Fakultät für Informatik
Technische Universität München

Adversarial Robustness of Graph Neural Networks

Daniel Zügner

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:
Prof. Debarghya Ghoshdastidar, Ph. D.

Prüfende der Dissertation:
1. Prof. Dr. Stephan Günnemann
2. Prof. Dr. Evangelos Papalexakis,
   University of California Riverside

Die Dissertation wurde am 07.02.2022 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 26.04.2022 angenommen.

Abstract

Graph neural networks (GNNs) have rapidly revolutionized the field of machine learning on graphs and are state of the art in many tasks such as node classification, link prediction, and graph classification. Moreover, GNNs have been adopted in industry for applications such as recommendation, drug discovery, or estimation of arrival time in routing. In this thesis we look at GNNs from a perspective of adversarial robustness. We generalize the notion of adversarial attacks – small perturbations to the input data deliberately crafted to mislead a machine learning model – from traditional vector data such as images to graphs. We present adversarial attack algorithms on GNNs that target the training phase of the models, leading to drastically reduced performance after training on the poisoned data. In some cases, a small number of perturbations is sufficient to degrade the performance of a state-of-the-art GNN below that of a simple classifier that neglects graph information altogether. In addition to adversarial attacks, we also focus on improving the robustness of GNNs against attacks. We propose robustness certification procedures for perturbations of the node attributes as well as the graph structure. These certificates guarantee that no perturbation within some constraints can change the prediction of the model. Further, we use our certification in a robust training procedure which strongly improves the GNN robustness. We further provide retrospective insight and summarize the current state of the research field of studying adversarial robustness of GNNs. Finally, we consider broader impact aspects of ML in general and GNNs in particular, and highlight open questions for future research.

Zusammenfassung

Graph-neuronale Netze (GNNs) haben das maschinelle Lernen (ML) auf Graphen in den letzten Jahren rasant verändert. GNNs sind *state of the art* in vielen typischen Aufgaben für maschinelles Lernen auf Graphen wie der Klassifikation von Knoten, Kanten-Vorhersage sowie Graph-Klassifikation. GNNs haben auch sehr schnell Eingang gefunden zu Anwendungen in der Wirtschaft wie beispielsweise der Empfehlung von Inhalten, Entdeckung neuer Medikamente oder der Schätzung der Ankunftszeit bei Navigation. In dieser Doktorarbeit werden GNNs aus der Perspektive der feindlichen (engl. *adversarial*) Robustheit betrachtet. Das Konzept der feindlichen Angriffe (engl. *adversarial attacks*) – kleiner Störungen in den Eingabedaten, die speziell darauf ausgerichtet sind, ein ML-Modell zu täuschen – wird generalisiert von traditionellen unabhängigen Datentypen wie Bildern hin zu Graphen. Es werden Algorithmen für feindliche Angriffe auf GNNs präsentiert, die die Lernphase des Modells adressieren und so zu drastischen Einbußen in der Genauigkeit der Modelle führen, die auf den vergifteten (engl. *poisoned*) Daten trainiert wurden. In einigen Fällen reicht eine kleine Anzahl an Störungen sogar aus, um die Genauigkeit eines *state of the art* GNN-Modells so stark zu reduzieren, dass sie unter der Genauigkeit eines einfachen Modells liegt, das die Graph-Struktur ignoriert. Im zweiten Teil der Arbeit liegt der Fokus auf der Verbesserung der Robustheit von GNNs gegenüber Angriffen. Es werden Methoden für die mathematische Zertifizierung der Robustheit von GNNs gegenüber Störungen der Knoten-Attribute sowie der Graph-Struktur selbst präsentiert. Diese Zertifikate garantieren, dass keine Störung innerhalb eines festgelegten Rahmens die Vorhersage des Modells verändern kann. Zudem wird die Zertifizierungs-Methode dazu genutzt, einen robusten Lernalgorithmus für GNNs zu entwickeln, der die Robustheit der finalen Modelle drastisch erhöht. Abschließend wird der derzeitige Stand der Forschung zu robusten GNNs zusammengefasst und offene Fragen für zukünftige Forschungsprojekte aufgezeigt.

Acknowledgements

I am deeply grateful to my supervisor Prof. Stephan Günnemann. I could not have wished for a better PhD supervisor. I am humbled by your technical expertise and personal mentoring skills, which you generously and extensively provided to me throughout my studies. I have learned incredibly much from you and I hope to continue to do so in the future. I will always remember our intense and fruitful research discussions and the wonderful time I had with the DAML group you have created at TUM. Thank you.

Special thanks also to Prof. Jure Leskovec from Stanford University for hosting me during my research stay, for which I am very grateful. I also thank my PhD mentor Dr. Michele Catasta, whom I got to know at Stanford. Thank you for supporting me and making me feel welcome during my research stay with your warm and positive energy. Thank you for continuing to support me after my stay; for your support finishing our research project and for providing valuable advice whenever I needed it.

Thank you Tim Januschowski, François-Xavier Aubet, Jan Gasthaus, Simon Durand, and Jan van Balen for your mentorship and support throughout my internships.

I would also like to thank my colleagues, collaborators, and co-authors at the Technical University of Munich. Thank you Aleksandar Bojchevski and Oleksandr Shchur for teaching me to do research during my Master thesis and for being great colleagues and friends after I joined as a PhD student. Special thanks also to Bertrand Charpentier for many great research collaborations and for always being a source of friendliness and calmness even under stress. I would also like to thank Simon Geisler and Anna Kopetzki for being great co-authors. You all have made my PhD studies some of the best years of my life and I am grateful for that. Thanks also to the great people at the database Chair, most importantly Moritz for his support setting up our computing infrastructure.

Throughout my studies I was lucky to get to work with bright and curious students. Thank you Oliver, Tobias, Sascha, Peng, Morgane, John, Hakan, Andreas, Sven, Raphael, Johannes, Tom, Maximilian, Tim, and Franz.

I would also like to thank my family, first and foremost my parents and my brother Tobias for their unconditional support. Thank you also to my grandparents and my aunt and uncle. Thank you Stephanie for your support throughout the years and with my research stay at Stanford. I would have loved to show this thesis to Andy, too. I thank my friends for their support, most importantly Christoph, Michael, Fabian, Chris, Sarah, Max C, Max B, and Thuy. Thank you for always being there for me.

Thank you Rebecca for supporting me unconditionally throughout all my endeavors and for bringing so much love, joy, and warmth into my life; for enduring even the most stressful periods of my studies with me; for being my partner and my best friend.

This thesis is also dedicated to my dear friend Sazan who recently passed away suddenly and too soon. I am thankful for the time I got to spend with you. I miss you.

Contents

Abstract iii

Zusammenfassung v

Acknowledgements vii

Part I Introduction 1

1 Introduction 3
1.1 Machine learning on graphs . . . 3
1.2 Data corruptions . . . 4
1.3 Contributions and outline . . . 6
1.4 Own publications . . . 6

2 Background 9
2.1 Graphs . . . 9
2.2 Transductive semi-supervised learning . . . 9
2.3 Graph neural networks . . . 11
2.4 Adversarial attacks . . . 12
2.4.1 Attack characteristics . . . 13
2.4.2 Improving robustness . . . 14

Part II Adversarial Attacks on Graph Neural Networks 17

3 Adversarial Attacks on Graph Neural Networks: Perturbations and their Patterns 19
3.1 Introduction . . . 19
3.1.1 Opportunities . . . 20
3.1.2 Challenges . . . 21
3.1.3 Contributions . . . 21
3.2 Preliminaries . . . 22
3.3 Related work . . . 23
3.3.1 Deep Learning for Graphs . . . 23

3.3.2 Adversarial Attacks . . . 23
3.3.3 Generating Adversarial Perturbations . . . 24
3.3.4 Adversarial Attacks when Learning with Graphs . . . 24
3.4 Attack Model . . . 25
3.4.1 Target vs Attackers . . . 25
3.4.2 Unnoticeable Perturbations . . . 26
3.4.2.1 Graph Structure Preserving Perturbations . . . 26
3.4.2.2 Feature Statistics Preserving Perturbations . . . 27
3.5 Generating Adversarial Graphs . . . 28
3.5.1 Surrogate Model . . . 29
3.5.2 Approximate Solution . . . 29
3.5.3 Fast Computation of Score Functions . . . 30
3.5.3.1 Structural attacks . . . 30
3.5.3.2 Feature Attacks . . . 32
3.5.4 Fast Computation of Candidate Sets . . . 33
3.5.5 Complexity . . . 34
3.6 Experiments . . . 34
3.6.1 Setup . . . 34
3.6.2 Attacks on the Surrogate Model . . . 35
3.6.3 Transferability of Attacks . . . 36
3.6.3.1 Evasion vs Poisoning Attack . . . 37
3.6.3.2 Comparison . . . 37
3.6.3.3 Limited Knowledge . . . 38
3.7 Patterns of Adversarial Attacks . . . 39
3.7.1 Statistical Analysis . . . 39
3.7.2 Estimating the Importance of Patterns via Ranking . . . 41
3.7.2.1 Feature Preprocessing . . . 42
3.7.2.2 Ranking of Scores . . . 43
3.7.2.3 Relevance of Metrics for Structure Perturbations . . . 43
3.7.2.4 Relevance of Metrics for Feature Perturbations . . . 44
3.8 Scalable Attacks using Patterns in Perturbations . . . 44
3.8.1 Adversarial Ratio . . . 45
3.8.2 Complexity . . . 46
3.8.3 Experimental Evaluation . . . 46
3.8.3.1 Effectiveness . . . 46
3.8.3.2 Transferability . . . 47
3.8.3.3 Runtime . . . 49
3.9 Conclusion . . . 49
Retrospective . . . 50

