[21] Elvin Isufi et al. “Graph Filters for Signal Processing and Machine Learning on Graphs”. In: (Nov. 2022). URL: http://arxiv.org/abs/2211.08854.

[22] Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. “AMMUS : A Survey of Transformer-based Pretrained Models in Natural Language Processing”. In: *CoRR* abs/2108.05542 (2021). arXiv: 2108.05542. URL: https://arxiv.org/abs/2108.05542.

[23] Henry Kenlay, Dorina Thanou, and Xiaowen Dong. *Interpretable Stability Bounds for Spectral Graph Filters*. Tech. rep. 2021.

[24] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: (Sept. 2016). URL: http://arxiv.org/abs/1609.02907.

[25] Namkyeong Lee, Junseok Lee, and Chanyoung Park. “Augmentation-Free Self-Supervised Learning on Graphs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.7 (June 2022), pp. 7372–7380. ISSN: 2374-3468. DOI: 10.1609/AAAI.V36I7.20700. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20700.

[26] Ron Levie et al. *Transferability of Spectral Graph Convolutional Neural Networks*. Tech. rep. 2021, pp. 1–59.

[27] Jintang Li et al. “MaskGAE: Masked Graph Modeling Meets Graph Autoencoders”. In: (May 2022). URL: http://arxiv.org/abs/2205.10053.

[28] Lu Lin, Jinghui Chen, and Hongning Wang. “Spectral Augmentation for Self-Supervised Learning on Graphs”. In: (Oct. 2022). URL: http://arxiv.org/abs/2210.00643.

[29] Meng Liu, Hongyang Gao, and Shuiwang Ji. “Towards Deeper Graph Neural Networks”. In: *CoRR* abs/2007.09296 (2020). arXiv: 2007.09296. URL: https://arxiv.org/abs/2007.09296.

[30] Nian Liu et al. “Revisiting Graph Contrastive Learning from the Perspective of Graph Spectrum”. In: (Oct. 2022). URL: http://arxiv.org/abs/2210.02330.

[31] Xiao Liu et al. “Self-supervised Learning: Generative or Contrastive”. In: *CoRR* abs/2006.08218 (2020). arXiv: 2006.08218. URL: https://arxiv.org/abs/2006.08218.

[32] Yixin Liu et al. “Graph self-supervised learning: A survey”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.6 (2022), pp. 5879–5900.

[33] Yue Liu et al. “Hard Sample Aware Network for Contrastive Deep Graph Clustering”. In: (Dec. 2022). URL: http://arxiv.org/abs/2212.08665.

[34] Yujie Mo et al. “Simple unsupervised graph representation learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 7. 2022, pp. 7797–7805. URL: https://github.com/YujieMo/SUGRL.

[35] Yishuang Ning et al. “A Review of Deep Learning Based Speech Synthesis”. In: *Applied Sciences* 9.19 (2019). ISSN: 2076-3417. DOI: 10.3390/app9194050. URL: https://www.mdpi.com/2076-3417/9/19/4050.

[36] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation Learning with Contrastive Predictive Coding”. In: *CoRR* abs/1807.03748 (2018). arXiv: 1807.03748. URL: http://arxiv.org/abs/1807.03748.

[37] Jiwoong Park et al. “Symmetric Graph Convolutional Autoencoder for Unsupervised Graph Representation Learning”. In: (Aug. 2019). URL: http://arxiv.org/abs/1908.02441.

[38] Sung-Wook Park et al. “Review on Generative Adversarial Networks: Focusing on Computer Vision and Its Applications”. In: *Electronics* 10.10 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10101216. URL: https://www.mdpi.com/2079-9292/10/10/1216.

[39] Zhen Peng et al. “Graph Representation Learning via Graphical Mutual Information Maximization”. In: *The Web Conference 2020 - Proceedings of the World Wide Web Conference, WWW 2020*. Association for Computing Machinery, Inc, Apr. 2020, pp. 259–270. ISBN: 9781450370233. DOI: 10.1145/3366423.3380112.

[40] Senthil Purushwalkam and Abhinav Gupta. “Demystifying Contrastive Self-Supervised Learning: Invariances, Augmentations and Dataset Biases”. In: (2020).

[41] Jiezhong Qiu et al. “GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training”. In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, Aug. 2020, pp. 1150–1160. ISBN: 9781450379984. DOI: 10.1145/3394486.3403168.

[42] Waseem Rawat and Zenghui Wang. “Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review”. In: *Neural Computation* 29.9 (2017), pp. 2352–2449. DOI: 10.1162/neco_a_00990.

[43] Prithviraj Sen et al. “Collective classification in network data”. In: *AI magazine* 29.3 (2008), pp. 93–93.

[44] David I Shuman et al. “The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains”. In: (Oct. 2012). DOI: 10.1109/MSP.2012.2235192. URL: http://arxiv.org/abs/1211.0053%20http://dx.doi.org/10.1109/MSP.2012.2235192.

[45] Fan-Yun Sun et al. “InfoGraph: Unsupervised and Semi-Supervised Graph-Level Representation Learning via Mutual Information Maximization”. In: (2020).

[46] Susheel Suresh et al. “Adversarial Graph Augmentation to Improve Graph Contrastive Learning”. In: (June 2021). URL: http://arxiv.org/abs/2106.05819.

[47] Qiaoyu Tan et al. “MGAE: Masked Autoencoders for Self-Supervised Learning on Graphs”. In: (Jan. 2022). URL: http://arxiv.org/abs/2201.02534.

[48] Qiaoyu Tan et al. “S2GAE: Self-Supervised Graph Autoencoders are Generalizable Learners with Graph Masking”. In: *WSDM 2023 - Proceedings of the 16th ACM International Conference on Web Search and Data Mining*. Association for Computing Machinery, Inc, Feb. 2023, pp. 787–795. ISBN: 9781450394079. DOI: 10.1145/3539597.3570404.

[49] Shantanu Thakoor et al. “Large-Scale Representation Learning on Graphs via Bootstrapping”. In: (Feb. 2021). URL: http://arxiv.org/abs/2102.06514.

[50] Yonglong Tian et al. “What Makes for Good Views for Contrastive Learning?” In: (2020). URL: http://hobbitlong.github.io/InfoMin.

[51] Amirsina Torfi et al. “Natural Language Processing Advancements By Deep Learning: A Survey”. In: *CoRR* abs/2003.01200 (2020). arXiv: 2003.01200. URL: https://arxiv.org/abs/2003.01200.