4 Adversarial Attacks on Graph Neural Networks via Meta Learning 51
4.1 Introduction . . . 51
4.2 Related Work . . . 51
4.3 Problem Formulation . . . 52

4.3.1 Attack Model . . . 53
4.3.2 Overall Goal . . . 54
4.4 Graph Structure Poisoning via Meta-Learning . . . 55
4.4.1 Poisoning via Meta-gradients . . . 55
4.4.2 Greedy Poisoning Attacks via Meta Gradients . . . 56
4.4.3 Approximating Meta-Gradients . . . 57
4.4.4 Complexity analysis . . . 58
4.5 Experiments . . . 59
4.6 Conclusion . . . 64
Retrospective . . . 64

5 Adversarial Attacks on GNNs: Retrospective 67

Part III Robustness Certification of Graph Neural Networks 71

6 Certifiable Robustness and Robust Training for Graph Convolutional Networks 73
6.1 Introduction . . . 73
6.2 Related Work . . . 74
6.3 Preliminaries . . . 75
6.4 Certifying Robustness for Graph Convolutional Networks . . . 75
6.4.1 Robustness Certificates for GNNs . . . 76
6.4.2 Convex Relaxations . . . 78
6.4.3 Efficient Lower Bounds via the Dual . . . 80
6.4.4 Primal Solutions and Certificates . . . 82
6.4.5 Activation Bounds . . . 82
6.5 Robust Training of GNNs . . . 84
6.6 Experimental Evaluation . . . 86
6.6.1 Certificates: Robustness of GNNs . . . 87
6.6.2 Robust Training of GNNs . . . 89
6.7 Conclusion . . . 91
Retrospective . . . 92

7 Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations 93
7.1 Introduction . . . 93
7.2 Preliminaries . . . 94
7.3 Robustness Certification for Graph Structure Perturbations . . . 95
7.3.1 Problem Definition . . . 95
7.3.2 Optimization over the graph structure . . . 96
7.3.2.1 Induced constraints on $\hat{\mathcal{A}}$ . . . 98
7.3.3 Relaxation of the neural network . . . 102
7.3.4 Jointly Constrained Bilinear Program . . . 103
7.3.4.1 Canonical form . . . 103

7.3.4.2 Branch-and-bound algorithm . . . 104
7.3.4.3 Improvements for Graph Certificates . . . 106
7.3.4.4 Generalizing to $L > 3$ . . . 106
7.3.4.5 Alternative solution approaches . . . 106
7.4 Experiments . . . 107
7.5 Conclusion . . . 110
Retrospective . . . 110

8 Improving GNN Robustness: Retrospective 113
8.1 Heuristic methods . . . 113
8.2 Provable robustness . . . 116

Part IV Conclusion 119

9 Conclusion 121
9.1 Beyond robustness of GNNs . . . 121
9.2 Broader impact . . . 122
9.3 Open questions . . . 123

Bibliography 125

A Data and Code 151
A.1 Datasets . . . 151
A.2 Code . . . 151

B Adversarial Attacks on Graph Neural Networks: Perturbations and their Patterns 153

C Adversarial Attacks on Graph Neural Networks via Meta Learning 157

D Certifiable Robustness and Robust Training for Graph Convolutional Networks 159
D.1 Proofs . . . 159

E Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations 165
E.1 Splitting and Branching Procedure . . . 165
E.2 Experimental setup . . . 165

Part I

Introduction

1 Introduction

Artificial intelligence (AI) is widely viewed as a key technology of our times. In fact, the European Commission considers AI "as one of the most strategically important technologies of the 21st century" [71]. AI startup funding has increased more than *50-fold* between 2012 and 2020 [207]. Much of this surge in interest and applications comes from recent improvements in machine learning, more specifically deep learning. Machine learning is the study of computer programs which learn from experience (e.g., data) [157]. Deep learning is a subfield of machine learning and can be defined as models which hierarchically learn complex representations based on simpler ones [86]. In the last decade, there have been several breakthroughs in deep learning such as deep convolutional neural networks [e.g., 125, 204, 211, 100] for image recognition or Transformers for natural language processing (NLP) [e.g., 62, 188, 33].

Due to the tremendous success of deep learning in recent years, it has found many applications in science and industry. These include automatic machine translation [74], recognition of e.g., plants [190] or bird species [233] on images, neural rendering of images [214], medical image analysis [114], fluid simulation [222], or autonomous driving [90], to name only a few.

More recently, deep learning has also lead to substantial improvements across tasks in machine learning for graphs. These deep learning models are often referred to as graph neural networks (GNNs) [e.g., 195, 89, 117, 84, 97, 119] and are the main subject of study in this thesis.

1.1 Machine learning on graphs

Machine learning on graphs has a long-standing history and has roots in the field of graph mining [39] and network science [11]. More recently, deep-learning-based approaches to machine learning on graphs are also referred to as *graph representation learning* [98]. In machine learning with graphs, we try to learn from patterns in graph-structured data (such as social networks, Web graphs, supply networks, protein interaction graphs, ...) to solve tasks such as node classification, where we predict a class label for each node in a graph; community detection, where the task is to find densely connected groups of nodes in an unsupervised way; link prediction, where we predict missing or potential future edges; or graph classification, where a model predicts a class label to a graph as a whole. See, e.g., Chapter 1 in [98] for an overview of these tasks. A key difference to 'standard' machine learning is that, on graphs, one of the fundamental assumptions, i.e., that all samples are *independent and identically distributed* (i.i.d. assumption), does not hold. On the other hand, this also enables us to leverage the information of a node's neighborhood to improve its prediction, something we cannot do in the traditional scenario.

Traditional methods for machine learning on graphs follow similar approaches as their non-graph counterparts before the advent of deep learning [98]. Typically, we first extract a set of statistics or features from a given graph, e.g., occurrence counts of certain graph motifs or the node degree distribution. Then, these features are used in traditional downstream models such as logistic regression or kernel-based approaches. In Chapter 2 of [98], readers can find an overview of traditional graph ML models.

Graph neural networks (GNNs). In roughly the last five years, graph neural networks (GNNs) have brought deep learning to the graph domain and rapidly set the state of the art in many graph-related machine learning tasks [106], even though the general GNN framework was developed already in the mid-to-late 2000s [89, 195]. From early on, GNNs have also been considered to be useful building blocks to improve ML systems in other fields such as reinforcement learning (RL) [e.g., 99, 256, 7] or computer vision [e.g., 48, 12, 186, 80]. In addition, GNNs have been applied to a variety of domains and applications. A non-exhaustive list of examples includes traffic forecasting [241, 202, 138, 263, 95]; molecular property prediction [84, 121, 120, 118, 199, 223]; recommendation in Web graphs [26, 252]; knowledge graphs [250, 229]; pharmacy and drug discovery [109, 29, 113, 272]; travel time prediction in routing [61]; combinatorial optimization [139, 37]; autonomous driving [63, 112]; and source code and programming languages [102, 279, 76, 5]. Recently, Hu et al. [106] created the 'Open Graph Benchmark' (OGB), which includes several real-world datasets from domains such as source code, molecules, or citation graphs. It further includes different tasks such as node classification, link prediction, or graph classification, and the graphs come in different scales of up to 100 million nodes. In Sections 2.1 to 2.3 we provide some background on machine learning for graphs and GNNs.

1.2 Data corruptions

Real-world data is never perfect: sensors cannot measure physical quantities to arbitrary precision; they could malfunction and record incorrect values. Humans can also be the cause for imperfect data: they sometimes make mistakes when manually inserting data; they do not always fill all the fields (e.g., leave some questions in a survey unanswered); they could make a mistake recording the label of a data point; or they could even deliberately create false or misleading data (e.g., by lying on a survey). Thus, any algorithm learning from real-world data is exposed to some degree of data imperfection. Often, this is not a problem, e.g., the sensor noise on daylight photos taken with modern digital cameras is very small. In general, though, noisy or incomplete data can degrade real-world performance of machine learning models.

Hendrycks and Dietterich [103] study the effect of typical data imperfections on natural images (e.g., out-of-focus or motion blur, brightness or contrast changes) on deep learning models for image classification. They find that neural networks' classification performance drops substantially when adding data corruptions at levels where humans

can typically still correctly classify the images. Thus, creating models which are robust to real-world noise and corruptions is an active field of research.

Adversarial perturbations are *worst-case* data corruptions in the sense that small perturbations, often imperceptible to humans, completely alter a model's prediction.

Deep neural networks are remarkably vulnerable to adversarial perturbations, i.e., small perturbations, imperceptible to humans, added to images to mislead a machine learning model [e.g., 212, 88, 177, 192, 91, 179, 132, 170, 140, 162, 17, 38]. It is even possible to create *physical* adversarial examples where, for instance, stickers attached to a stop sign lead it to be classified as a different sign by a neural network [72], highlighting the real-world harm that could be done by exploiting vulnerabilities in machine learning models. Further, adversarial examples shine a spotlight on gaps in our understanding of neural networks: neural networks are hypothesized to learn meaningful representations that capture semantic understanding of the domain and task [134]. Adversarial examples are counterexamples to this hypothesis: the semantic content of the samples is unchanged but the network is fooled.

There are many possible reasons why adversarial examples could be created in the real world: e.g., they could be used to fool a financial assessment system in order to be granted a loan; to make fake or bot accounts look more natural so that they are not blocked; or humans could try to protect their privacy from mass surveillance systems by wearing 'adversarial' hats [123], glasses [203], or face stickers [96]. Thus, the study of adversarial examples and potential defenses [e.g., 16, 53, 88, 180, 176, 248, 101, 156, 234, 51, 55, 194, 189, 146, 158] against them has become a very active field of research.

Adversarial robustness of GNNs. In this thesis we study the adversarial robustness of graph neural networks. Before the initial study presented in this thesis, the robustness of GNNs w.r.t. adversarial perturbations was not known. Because nodes are not independent but connected to other nodes in graphs, adversarial attacks on GNNs bring unique challenges and opportunities. For instance, we can attack a node's prediction without changing its features and by only perturbing nodes in its neighborhood; this cannot be done on traditional independent data such as images. Moreover, we can influence a GNN's prediction of a node by inserting or removing edges, which are discrete operations as opposed to the typically continuous-valued adversarial perturbations on images.

In addition, adversarial attacks on GNNs are highly relevant in practice: as GNNs are rapidly adopted in industry [e.g., 252, 26], their typical application domain are Web graphs and social networks. Here, adversarial actors are omnipresent, and billions of users are potentially affected by vulnerabilities. Thus, understanding and improving the robustness of GNNs is important to ensure their safe application in the real world. For a recent overview of GNN robustness we refer the reader to [94].

Ch.	Ref.	Title	Venue	Project page
3	[273], [275]	Adversarial Attacks on Graph Neural Networks: Perturbations and their Patterns	KDD 2018, TKDD 2020	/nettack/
4	[276]	Adversarial Attacks on Graph Neural Networks via Meta Learning	ICLR 2019	/gnn_meta_attack/
6	[277]	Certifiable Robustness and Robust Training for Graph Convolutional Networks	KDD 2019	/robust_gcn/
7	[278]	Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations	KDD 2020	/robust_gcn/

Table 1.1: List of own publications which form the basis of this thesis. The project pages are available at `https://www.daml.in.tum.de/[project]`.

1.3 Contributions and outline

We first establish preliminaries and background information in Chapter 2. In Part II, we present the first study of adversarial robustness of graph neural networks. We develop different attack strategies such as edge manipulation or feature perturbation to change a GNN's prediction. At the same time, we explore ways to ensure that the adversarial perturbations are subtle and hard to detect. We study both local attacks on individual nodes as well as global attacks aiming to decrease the overall classification performance of GNNs. In summary, the first two studies presented in this thesis establish that GNNs are not robust w.r.t. adversarial perturbations and highlight the need to develop methods to improve GNN robustness. We investigate this in Part III, where we present two more studies. In the first one we develop the first robustness certification technique for message-passing GNNs w.r.t. perturbations of the node features. In addition, we use this certification method to propose a robust training procedure for GNNs, which substantially increases the robustness of GNNs. In the second study, we consider perturbations of the graph structure and develop a robustness certification method for these attacks. At the ends of Parts II and III, respectively, we offer retrospective insight regarding the studies presented in this thesis and discuss how the research field of studying the robustness of GNNs has evolved since our first study. Finally, in Part IV, we conclude this thesis by highlighting open questions as suggestions for future research directions.

1.4 Own publications

Chapters 3, 4, 6 and 7 contain material previously published at international peer-reviewed conferences. In Table 1.1 we provide references to these publications. At the respective project pages we provide additional material such as code, presentation slides, and posters. Pointers to the code implementations can be found in Appendix A.2

Besides these publications, the author was involved in a number of additional projects. A full list of all publications with the author's involvement during the period of his PhD studies is provided in the following:

[8] Angriman, E., van der Grinten, A., Bojchevski, A., Zügner, D., Günnemann, S., and Meyerhenke, H. (2020). Group Centrality Maximization for Large-scale Graphs. In *2020 Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 56–69. SIAM.

[28] Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. (2018). NetGAN: Generating Graphs via Random Walks. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 610–619. PMLR.

[42] Charpentier, B., Borchert, O., Zügner, D., Geisler, S., and Günnemann, S. (2022). Natural Posterior Network: Deep Bayesian Predictive Uncertainty for Exponential Family Distributions. In *International Conference on Learning Representations (ICLR)*.

[43] Charpentier, B., Zügner, D., and Günnemann, S. (2020). Posterior Network: Uncertainty Estimation without OOD Samples via Density-Based Pseudo-Counts. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[66] Elflein, S., Charpentier, B., Zügner, D., and Günnemann, S. (2021). On Out-of-distribution Detection with Energy-based Models. In *ICML Workshop on Uncertainty & Robustness in Deep Learning*.

[81] Geisler, S., Schmidt, T., Şirin, H., Zügner, D., and Günnemann, S. (2021). Robustness of Graph Neural Networks at Scale. In Ranzato, M., Beygelzimer, A., Nguyen, K., Liang, P., Vaughan, J., and Dauphin, Y., editors, *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc.

[83] Geisler, S., Zügner, D., and Günnemann, S. (2020). Reliable Graph Neural Networks via Robust Aggregation. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[124] Kopetzki, A.-K., Charpentier, B., Zügner, D., Giri, S., and Günnemann, S. (2021). Evaluating Robustness of Predictive Uncertainty Estimation: Are Dirichlet-based Models Reliable? In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5707–5718. PMLR.

[206] Stadler, M., Charpentier, B., Geisler, S., Zügner, D., and Günnemann, S. (2021). Graph Posterior Network: Bayesian Predictive Uncertainty for Node Classification. In Ranzato, M., Beygelzimer, A., Nguyen, K., Liang, P., Vaughan, J., and Dauphin, Y., editors, *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc.

[273] Zügner, D., Akbarnejad, A., and Günnemann, S. (2018). Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 2847–2856. ACM.

[274] Zügner, D., Aubet, F.-X., Satorras, V. G., Januschowski, T., Günnemann, S., and Gasthaus, J. (2021a). A Study of Joint Graph Inference and Forecasting. In *ICML Time Series Workshop*.

[275] Zügner, D., Borchert, O., Akbarnejad, A., and Günnemann, S. (2020). Adversarial Attacks on Graph Neural Networks: Perturbations and Their Patterns. *ACM Transactions on Knowledge Discovery from Data*, 14(5).

[13] Zügner, D., Charpentier, B., Geringer, S., Ayle, M., and Günnemann, S. (2022). End-to-End Learning of Probabilistic Hierarchies on Graphs. In *International Conference on Learning Representations (ICLR)*.

[276] Zügner, D. and Günnemann, S. (2019a). Adversarial Attacks on Graph Neural Networks via Meta Learning. In *International Conference on Learning Representations (ICLR)*.

[277] Zügner, D. and Günnemann, S. (2019b). Certifiable Robustness and Robust Training for Graph Convolutional Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 246–256. ACM.

[278] Zügner, D. and Günnemann, S. (2020). Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 1656–1665. ACM.

[279] Zügner, D., Kirschstein, T., Catasta, M., Leskovec, J., and Günnemann, S. (2021b). Language-Agnostic Representation Learning of Source Code from Structure and Context. In *International Conference on Learning Representations (ICLR)*.

2 Background

In this section we provide some background on the contents of this thesis. In order to preserve the original storyline of the respective publications, we kept the background sections of the publications in Chapters 3, 4, 6 and 7; thus, some parts might be repeated in the later chapters. Readers are encouraged to skip these sections whenever they occur.

2.1 Graphs

We define an attributed graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{X})$, where $\mathcal{V}$ is the set of nodes, $N = |\mathcal{V}|$ is the number of nodes in the graph, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ the set of edges, and $\mathcal{X} : \mathcal{V} \to \mathcal{D}$ the node attributes. In much of this thesis, we consider graphs where the node attributes are binary feature vectors, i.e., we instantiate $\mathcal{X} : \mathcal{V} \to \{0,1\}^D$, where D is the feature dimension.