[52] Puja Trivedi et al. “Augmentations in Graph Contrastive Learning: Current Methodological Flaws & Towards Better Practices”. In: *WWW 2022 - Proceedings of the ACM Web Conference 2022*. Association for Computing Machinery, Inc, Apr. 2022, pp. 1538–1549. ISBN: 9781450390965. DOI: 10.1145/3485447.3512200.

[53] Wenxuan Tu et al. “RARE: Robust Masked Graph Autoencoder”. In: (Apr. 2023). URL: http://arxiv.org/abs/2304.01507.

[54] Petar Veličković et al. “Deep Graph Infomax”. In: (Sept. 2018). URL: http://arxiv.org/abs/1809.10341.

[55] Petar Veličković et al. “Graph attention networks”. In: *arXiv preprint arXiv:1710.10903* (2017).

[56] Haonan Wang et al. “Augmentation-Free Graph Contrastive Learning with Performance Guarantee”. In: (Apr. 2022). URL: http://arxiv.org/abs/2204.04874.

[57] Xu Wang et al. “Deep Reinforcement Learning: A Survey”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–15. DOI: 10.1109/TNNLS.2022.3207346.

[58] Yuyang Wang et al. “Molecular contrastive learning of representations via graph neural networks”. In: *Nature Machine Intelligence* 4.3 (Mar. 2022), pp. 279–287. ISSN: 25225839. DOI: 10.1038/s42256-022-00447-x.

[59] Zixin Wen and Yuanzhi Li. “Toward Understanding the Feature Learning Process of Self-supervised Contrastive Learning”. In: (2021).

[60] Jun Xia et al. “ProGCL: Rethinking Hard Negative Mining in Graph Contrastive Learning”. In: (Oct. 2021). URL: http://arxiv.org/abs/2110.02027.

[61] Yaochen Xie et al. “Self-supervised learning of graph neural networks: A unified review”. In: *IEEE transactions on pattern analysis and machine intelligence* 45.2 (2022), pp. 2412–2429.

[62] Dongkuan Xu et al. “Infogcl: Information-aware graph contrastive learning”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 30414–30425.

[63] Keyulu Xu et al. “How Powerful are Graph Neural Networks?” In: (Oct. 2018). URL: http://arxiv.org/abs/1810.00826.

[64] Longqi Yang, Liangliang Zhang, and Wenjing Yang. “Graph Adversarial Self-Supervised Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 14887–14899. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/7d3010c11d08cf990b7614d2c2ca9098-Paper.pdf.

[65] Chengxuan Ying et al. “Do Transformers Really Perform Bad for Graph Representation?” In: (June 2021). URL: http://arxiv.org/abs/2106.05234.

[66] Yuning You et al. “Bringing your own view: Graph contrastive learning without prefabricated data augmentations”. In: *WSDM 2022 - Proceedings of the 15th ACM International Conference on Web Search and Data Mining*. Association for Computing Machinery, Inc, Feb. 2022, pp. 1300–1309. ISBN: 9781450391320. DOI: 10.1145/3488560.3498416.

[67] Yuning You et al. “When Does Self-Supervision Help Graph Convolutional Networks?” In: (June 2020). URL: http://arxiv.org/abs/2006.09136.

[68] Jure Zbontar et al. “Barlow Twins: Self-Supervised Learning via Redundancy Reduction”. In: (2021). URL: https://github.com/facebookresearch/barlowtwins.

[69] Hengrui Zhang et al. “From Canonical Correlation Analysis to Self-supervised Graph Neural Networks”. In: (June 2021). URL: http://arxiv.org/abs/2106.12484.

[70] Sixiao Zhang et al. “Graph Masked Autoencoders with Transformers”. In: (Feb. 2022). URL: http://arxiv.org/abs/2202.08391.

[71] Ziwei Zhang, Peng Cui, and Wenwu Zhu. “Deep Learning on Graphs: A Survey”. In: *CoRR* abs/1812.04202 (2018). arXiv: 1812.04202. URL: http://arxiv.org/abs/1812.04202.

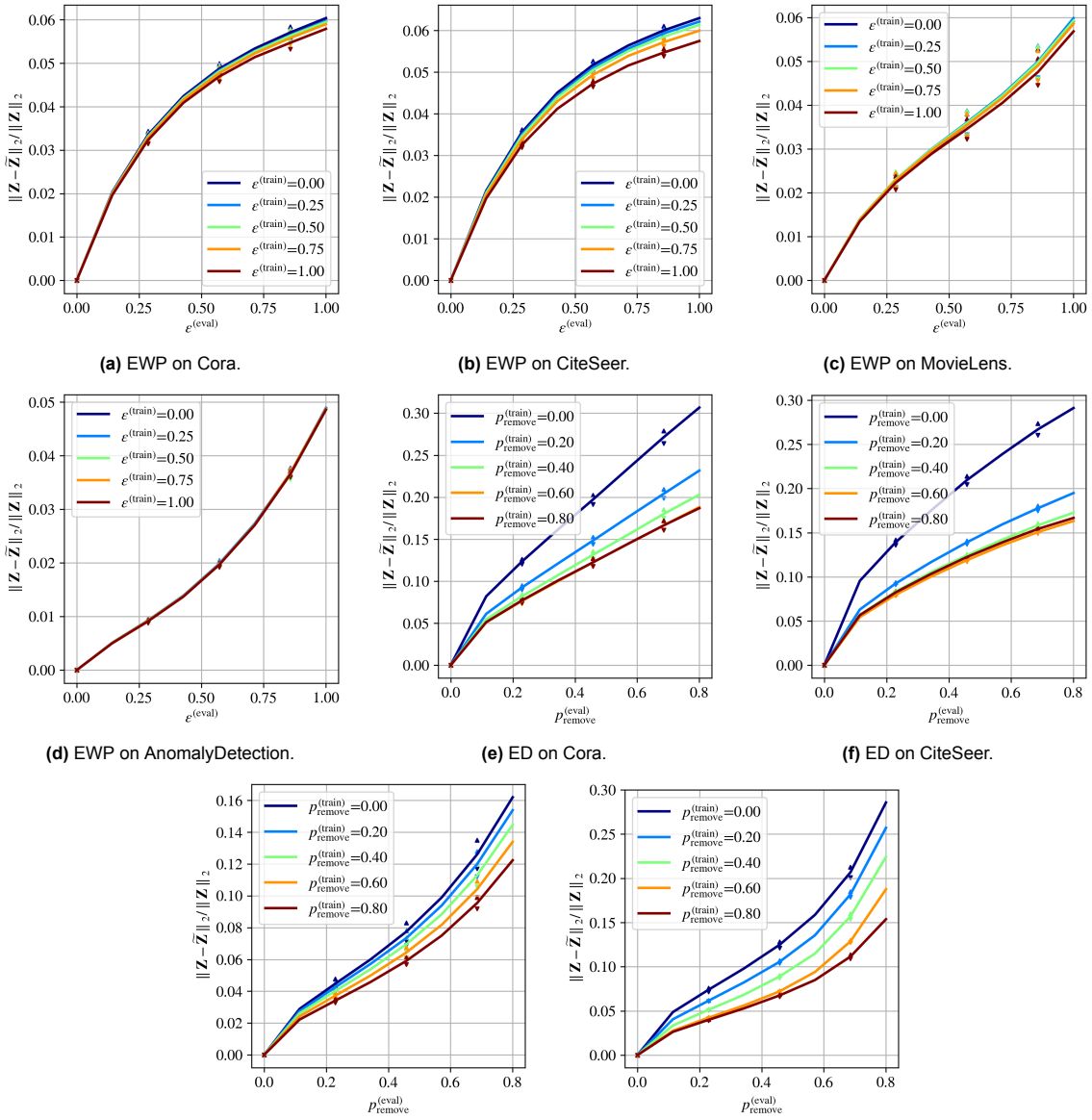
[72] Yanqiao Zhu et al. “Deep Graph Contrastive Representation Learning”. In: (June 2020). URL: http://arxiv.org/abs/2006.04131.

[73] Yanqiao Zhu et al. “Graph contrastive learning with adaptive augmentation”. In: *The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021*. Association for Computing Machinery, Inc, Apr. 2021, pp. 2069–2080. ISBN: 9781450383127. DOI: 10.1145/3442381.3449802.

A

Full Instability Plots

A.1. Experiments For RQ1



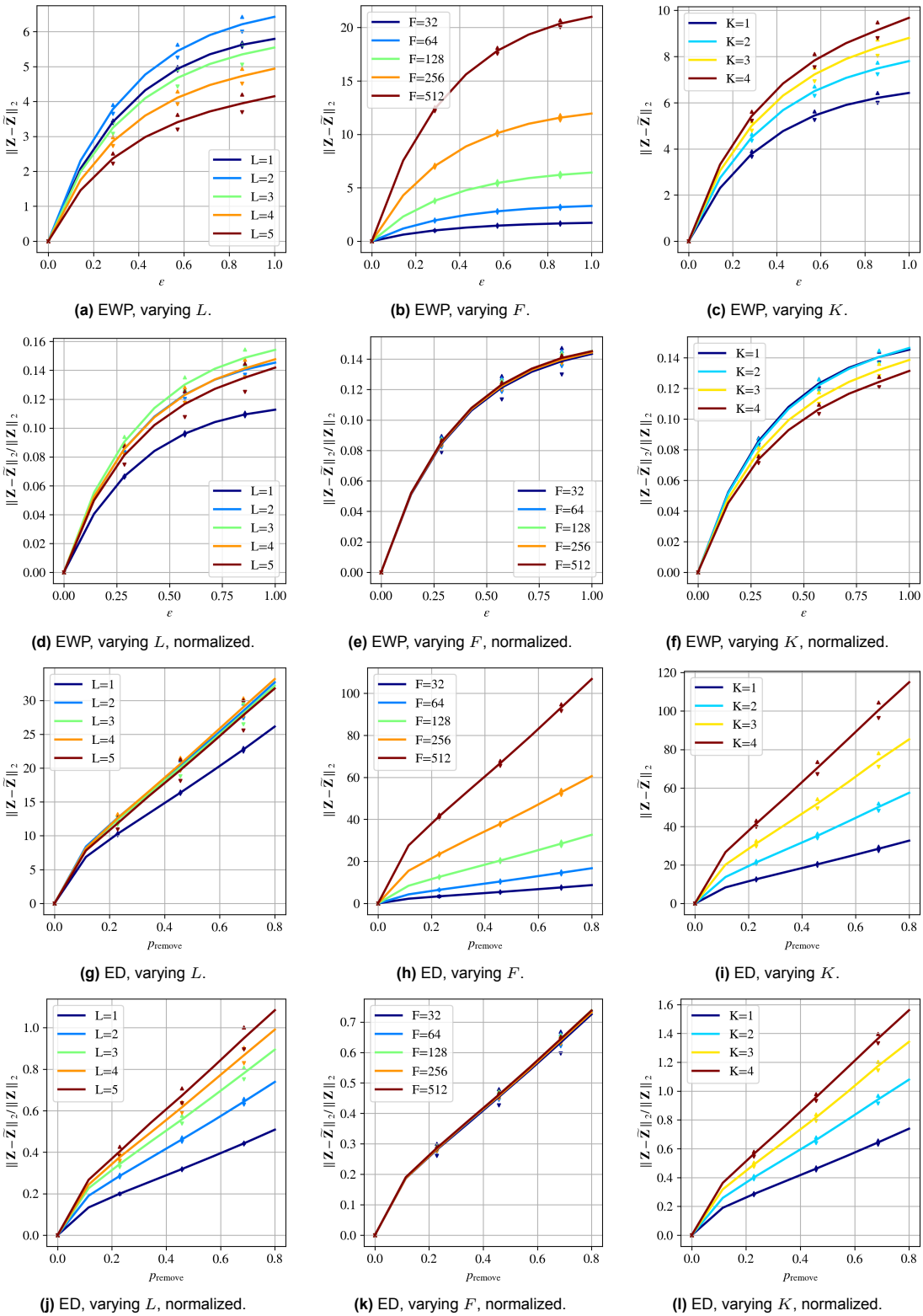
(a) EWP on Cora. **(b)** EWP on CiteSeer. **(c)** EWP on MovieLens.

(d) EWP on AnomalyDetection. **(e)** ED on Cora. **(f)** ED on CiteSeer.

(g) ED on MovieLens. **(h)** ED on AnomalyDetection.