To work with and manipulate a graph for use in machine learning, we find it useful to represent a graph as $G = (\boldsymbol{A}, \boldsymbol{X})$, where $\boldsymbol{A} \in \{0,1\}^{N \times N}$ is the adjacency matrix representing the connections, i.e., $\boldsymbol{A}_{ij} = 1$ iff $(i,j) \in \mathcal{E}$. In this thesis we consider unweighted and undirected graphs only. $\boldsymbol{X}$ is the node attribute matrix, i.e., each row $\boldsymbol{X}_i \in \mathcal{D}$ contains the attribute vector of the corresponding node. Thus, $\boldsymbol{X}^T \in \mathcal{D}^N$, e.g., $\boldsymbol{X} \in \{0,1\}^{N \times D}$ in the case of binary feature vectors. Here, we assume w.l.o.g. an arbitrary bijection of integer IDs to nodes in the graph.

The graphs we consider in this thesis, as defined above, are *homogeneous*. That is, there is only a single type of nodes and edges in a graph. This is in contrast to *heterogeneous* graphs, where there are different node types in a graph, e.g., users and products in user-product graphs, or different edge types such as different atom bond types in molecular graphs.

2.2 Transductive semi-supervised learning

The most important task we consider in this thesis is semi-supervised transductive node classification in graphs [41]. In *supervised learning*, we learn a function $f : \mathcal{X} \to \mathcal{Y}$ from datapoints to targets. That is, we need a set of labeled datapoints $\{(x_i, y_i)\}_{i=1}^N$, $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$ to approximate the true underlying mapping function. Commonly the targets are either continuous-valued or categorical, corresponding to the tasks of *regression* and *classification*, respectively.

In *unsupervised learning*, on the other hand, we only have access to a set of N datapoints $\{x_i\}_{i=1}^N$, $x_i \in \mathcal{X}$ sampled from the distribution $p(x)$. The goal is to find interesting and/or useful structure in the data distribution (e.g., using K-means) or to learn a

model that can (approximately) generate additional samples from $p(x)$. Prominent and recent examples for the latter *generative* perspective are generative adversarial networks (GANs) [e.g., 87, 270, 45, 54, 9, 32] (Bojchevski et al. [28] propose a GAN specifically for graphs), as well as language models in natural language processing (NLP), most notably GPT-2 [188] and GPT-3 [33].

Semi-supervised learning sits between *unsupervised* and *supervised* learning. Here, we have a set of N datapoints, and for a number L of these samples we know their true targets. More formally, we can decompose the dataset $\mathcal{V}$ into the (non-overlapping) labeled $\mathcal{V}_L$ and unlabeled parts $\mathcal{V}_U$, i.e., $\mathcal{V} = \mathcal{V}_U \cup \mathcal{V}_L, \mathcal{V}_U \cap \mathcal{V}_L = \emptyset$. The unlabeled datapoints are defined as $\mathcal{V}_U = \{x_i\}_{i=L+1}^{N}$, $x_i \in \mathcal{X}$, and the labeled datapoints as $\mathcal{V}_\mathcal{L} = \{(x_i, y_i)\}_{i=1}^{L}$, $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$. Typically, only a small fraction of data points are labeled, i.e., $L \ll N$. Put differently: we use information about $p(x)$ (the data distribution) to improve our model of $p(y|x)$ (the mapping from data to targets). In doing so we make a set of assumptions, most notably the *cluster assumption* [41], which states that data points from the same cluster should belong to the same class y. An illustrative example is to imagine learning an unsupervised model such as K-means. By the clustering assumption, we can leverage the labeled datapoints to assign each unlabeled datapoint the majority class for labeled datapoints in the same cluster. For more details, we refer the reader to Chapter 1 of [41]. In Chapter 4 of the same book, the authors further discuss scenarios in which semi-supervised learning can fail.

Transductive learning is the counterpart of *inductive learning*. In inductive learning, we learn a function $f : \mathcal{X} \to \mathcal{Y}$ which can predict the target for *any* potential datapoint $x_i \in \mathcal{X}$. For example, an image classification model should be able to classify any picture, even those that did not exist when we trained the model. In transductive learning, on the other hand, we directly estimate the labels of a set of test datapoints $\mathcal{V}_U$; this set of datapoints is fixed, so we cannot classify additional datapoints after the model has been trained [41]. This setting is typical in machine learning for graphs, where we often assume the graph is fixed and want to classify the unlabeled nodes given a small set of labeled nodes. We again refer to Chapter 1 of [41] for an excellent introduction to transductive learning.

Node classification is a traditional application of semi-supervised transductive learning and the main task considered in this thesis. For this task, we are provided a single attributed graph $G = (\boldsymbol{A}, \boldsymbol{X})$ as well as a subset $\mathcal{V}_L \subseteq \mathcal{V}$ of labeled nodes, and want to learn a function $g : \mathcal{V} \to \mathcal{C}$ which maps each node $v \in \mathcal{V}$ to one class in $\mathcal{C}$. Typically, models leverage the unlabeled nodes for learning under the *homophily assumption* that nodes from the same class tend to connect more densely than nodes from different classes [153]. A prominent recent class of models for node classification are graph neural networks, which are briefly explained in the following section.

2.3 Graph neural networks

The first graph neural networks (GNNs) date back to 2005 and the work of Gori et al. [89], which was later refined by Scarselli et al. [195]. In 2017, GNNs had important breakgthroughs in node classification [117, 97] and molecular property prediction [84].

General framework. In this thesis we focus on message-passing neural networks [84] which can be formalized with the framework used by Hamilton [98]. In this framework, a GNN alternates between feature processing on the node level and aggregation of neighbor representations, followed by an element-wise activation function:

$$
\begin{aligned}
&\boldsymbol{h}_u^{(1)} := \boldsymbol{x}_u \\
&\hat{\boldsymbol{h}}_u^{(l)} = f^{(l-1)}(\boldsymbol{h}_u^{(l-1)}) && \text{for } l = 2, \dots, L \\
&\boldsymbol{h}_u^{(l)} = \sigma^{(l)}\left(\text{AGGREGATE}^{(l)}\left(\left\{\left(\boldsymbol{A}_{uv}, \hat{\boldsymbol{h}}_v^{(l)}\right)\right\}, v \in \mathcal{N}(u) \cup \{u\}\right)\right) && \text{for } l = 2, \dots, L
\end{aligned}
$$

where $\mathcal{N}(u)$ denotes the set of neighbors of node u, $\sigma^{(l)}$ the activation function at layer l, and $f^{(l-1)} : \mathbb{R}^{D^{(l-1)}} \to \mathbb{R}^{D^{(l)}}$ some function, e.g., a linear mapping or a neural network. L denotes the number of layers of the neural network (including the input layer). $\text{AGGREGATE}^{(l)}(\cdot)$ is the function that maps from the set of neighbors of node u (including u itself) and their representation to the single vector representation of node u for the next layer. Since at each layer we aggregate information from a node's neighbors to update its representation, a GNN as defined above with L layers has a receptive field of $L-1$. This means that we can compute the output representation of any node using only the induced subgraph of its $L-1$-hop neighborhood, i.e., the graph obtained when only keeping nodes that can be reached within $L-1$ hops as well as the edges between the selected nodes. Next, we show how the graph neural network GCN instantiates this framework.

Graph convolutional network (GCN) [117]. In GCN, $f^{(l)}(\boldsymbol{h}_u^{(l)})$ is defined as $f^{(l)}(\boldsymbol{h}_u^{(l)}) := \boldsymbol{h}_u^{(l)T} \cdot \boldsymbol{W}^{(l)}$, i.e. a simple linear map per layer. The aggregation function is defined as

$$
\text{AGGREGATE}^{(l)}\left(\left\{\left(\boldsymbol{A}_{uv}, \hat{\boldsymbol{h}}_v^{(l)}\right)\right\}, v \in \mathcal{N}(u) \cup \{u\}\right) = \sum_{v \in \mathcal{N}(u) \cup \{u\}} \hat{\boldsymbol{A}}_{uv} \hat{\boldsymbol{h}}_v^{(l)} + \boldsymbol{b}^{(l)}, \quad (2.1)
$$

where $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-1/2} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-1/2}$ and $\boldsymbol{b}^{(l)}$ is a vector of learnable biases. Here, $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is a modified adjacency matrix with and $\tilde{\boldsymbol{D}}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$ the diagonal matrix of node degrees of the modified graph. Equivalently, we can express the GCN forward pass in matrix form:

$$
\begin{aligned}
&\boldsymbol{H}^{(1)} = \boldsymbol{X} \\
&\hat{\boldsymbol{H}}^{(l)} = \boldsymbol{H}^{(l-1)} \boldsymbol{W}^{(l-1)} && \text{for } l = 2, \dots, L \\
&\boldsymbol{H}^{(l)} = \sigma^{(l)}\left(\hat{\boldsymbol{A}} \hat{\boldsymbol{H}}^{(l)} + \boldsymbol{b}^{(l)}\right) && \text{for } l = 2, \dots, L
\end{aligned}
$$

In GCN, all layers' activation functions except the last layer's is the rectified linear unit, i.e., $\sigma^{(l)}(\boldsymbol{Z}_{ij}) = \mathrm{ReLU}(\boldsymbol{Z}_{ij}) = \max\{\boldsymbol{Z}_{ij}, 0\}, 2 \leq l < L$, and $\sigma^{(L)}(\cdot) = \mathrm{softmax}(\cdot)$.

Extensions. The framework above is a simplification of the message-passing neural network framework introduced by Gilmer et al. [84]. Many other GNNs such as GAT [224], GraphSAGE [97], (A)PPNP [119], SGC [236], PPRGo [26], and CLN [184], are instantiations of the framework above. The general framework [84] also includes GNNs which take into account attributes on the edges of the graph [e.g., 121, 118, 199, 223, 84], which however is not necessary for the content of this thesis. Further, there are numerous variants of GNNs such as position-aware GNNs [254], GNNs with hierarchical pooling of the graph [253], or GNNs for causal inference [257]. In addition, some GNNs can scale to massive graphs with billions of nodes [e.g., 26, 228, 252, 49]. GNNs have also been extended to heterogeneous graphs [e.g., 269, 271, 196].

The aggregation functions of many GNNs effectively act as low-pass filters on the graph structure [50, 269]. Thus, typical GNNs implicitly follow the *homophily assumption* that nodes from the same class tend to connect more densely than nodes from different classes [153]. Even more, pre-filtering the adjacency matrix with a low-pass filter [122] tends to improve results of many GNNs on homophilic datasets. Chien et al. [50], Zhu et al. [269] propose GNNs which are suited for non-homophilic datasets.

Another stream of work concerns itself with the theoretical expressivity of graph neural networks [246, 164, 73, 31, 163, 151]. Xu et al. [246], for instance, show that message-passing GNNs' expressive power of detecting graph isomorphism is bounded by the 1-Weisfeiler-Lehman test [131].

Finally, we want to mention that we only consider graphs where both the structure and the attributes are static and do not change over time. Dynamic GNNs [e.g., 191, 205, 243] are GNN variants which take into account settings where the graph changes over time. For a comprehensive overview on GNNs, we refer the reader to [240] or [238].

2.4 Adversarial attacks

Adversarial examples contain small, *worst-case* perturbations that mislead a machine learning model while looking (almost) unchanged to a human [212]. Adversarial examples have been studied since at least 2004 [57] and w.r.t. image data since at least 2006 [85]. However, in the latter work on images, the adversarial examples crafted by the method create semantic ambiguity also for humans. This is in contrast with the adversarial examples on deep neural networks typically studied since the work of [212] in 2013. A remarkable feature of these adversarial examples is that they look indistinguishable to the non-perturbed images for humans. Figure 2.1 shows an example of this: the left image is classified correctly as "pig" by a deep neural network, while the right one – indistinguishable from the left picture for a human – is labeled as "airliner" by the model.

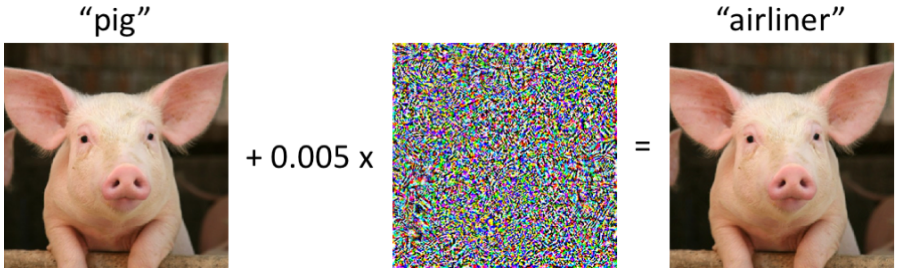


Figure 2.1: An adversarial example. Figure from Madry and Schmidt [147].

2.4.1 Attack characteristics

Biggio and Roli [19] provide an overview of the different dimensions of adversarial attack scenarios; we loosely follow their structure here.

Attacker's goal. This encompasses which violation the attacker aims to cause (e.g., to obtain private data, to disrupt system operations or to stay undetected while manipulating certain predictions); the specificity of the attack, i.e., whether the goal is to misclassify a specific target datapoint (referred to as *local* attacks in this thesis) or to degrade the overall performance of a model (referred to as *global* attacks in this thesis); and whether or not the desired misclassification error is specific, i.e., whether the goal is to have a sample be misclassified as some *specific* target class (otherwise we call the attack *generic*).

Attacker's knowledge. Here we capture the attacker's knowledge of the system under attack, including the training data, the model architecture, the model parameters or hyperparameters, the loss function, and the optimization algorithm. It is standard to distinguish three cases: **(i)** white-box attacks in which the knowledge of the attacker is unrestricted; **(ii)** grey-box attacks, where the attacker has some limited knowledge of the system; and **(iii)** black-box attacks assuming no knowledge of the system.

Attacker's capability. Here we describe the attacker's power to manipulate the data. An important distinction for this thesis is between *poisoning* (also known as *causative*) attacks and *evasion* (also known as *exploratory*) attacks. In poisoning attacks, the attacker can modify the training data of the machine learning model to some extent, with the goal of leading to poor classification performance after training or to have certain samples be misclassified by the trained model. Evasion attacks are the more commonly studied case. Here, the attacker modifies samples from the test set in order for them to be misclassified by a fixed, trained model.

Moreover, the attacker is typically limited in the amount of manipulation they can perform, e.g., because they have limited access or because they need to avoid detection. This is typically modeled by a constraint ensuring that a perturbed datapoint $\hat{\boldsymbol{x}}$ is within

a radius r around the original data point $\boldsymbol{x}$, measured by some L_p norm:

$$\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|_p \leq r. \tag{2.2}$$

Moreover, we often assume that within some small radius r the *semantic* meaning of a data sample does not change and that (for images) the perturbations are imperceptible to humans [212].

Attack strategy. This is the actual procedure how the attacker produces adversarial examples. Given the attacker's knowledge and capabilities, they optimize an objective function which reflects their goal. In a white-box attack, for instance, the attacker could optimize the opposite of the loss function $\mathcal{L}$ used to train the model [88]:

$$\max_{\hat{\boldsymbol{x}}} \mathcal{L}(\hat{\boldsymbol{x}}, y) \qquad \text{s.t. } \|\hat{\boldsymbol{x}} - \boldsymbol{x}\|_p \leq r.$$

2.4.2 Improving robustness

A large number of methods to 'defend' neural networks against adversarial examples, i.e., to improve the models' robustness, exist. We can broadly group them into two categories: methods to improve the empirical robustness and methods for improved certified robustness. A detailed introduction to these is out of scope for this thesis; the notes of lectures 9-14 by Li [135] are an excellent resource for interested readers.

Improving empirical robustness. Here, the goal is to improve the robustness of a model in the sense of making it unlikely that an attack succeeds. Methods are typically based on heuristics or ad-hoc approaches against specific attacks. They do not come with a guarantee that they prevent *all* possible adversarial attacks within some allowed constraint set; nonetheless, they can lead to improved provable robustness properties. Still, most empirical 'defenses' have later been broken by stronger attacks [133].

One notable method to improve empirical robustness is *adversarial training* [88]. While the model is being trained, we take the role of the adversary to find an adversarial example of the training sample, and compute the loss on this perturbed example instead. Adversarial training is an instance of the more general paradigm of robust training. In robust training, instead of performing standard model training as

$$\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}, y \in \mathcal{D}_{\text{train}}} \mathcal{L}(\boldsymbol{x}, y),$$

we minimize a *robust loss*, e.g., the worst-case loss

$$\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}, y \in \mathcal{D}_{\text{train}}} \max_{\|\boldsymbol{\epsilon}\|_p \leq r} \mathcal{L}(\boldsymbol{x} + \boldsymbol{\epsilon}, y).$$

In practice, determining the worst perturbation, i.e., $\max_{\|\boldsymbol{\epsilon}\|_p \leq r} \mathcal{L}(\boldsymbol{x}+\boldsymbol{\epsilon}, y)$ is intractable. Instead, as a heuristic, we plug in some existing adversarial attack algorithm which returns a perturbed training example $\hat{\boldsymbol{x}}$:

$$\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{x}, y \in \mathcal{D}_{\text{train}}} \mathcal{L}(\hat{\boldsymbol{x}}, y).$$

Adversarial training is known as one of the few heuristic methods which consistently lead to improved robustness of the trained model. One exception are graph neural networks, where previous attempts at implementing adversarial training have been less fruitful, as we will discuss in Chapter 5.

Adversarial training leads to classifiers which tend to be more robust, i.e., less affected by adversarial examples. Another conceivable strategy could be to try to detect adversarial examples. The motivation is that adversarial examples come from a different distribution than the "natural" data; hence, we could try to detect adversarial examples via outlier detection or uncertainty estimation methods. However, this does not appear to work in practice [133]. Even more, the uncertainty-based detection model can itself be adversarially attacked [124]. In addition, Kopetzki et al. [124] argue that since adversarial perturbations are imperceptible to humans and do not change the semantic meaning of samples, we would intuitively expect machine learning classifiers to also be robust or even invariant to these perturbations.