Figure A.1: The *normalized* instability (2.15), plotted on the vertical axis, of the GRACE encoder when evaluated under increasing amounts of perturbation to the input graph, plotted on the horizontal axis. The different lines represents the mean instability of encoders trained under different augmentation strengths of the respective augmentation. The arrow heads indicate the standard deviation from the mean. The first four plots correspond to experiments using edge weight perturbation (EWP), the remaining four correspond to those using edge dropping (ED). Perturbation intensity is measured in ϵ and p_{remove} for EWP and ED, respectively. Note that, in each experiment, there are two independent intensities: train intensity and evaluation intensity. Train intensity refers to the intensity of augmentation that was used during contrastive training. Evaluation intensity refers to the perturbation intensity used to measure instability.

A.2. Experiments For RQ2



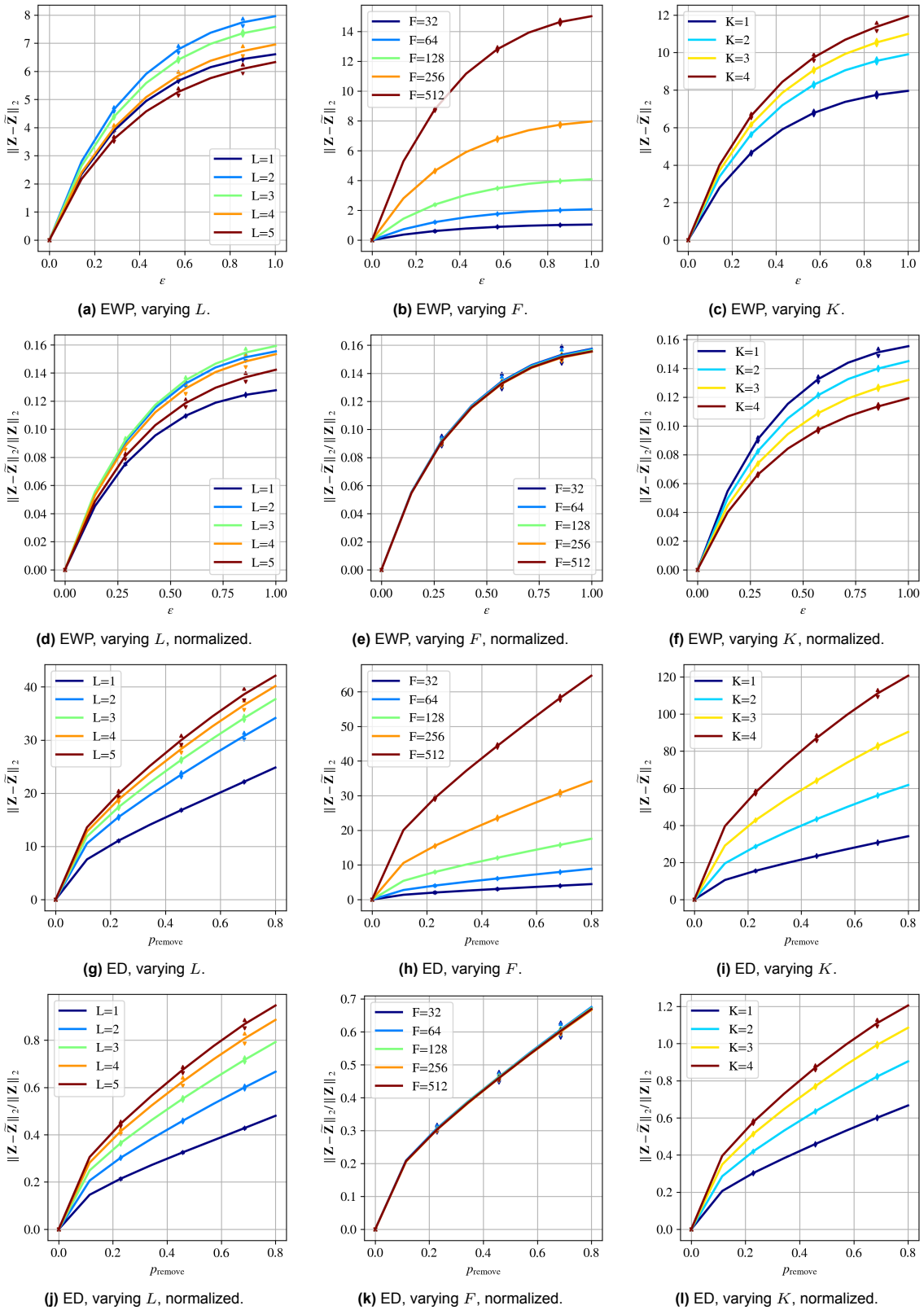
(a) EWP, varying L. **(b)** EWP, varying F. **(c)** EWP, varying K.

(d) EWP, varying L, normalized. **(e)** EWP, varying F, normalized. **(f)** EWP, varying K, normalized.

(g) ED, varying L. **(h)** ED, varying F. **(i)** ED, varying K.

(j) ED, varying L, normalized. **(k)** ED, varying F, normalized. **(l)** ED, varying K, normalized.

Figure A.2: Stability plots on the **Cora** dataset in the same style as Figure A.1, but this time varying architecture of GCNNs. Both perturbation styles, EWP and ED, are considered and results are given in two different instability metrics: normalized instability (2.15) and absolute instability (2.14). Each setting is repeated sixteen times.



(a) EWP, varying L.

(b) EWP, varying F.

(c) EWP, varying K.

(d) EWP, varying L, normalized.

(e) EWP, varying F, normalized.

(f) EWP, varying K, normalized.

(g) ED, varying L.

(h) ED, varying F.