Provable robustness methods provide guarantees that the predicted class of a sample does not change within a radius $\hat{r}$ measured by some norm. Thus, we can be certain that there is no adversarial example in the vicinity of the respective sample and do not need to worry that some stronger attack developed in the future may break our certificate.

Exact verification [e.g., 218], i.e., proving or disproving that the model's prediction for a sample does not change within some radius, is NP-hard in general [134]. Thus, this is infeasible for large models.

Verification methods using convex relaxation [e.g., 234, 189], on the other hand, can be run in polynomial time. The price we pay is that the methods cannot prove or disprove robustness on all samples, i.e., sometimes it cannot make a decision. On the other hand, if the verification states that the model is robust, we can be sure that there are no adversarial examples. The methods we present in Part III fall into this category.

A third category which emerged relatively recently is randomized smoothing [53]. Here, we define the *smoothed* classifier to be the expected output of the model (called the *base* classifier) under some noise distribution, e.g., Gaussian noise. This smoothed classifier has provable Lipschitz-smoothness properties, which can be exploited to guarantee that a prediction cannot change within some radius. However, since we do not have access to the smoothed classifier directly, we need to estimate it via Monte Carlo samples; as a consequence, randomized smoothing leads to probabilistic certificates, i.e., guaranteed to hold with some high probability. A big advantage of randomized smoothing methods is that they can be applied to *any* model. Hence, randomized smoothing attracted a lot of attention recently, leading to many extensions of the method [e.g., 193, 130, 126], including an extension to discrete data such as graphs [25].

Part II

Adversarial Attacks on Graph Neural Networks

3 Adversarial Attacks on Graph Neural Networks: Perturbations and their Patterns

3.1 Introduction

One of the most frequently addressed tasks on graph data is *node classification*: given a single large (attributed) graph and the class labels of a few nodes, the goal is to predict the labels of the remaining nodes. For example, one might wish to classify the role of a protein in a biological interaction graph [97], predict the customer type of users in e-commerce networks [70], or assign scientific papers from a citation network into topics [117].

While many classical approaches have been introduced in the past to tackle the node classification problem [145, 41], the last years have seen a tremendous interest in methods for *deep learning on graphs* [22, 161, 36]. Specifically, approaches from the class of graph convolutional networks [117, 184] have achieved strong performance in many graph-learning tasks including node classification.

The strength of these methods — beyond their non-linear, hierarchical nature — relies on their use of the graphs' relational information to perform classification: instead of only considering the instances individually (nodes and their features), the relationships between them are exploited as well (the edges). Put differently: the instances are not treated independently; we deal with a certain form of non-i.i.d. data where network effects such as homophily [145] support the classification.

However, there is one big caveat: Many researchers have noticed that deep learning architectures for classical learning tasks can easily be fooled/attacked [212, 88] . Even only slight, deliberate perturbations of an instance — also known as *adversarial perturbations/examples* — can lead to wrong predictions. Such negative results significantly hinder the applicability of these models, leading to unintuitive and unreliable results, and they additionally open the door for attackers that can exploit these vulnerabilities. Recent works show that graph neural networks are also vulnerable to adversarial attacks [273, 276]. This is highly critical, since especially in domains where graph-based learning is used (e.g. the web) adversaries are common and false data is *easy to inject*: spammers add wrong information to social networks; fraudsters frequently manipulate online reviews and product websites [104].

In this chapter, we propose a method to create adversarial examples for graph neural networks. In addition, we take a step towards answering the question of *what makes adversarial attacks on graph neural networks so effective?* Compared to adversarial

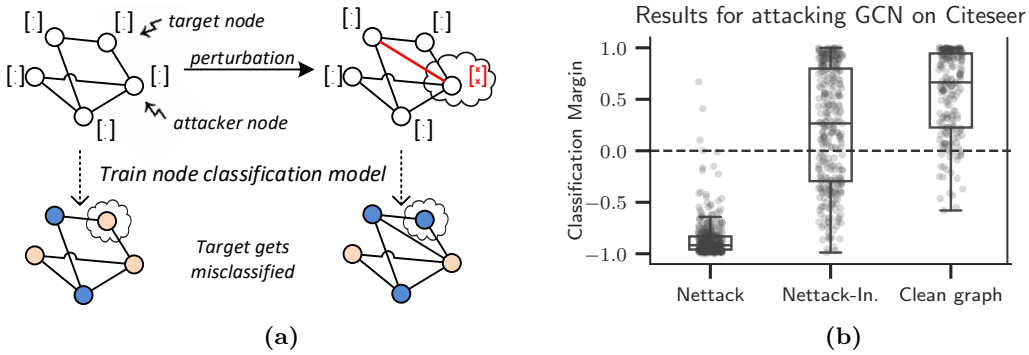


Figure 3.1: Small perturbations of the graph structure and node features lead to misclassification of the target. **(a)** Attack overview; **(b)** accuracy degradation on CITESEER.

attacks on deep networks for continuous, i.i.d. data (e.g., images), our work significantly differs in various aspects.

3.1.1 Opportunities

(1) Since we are operating on an attributed graph, adversarial perturbations can manifest in two different ways: by changing the nodes' features or the graph structure. Manipulating the graph, i.e., the dependency structure between instances, has not been studied so far, but is a highly likely scenario in real-life. For example, one might add or remove (fake) friendship relations to a social network. (2) While existing works were limited to manipulating an instance itself to enforce its wrong prediction[1], the relational effects give us more power: by manipulating one instance, we might specifically misguide the prediction for another instance. Again, this scenario is highly realistic. Think about a fraudster who hijacks some accounts, which he then manipulates to enforce a wrong prediction for *another* account they have *not* under control. Thus, in graph-based learning scenarios we can distinguish between (i) nodes which we aim to misclassify, called *targets*, and (ii) nodes which we can directly manipulate, called *attackers*. Figure 3.1 illustrates the goal of our work and shows the result of our method on the CITESEER network. Clearly, compared to classical attacks to learning models, graphs enable much richer potential for perturbations. But likewise, constructing them is far more challenging. (3) Uncovering the underlying patterns of what makes our adversarial attacks so harmful to graph neural networks is a major step towards detecting and preventing attacks, a problem that has yet to be solved. In short, we leap towards making graph neural networks safe for use in real-world applications.

[1]Due to the independence assumption, a misclassification for instance i can only be achieved by manipulating instance i itself for the commonly studied evasion (test-time) attacks. For the less studied poisoning attacks we might have indirect influence.

3.1.2 Challenges

(1) Unlike, e.g., images consisting of continuous features, the graph structure — and often also the nodes' features — is discrete. Therefore, gradient based approaches [88, 155] for finding perturbations are not suited. How to design efficient algorithms that are able to find adversarial examples in a discrete domain? (2) Adversarial perturbations are aimed to be unnoticeable (by humans). For images, one often enforces, e.g., a maximum deviation per pixel value. How can we capture the notion of 'unnoticeable changes' in a (binary, attributed) graph? (3) Last, node classification is usually performed in a *transductive* learning setting. Here, the train and test data are used jointly to learn a new classification model before the predictions are performed on the specific test data. This means that the predominantly performed evasion attacks — where the parameters of the classification model are assumed to be static — are not realistic. The model has to be (re)trained on the manipulated data. Thus, graph-based learning in a transductive setting is inherently related to the challenging poisoning/causative attacks [18].

3.1.3 Contributions

Given these challenges, we propose a principle for adversarial perturbations of attributed graphs that aim to fool state-of-the art deep learning models for graphs. In particular, we focus on semi-supervised classification models based on graph convolutions such as GCN [117] and Column Network (CLN) [184] — but we will also showcase our method's potential on the unsupervised model DeepWalk [182]. By default, we assume an attacker with knowledge about the full data (but only a small fraction of class labels), which can, however, only manipulate parts of it. This assumption ensures reliable vulnerability analysis in the worst case. But even when only parts of the data are known, our attacks are still successful as shown by our experiments. Overall, our contributions are:

- *Model:* We propose a model for adversarial attacks on attributed graphs considering node classification. We introduce new types of attacks where we explicitly distinguish between the attacker and the target nodes. Our attacks can manipulate the graph structure and node features while ensuring unnoticeable changes by preserving important data characteristics (e.g., degree distribution, co-occurrence of features).
- *Algorithm:* We develop an efficient algorithm NETTACK for computing these attacks based on linearization ideas. Our method enables incremental computations and exploits the graph's sparsity for fast execution.
- *Experiments:* We show that our model can dramatically worsen classification results for the target nodes by only requiring few changes to the graph. We furthermore show that these results transfer to other established models, hold for various datasets, and even work when only parts of the data are observed. Overall, this highlights the need to handle attacks to graph data.
- *Patterns:* For the first time, we present a study of the patterns of adversarial perturbations on graph neural networks. We identify statistically significant patterns

in adversarial perturbations using hypothesis testing. We use these findings to introduce FASTTACK, a highly scalable adversarial attack algorithm. FASTTACK successfully exploits the patterns we identify using hypothesis testing to create adversarial perturbations that are effective across models and datasets. This highlights the expressiveness and generality of the regularities we uncover in adversarial perturbations and represents a first major step towards being able to detect and prevent adversarial attacks on graph neural networks.