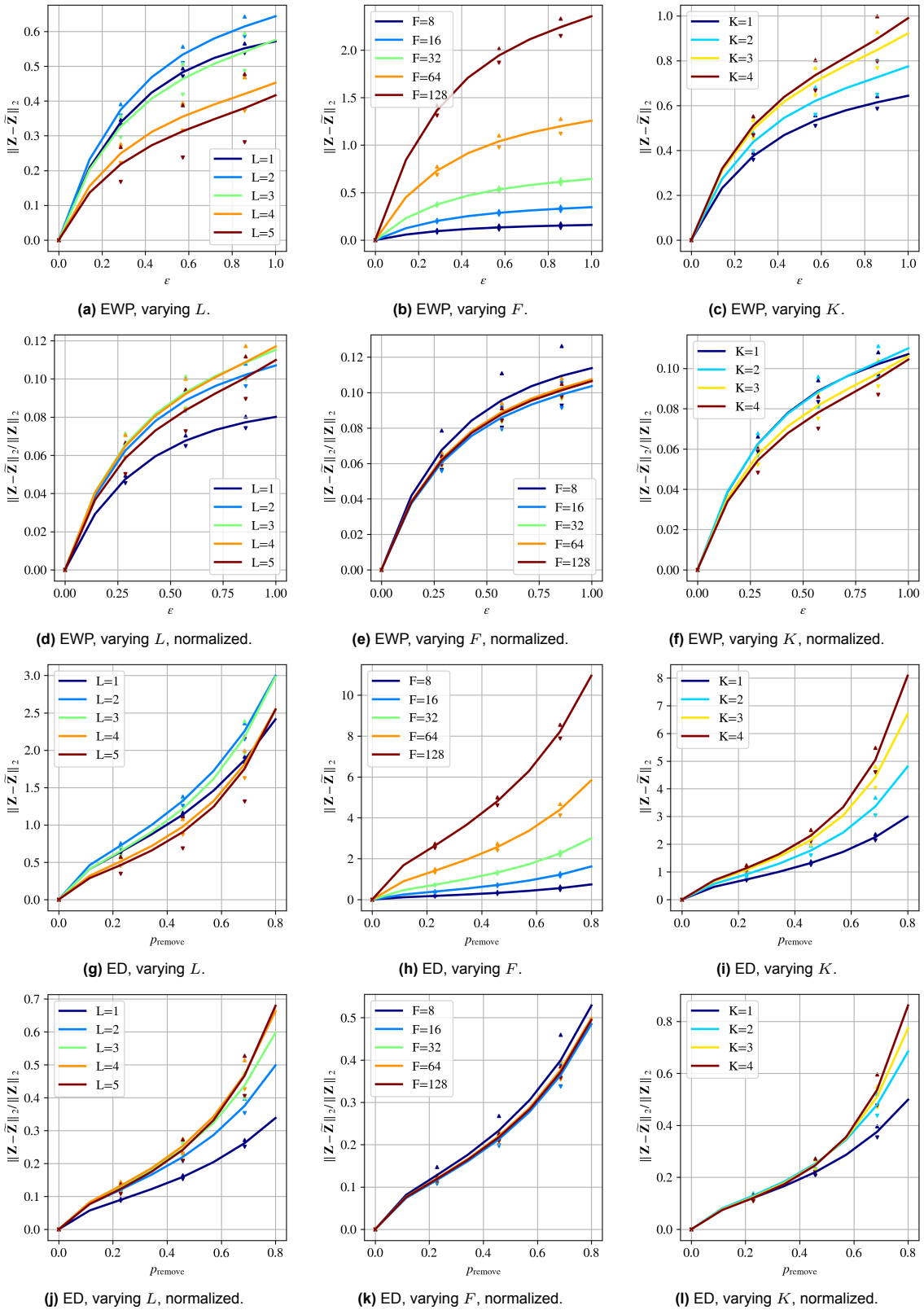
(i) ED, varying K.

(j) ED, varying L, normalized.

(k) ED, varying F, normalized.

(l) ED, varying K, normalized.

Figure A.3: Stability plots in the same style as Figure A.2, but this time on the **CiteSeer** dataset.



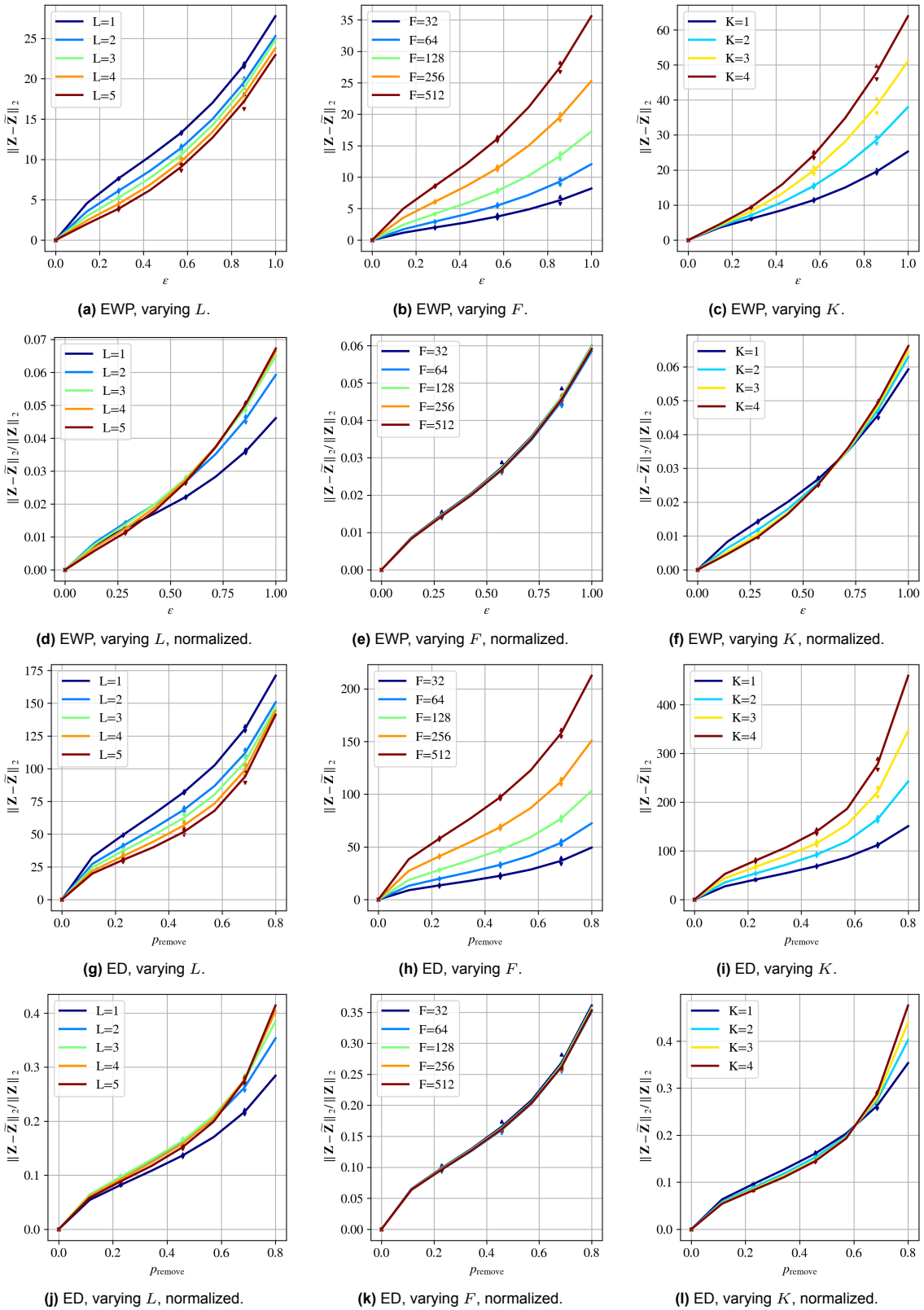
(a) EWP, varying L. **(b)** EWP, varying F. **(c)** EWP, varying K.

(d) EWP, varying L, normalized. **(e)** EWP, varying F, normalized. **(f)** EWP, varying K, normalized.

(g) ED, varying L. **(h)** ED, varying F. **(i)** ED, varying K.

(j) ED, varying L, normalized. **(k)** ED, varying F, normalized. **(l)** ED, varying K, normalized.

Figure A.4: Stability plots in the same style as Figure A.2, but this time on the **MovieLens** dataset.



(a) EWP, varying L.

(b) EWP, varying F.

(c) EWP, varying K.

(d) EWP, varying L, normalized.

(e) EWP, varying F, normalized.

(f) EWP, varying K, normalized.

(g) ED, varying L.

(h) ED, varying F.

(i) ED, varying K.

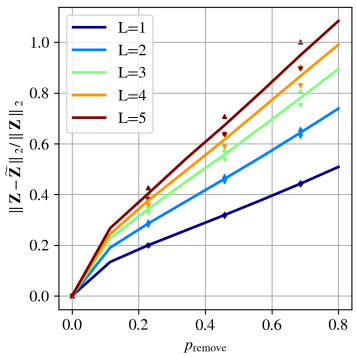
(j) ED, varying L, normalized.

(k) ED, varying F, normalized.

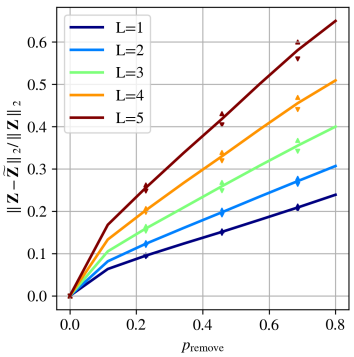
(l) ED, varying K, normalized.

Figure A.5: Stability plots in the same style as Figure A.2, but this time on the **AnomalyDetection** dataset.

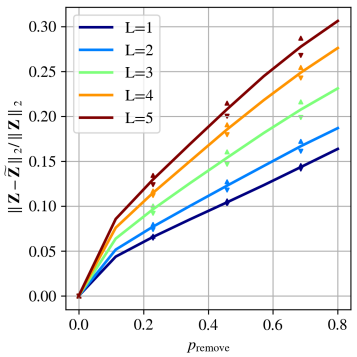
A.3. Experiments For RQ3 And RQ4



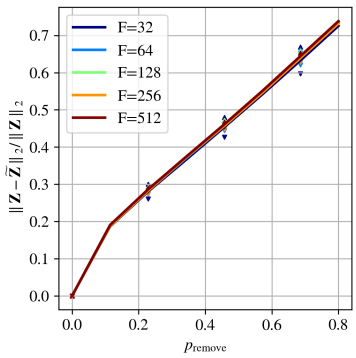
(a) Untrained, varying L.



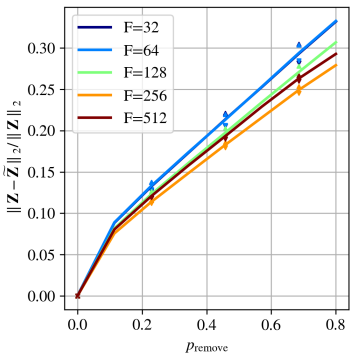
(b) Pre-trained with identity, varying L.



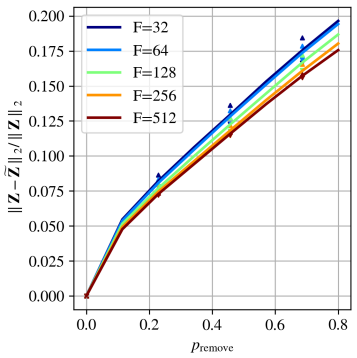
(c) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying L.



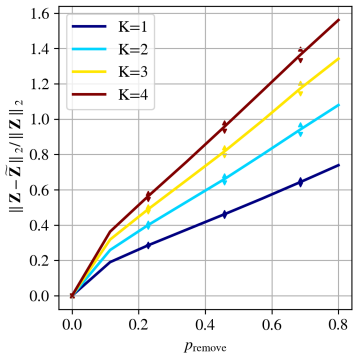
(d) Untrained, varying F.



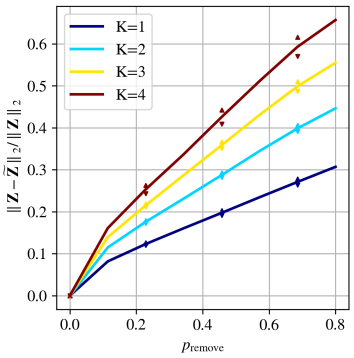
(e) Pre-trained with identity, varying F.



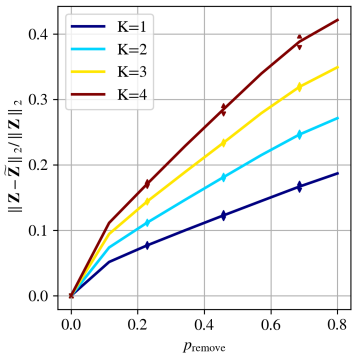
(f) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying F.



(g) Untrained, varying K.