3.2 Preliminaries

We consider the task of (semi-supervised) node classification in a single large graph having binary node features. Formally, let $G = (\boldsymbol{A}, \boldsymbol{X})$ be an attributed graph, where $\boldsymbol{A} \in \{0,1\}^{N \times N}$ is the adjacency matrix representing the connections and $\boldsymbol{X} \in \{0,1\}^{N \times D}$ represents the nodes' features. We denote with $\boldsymbol{x}_v \in \{0,1\}^D$ the D-dim. feature vector of node v. W.l.o.g. we assume the node-ids to be $\mathcal{V} = \{1, \ldots, N\}$ and the feature-ids to be $\mathcal{F} = \{1, ..., D\}$.

Given a subset $\mathcal{V}_L \subseteq \mathcal{V}$ of labeled nodes, with class labels from $\mathcal{C} = \{1, 2, \ldots, c_K\}$, the goal of node classification is to learn a function $g : \mathcal{V} \to \mathcal{C}$ which maps each node $v \in \mathcal{V}$ to one class in $\mathcal{C}$.[2] Since the predictions are done for the *given* test instances, which are already known before (and also used during) training, this corresponds to a typical *transductive* learning scenario [41].

In this chapter, we focus on node classification employing graph convolution layers. In particular, we will consider the well established work [117]. Here, the hidden layer $l+1$ is defined as

$$\boldsymbol{H}^{(l+1)} = \sigma\left(\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right), \tag{3.1}$$

where $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_N$ is the adjacency matrix of the (undirected) input graph G after adding self-loops via the identity matrix $\boldsymbol{I}_N$. $\boldsymbol{W}^{(l)}$ is the trainable weight matrix of layer l, $\widetilde{\boldsymbol{D}}_{ii} = \sum_j \widetilde{\boldsymbol{A}}_{ij}$, and $\sigma(\cdot)$ is an activation function (usually ReLU). In the first layer we have $\boldsymbol{H}^{(0)} = \boldsymbol{X}$, i.e., using the nodes' features as input. Since the latent representations $\boldsymbol{H}$ are (recursively) relying on the neighboring ones (multiplication with $\widetilde{\boldsymbol{A}}$), all instances are coupled together. Following the authors of [117], we consider GCNs with a single hidden layer:

$$\boldsymbol{Z} = f_\theta(\boldsymbol{A}, \boldsymbol{X}) = \text{softmax}\left(\hat{\boldsymbol{A}}\,\sigma\left(\hat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}^{(1)}\right)\,\boldsymbol{W}^{(2)}\right), \tag{3.2}$$

where $\hat{\boldsymbol{A}} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$. The output z_{vc} denotes the probability of assigning node v to class c. Here, we use θ to denote the set of all parameters, i.e., $\theta = \{\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}\}$. The optimal parameters θ are then learned in a semi-supervised fashion by minimizing

[2] Please note the difference to (structured) learning settings where we have *multiple but independent* graphs as training input with the goal to perform a prediction for each *graph*. Here, the prediction is done per *node* (e.g. a person in a social network) — and especially we have *dependencies* between the nodes/data instances via the edges.

cross-entropy on the output of the labeled samples $\mathcal{V}_L$, i.e., minimizing

$$\mathcal{L}(\theta; \boldsymbol{A}, \boldsymbol{X}) = -\sum_{v \in \mathcal{V}_L} \ln z_{v,c_v} \quad , \quad \boldsymbol{Z} = f_\theta(\boldsymbol{A}, \boldsymbol{X}) \tag{3.3}$$

where c_v is the given label of v from the training set. After training, $\boldsymbol{Z}$ denotes the class probabilities for every instance in the graph.

3.3 Related work

In line with the focus of this chapter, we briefly describe deep learning methods for graphs aiming to solve the node classification task.

3.3.1 Deep Learning for Graphs

Mainly two streams of research can be distinguished: (i) *node embeddings* [36, 182, 93, 22] — that often operate in an unsupervised setting — and (ii) *architectures employing layers specifically designed for graphs* [117, 184, 161]. In this chapter, we focus on the second type of principles and additionally show that our adversarial attack transfers to node embeddings as well. Regarding the developed layers, most works seek to adapt conventional CNNs to the graph domain: called graph convolutional layers or neural message passing [117, 59, 161, 184, 84]. Simply speaking, they reduce to some form of aggregation over neighbors as seen in Eq. (3.2). A more general setting is described in [84] and an overview of methods given in [161, 36].

3.3.2 Adversarial Attacks

Attacking machine learning models has a long history, with seminal works on, e.g., SVMs or logistic regression [155]. In contrast to outliers, e.g. present in attributed graphs [21], adversarial examples are created deliberately to mislead machine learning models and often are designed to be unnoticeable. Recently, deep neural networks have shown to be highly sensitive to these small adversarial perturbations to the data [212, 88]. Even more, the adversarial examples generated for one model are often also harmful when using another model: known as transferability [220]. Many tasks and models have been shown to be sensitive to adversarial attacks; however, all assume the data instances to be independent. Even [264], which considers relations between different tasks for multi-task relationship learning, still deals with the classical scenario of i.i.d. instances within each task. For interrelated data such as graphs, where the data instances (i.e., nodes) are not treated independently, little work has been done yet.

Taxonomies characterizing the attack have been introduced in [18, 178]. The two dominant types of attacks are poisoning/causative attacks which target the training data (specifically, the model's training phase is performed *after* the attack) and evasion/exploratory attacks which target the test data/application phase (here, the learned model is assumed fixed). Deriving effective poisoning attacks is usually computationally harder since also the subsequent learning of the model has to be considered. *This*

categorization is not optimally suited for our setting. In particular, attacks on the test data are causative as well since the test data is used while training the model (transductive, semi-supervised learning). Further, even when the model is fixed (evasion attack), manipulating one instance might affect all others due to the relational effects imposed by the graph structure. Our attacks are powerful even in the more challenging scenario where the model is retrained.

3.3.3 Generating Adversarial Perturbations

While most works have focused on generating adversarial perturbations for evasion attacks, poisoning attacks are far less studied [132, 155, 264] since they require to solve a challenging bi-level optimization problem that considers learning the model. In general, since finding adversarial perturbations often reduces to some non-convex (bi-level) optimization problem, different approximate principles have been introduced. Indeed, almost all works exploit the gradient or other moments of a given differentiable (surrogate) loss function to guide the search in the neighborhood of legitimate perturbations [88, 92, 178, 132, 155]. For discrete data, where gradients are undefined, such an approach is suboptimal.

Hand in hand with the attacks, improving the robustness of machine learning models has been studied — known as adversarial machine learning or robust machine learning. Since this is out of the scope of this chapter, we do not discuss these approaches here. Part III of this thesis deals with methods to improve GNN robustness.

3.3.4 Adversarial Attacks when Learning with Graphs

Works on adversarial attacks for graph learning tasks are sparse in general. For graph clustering, the work [46] has measured the changes in the result when injecting noise to a bi-partite graph that represent DNS queries. Though, they do not focus on generating attacks in a principled way. Our work [27] considered noise in the graph structure to improve the robustness when performing spectral clustering. Similarly, to improve robustness of collective classification via associative Markov networks, the work [219] considers adversarial noise in the features. They only use label smoothness and assume that the attacker can manipulate the features of every instance. After our initial study [273], there have been some additional works on adversarial attacks on graphs. [56] attack graph and node classification using reinforcement learning and evolutionary algorithms. However, in contrast to this chapter, they do not consider *poisoning* attacks or attributed graphs, do not evaluate their adversarial attacks on different models (i.e., transfer setting), and their attack on node classification only deletes edges (and does not insert any). [276] use meta learning to perform structure (poisoning) attacks on graph neural networks. They focus on *global* attacks, i.e., reducing the classification accuracy of a model across the whole graph, while in this chapter the focus is on *targeted* attacks on single nodes. [23] propose poisoning attacks on a different task: unsupervised node representation learning (or node embeddings). They exploit perturbation theory

to maximize the loss obtained after training DeepWalk. In this chapter, we focus on semi-supervised learning.

3.4 Attack Model

Given the node classification setting as described in Sec. 3.2, our goal is to perform small perturbations on the graph $G^{(0)} = (\boldsymbol{A}^{(0)}, \boldsymbol{X}^{(0)})$, leading to the graph $G' = (\boldsymbol{A}', \boldsymbol{X}')$, such that the classification performance drops. Changes to $\boldsymbol{A}^{(0)}$, are called **structure attacks**, while changes to $\boldsymbol{X}^{(0)}$ are called **feature attacks**.