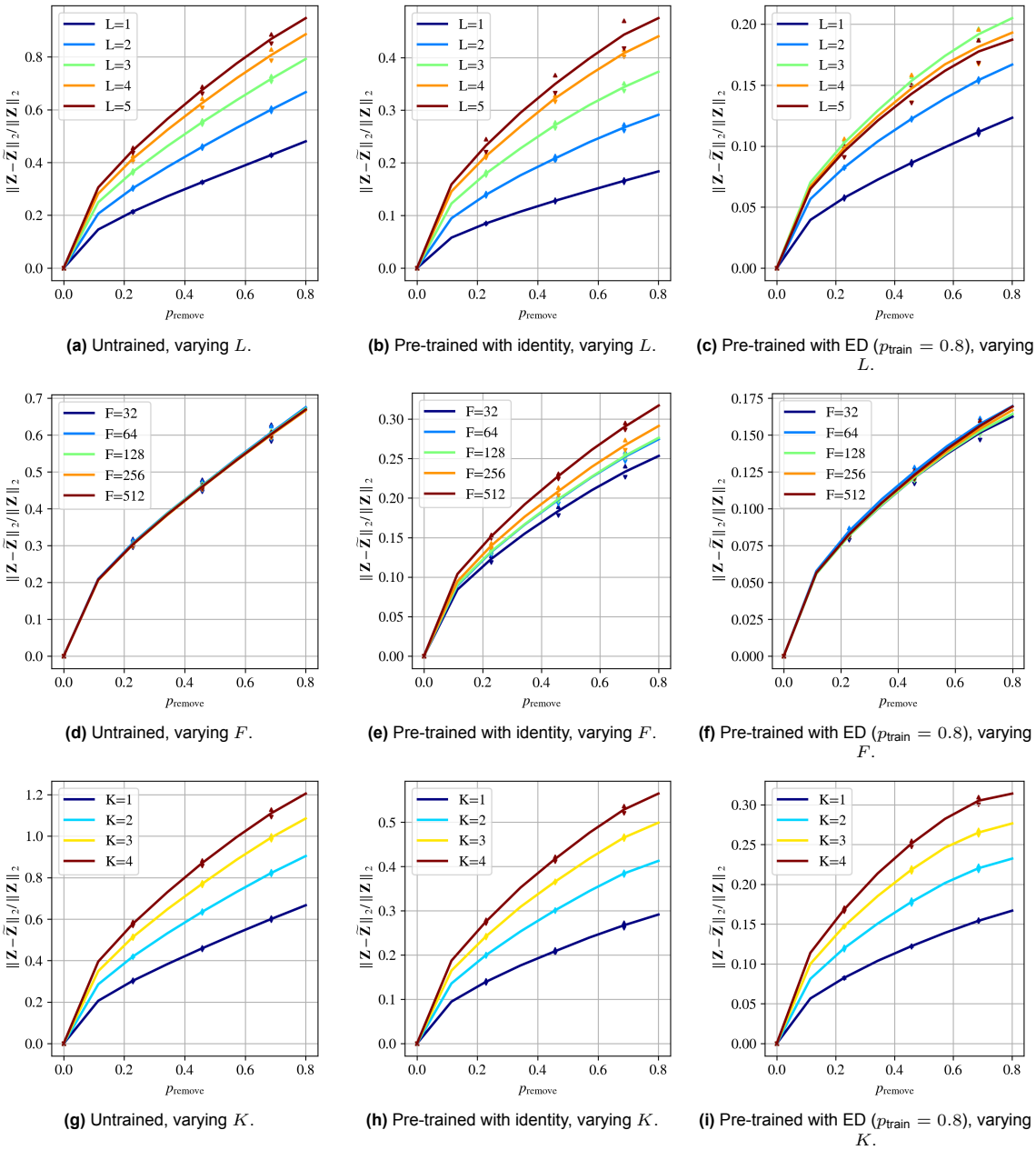


(h) Pre-trained with identity, varying K.



(i) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying K.

Figure A.6: This set of *normalized* instability (2.15) plots gives an overview of how untrained stability and trained stability interact under edge dropping (ED) on the **Cora** dataset. On the rows different variations of architecture are shown and on the columns different training settings. Those training settings are: randomly initialized (left), trained with GRACE with identity augmentation (middle), and trained with GRACE with a strong ED as augmentation (right). For the randomly initialized models, experiments are repeated 16 times. For the trained models, experiments are repeated 8 times. A noteworthy observation is that instabilities decrease from left to right, meaning that the models gain trained stability from pre-training. Additionally, the ordering and relative spread of the instability curves remain more or less the same, most notably for the settings that vary L and K.



(a) Untrained, varying L.

(b) Pre-trained with identity, varying L.

(c) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying L.

(d) Untrained, varying F.

(e) Pre-trained with identity, varying F.

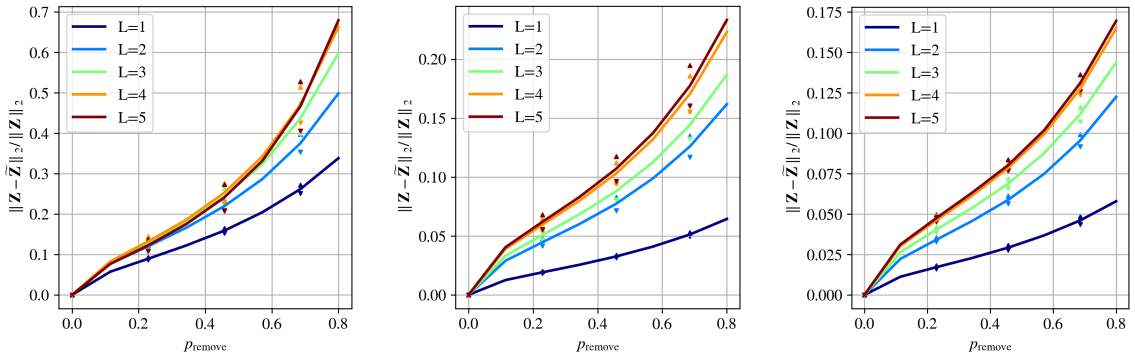
(f) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying F.

(g) Untrained, varying K.

(h) Pre-trained with identity, varying K.

(i) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying K.

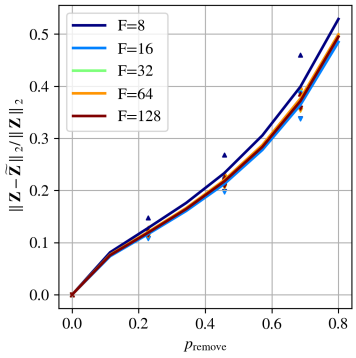
Figure A.7: Stability plots in the same style as Figure A.6, but on the **CiteSeer** dataset.



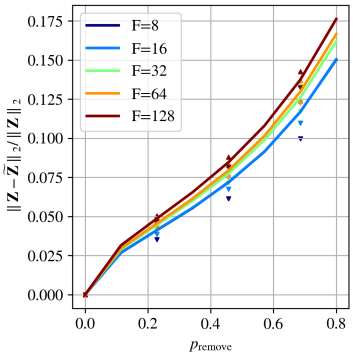
(a) Untrained, varying L.

(b) Pre-trained with identity, varying L.

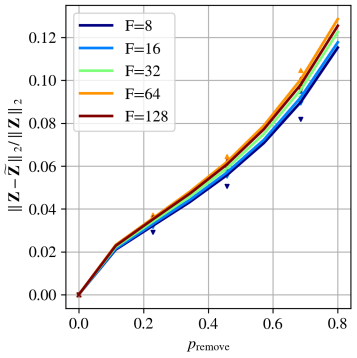
(c) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying L.



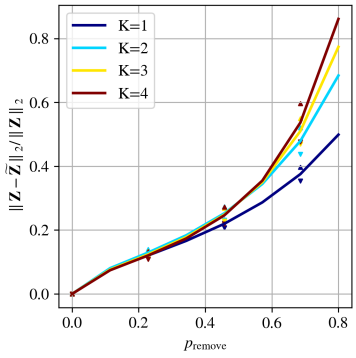
(d) Untrained, varying F.



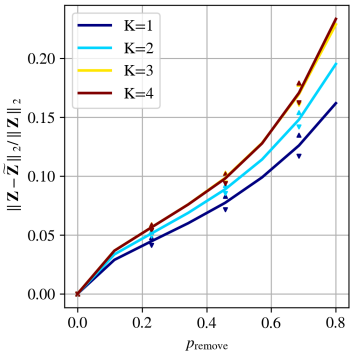
(e) Pre-trained with identity, varying F.



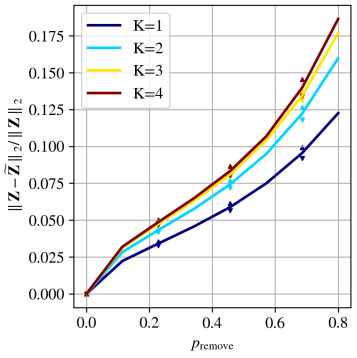
(f) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying F.



(g) Untrained, varying K.

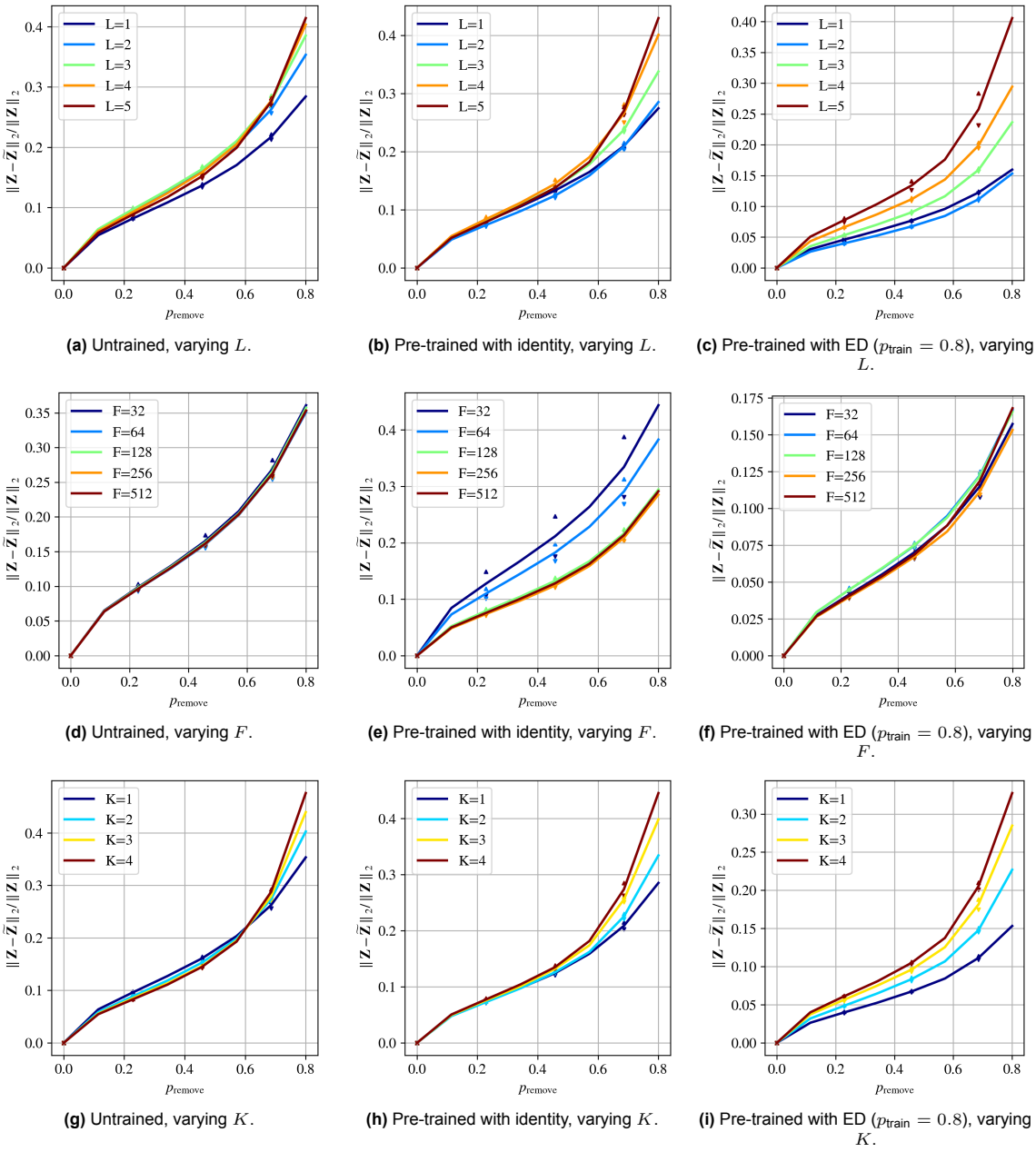


(h) Pre-trained with identity, varying K.



(i) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying K.

Figure A.8: Stability plots in the same style as Figure A.6, but on the **MovieLens** dataset.



(a) Untrained, varying L.

(b) Pre-trained with identity, varying L.

(c) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying L.

(d) Untrained, varying F.

(e) Pre-trained with identity, varying F.

(f) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying F.

(g) Untrained, varying K.

(h) Pre-trained with identity, varying K.

(i) Pre-trained with ED ($p_{\text{train}} = 0.8$), varying K.

Figure A.9: Stability plots in the same style as Figure A.6, but on the **AnomalyDetection** dataset.