3.4.1 Target vs Attackers

Specifically, our goal is to attack a specific target node $v_0 \in \mathcal{V}$, i.e. we aim to change v_0's prediction. Due to the non-i.i.d. nature of the data, v_0's outcome not only depends on the node itself, but also on the other nodes in the graph. Thus, we are not limited to perturbing v_0 but we can achieve our aim by changing other nodes as well. Indeed, this reflects real world scenarios much better since it is likely that an attacker has access to a few nodes only, and not to the entire data or v_0 itself. Therefore, besides the target node, we introduce the attacker nodes $\mathcal{A} \subseteq \mathcal{V}$. The perturbations on $G^{(0)}$ are constrained to these nodes, i.e., it must hold

$$x'_{ui} \neq x^{(0)}_{ui} \Rightarrow u \in \mathcal{A} \quad , \quad a'_{uv} \neq a^{(0)}_{uv} \Rightarrow u \in \mathcal{A} \vee v \in \mathcal{A} \tag{3.4}$$

If the target $v_0 \notin \mathcal{A}$, we call the attack an **influencer attack**, since v_0 gets not manipulated directly, but only indirectly via some influencers. If $\{v_0\} = \mathcal{A}$, we call it a **direct attack**.

To ensure that the attacker can not modify the graph completely, we further limit the number of allowed changes by a budget Δ:

$$\sum_u \sum_i |x^{(0)}_{ui} - x'_{ui}| + \sum_{u<v} |a^{(0)}_{uv} - a'_{uv}| \leq \Delta \tag{3.5}$$

More advanced ideas will be discussed in Sec. 3.4.2. For now, we denote with $\mathcal{P}^{G0}_{\Delta,\mathcal{A}}$ the set of all graphs G' that fulfill Eq. (3.4) and (3.5). Given this basic set-up, our problem is defined as:

Problem 1. *Given a graph $G^{(0)} = (\boldsymbol{A}^{(0)}, \boldsymbol{X}^{(0)})$, a target node v_0, and attacker nodes $\mathcal{A}$. Let c_{old} denote the class for v_0 based on the graph $G^{(0)}$ (predicted or using some ground truth). Determine*

$$\underset{(\boldsymbol{A}', \boldsymbol{X}') \in \mathcal{P}^{G0}_{\Delta,\mathcal{A}}}{\arg\max} \max_{c \neq c_{old}} \ln z^*_{v_0,c} - \ln z^*_{v_0,c_{old}}$$

$$\textit{subject to } \boldsymbol{Z}^* = f_{\theta^*}(\boldsymbol{A}', \boldsymbol{X}') \textit{ with } \theta^* = \arg\min_\theta L(\theta; \boldsymbol{A}', \boldsymbol{X}')$$

That is, we aim to find a perturbed graph G' that classifies v_0 as c_{new} and has maximal 'distance' (in terms of log-probabilities/logits) to c_{old}. Note that for the perturbed graph

G', the optimal parameters θ^* are used, matching the transductive learning setting where the model is learned on the given data. Therefore, we have a bi-level optimization problem. As a simpler variant, one can also consider an evasion attack assuming the parameters are static and learned based on the old graph, $\theta^* = \arg\min_\theta L(\theta; \boldsymbol{A}^{(0)}, \boldsymbol{X}^{(0)})$.

3.4.2 Unnoticeable Perturbations

Typically, in an adversarial attack scenario, the attackers try to modify the input data such that the changes are *unnoticeable*. Unlike to image data, where this can easily be verified visually and by using simple constraints, in the graph setting this is much harder mainly for two reasons: (i) the graph structure is discrete preventing to use infinitesimal small changes, and (ii) sufficiently large graphs are not suitable for visual inspection.

How can we ensure unnoticeable perturbations in our setting? In particular, we argue that only considering the budget Δ might not be enough. Especially if a large Δ is required due to complicated data, we still want realistically looking perturbed graphs G'. Therefore, our core idea is to allow only those perturbations that preserve specific inherent properties of the input graph.

3.4.2.1 Graph Structure Preserving Perturbations

Undoubtedly, the most prominent characteristic of the graph structure is its degree distribution, which often resembles a power-law like shape in real networks. If two networks show very different degree distributions, it is easy to tell them apart. Therefore, we aim to only generate perturbations which follow similar power-law behavior as the input.

For this purpose we refer to a statistical two-sample test for power-law distributions [15]. That is, we estimate whether the two degree distributions of $G^{(0)}$ and G' stem from the same distribution or from individual ones, using a likelihood ratio test.

More precisely, the procedure is as follows: We first estimate the scaling parameter α of the power-law distribution $p(x) \propto x^{-\alpha}$ referring to the degree distribution of $G^{(0)}$ (equivalently for G'). While there is no exact and closed-form solution to estimate α in the case of discrete data, [52] derived an approximate expression, which for our purpose of a graph G translates to

$$\alpha_G \approx 1 + |\mathcal{D}_G| \cdot \left[\sum_{d_i \in \mathcal{D}_G} \log \frac{d_i}{d_{\min} - \frac{1}{2}} \right]^{-1} \tag{3.6}$$

where $d_{\min}$ denotes the minimum degree a node needs to have to be considered in the power-law test and $\mathcal{D}_G = \{d_v^G \mid v \in \mathcal{V}, d_v^G \geq d_{\min}\}$ is the multiset containing the list of node degrees, where d_v^G is the degree of node v in G. Using this, we get estimates for the values $\alpha_{G^{(0)}}$ and $\alpha_{G'}$. Similarly, we can estimate α_{comb} using the combined samples $\mathcal{D}_{comb} = \mathcal{D}_{G^{(0)}} \cup \mathcal{D}_{G'}$.

Given the scaling parameter α_x, the log-likelihood for the samples $\mathcal{D}_x$ can easily be evaluated as[3]

$$l(\mathcal{D}_x) = |\mathcal{D}_x| \cdot \log \alpha_x + |\mathcal{D}_x| \cdot \alpha_x \cdot \log d_{\min} - (\alpha_x + 1) \sum_{d_i \in \mathcal{D}_x} \log d_i \tag{3.7}$$

Using these log-likelihood scores, we set up the significance test, estimating whether the two samples $\mathcal{D}_{G^{(0)}}$ and $\mathcal{D}_{G'}$ come from the same power law distribution (null hypotheses H_0) as opposed to separate ones (H_1). That is, we formulate two competing hypotheses

$$l(H_0) = l(\mathcal{D}_{comb}) \quad \text{and} \quad l(H_1) = l(\mathcal{D}_{G^{(0)}}) + l(\mathcal{D}_{G'}) \tag{3.8}$$

Following the likelihood ratio test, the final test statistic is

$$\Lambda(G^{(0)}, G') = -2 \cdot l(H_0) + 2 \cdot l(H_1). \tag{3.9}$$

which for large sample sizes follows a χ^2 distribution with one degree of freedom [15].

A typical p-value for rejecting the null hypothesis H_0 (i.e. concluding that both samples come from different distributions) is 0.05, i.e., statistically, in one out of twenty cases we reject the null hypothesis although it holds (*type I error*). In our adversarial attack scenario, however, we argue that a human trying to find out whether the data has been manipulated would be far more conservative and ask the other way: Given that the data was manipulated, what is the probability of the test falsely not rejecting the null hypothesis (*type II error*).

While we cannot compute the type II error in our case easily, type I and II error probabilities have an inverse relation in general. Thus, by selecting a very conservative p-value corresponding to a high type I error, we can reduce the probability of a type II error. We therefore set the critical p-value to 0.95, i.e., if we were to sample two degree sequences from the same power law distribution, we were to reject the null hypothesis in 95% of the times and could then investigate whether the data has been compromised based on this initial suspicion. On the other hand, if our modified graph's degree sequence passes this very conservative test, we conclude that the changes to the degree distribution are *unnoticeable*.

Using the above p-value in the χ^2 distribution, we only accept perturbations $G' = (\boldsymbol{A}', \boldsymbol{X}')$ where the degree distribution fulfills

$$\Lambda(G^{(0)}, G') < \tau \approx 0.004 \tag{3.10}$$

3.4.2.2 Feature Statistics Preserving Perturbations

While the above principle could be applied to the nodes' features as well (e.g., preserving the distribution of feature occurrences), we argue that such a procedure is too limited. In particular, such a test would not well reflect the correlation/co-occurrence of different

[3] due to a typing error, the '−' before the last term was a '+' in the originally published version [273].

features: If two features have never occurred together in $G^{(0)}$, but they do once in G', the distribution of feature occurrences would still be very similar. Such a change, however, is easily noticeable. Think, e.g., about two words which have never been used together but are suddenly used in G'. Thus, we refer to a test based on feature co-occurrence.

Since designing a statistical test based on the co-occurrences requires to model the joint distribution over features — intractable for correlated multivariate binary data [160] — we refer to a deterministic test. In this regard, setting features to 0 is uncritical since it does not introduce new co-occurrences. The question is: Which features of a node u can be set to 1 to be regarded unnoticeable?

To answer this question, we consider a probabilistic random walker on the co-occurrence graph $C = (\mathcal{F}, E)$ of features from $G^{(0)}$, i.e., $\mathcal{F}$ is the set of features and $E \subseteq \mathcal{F} \times \mathcal{F}$ denotes which features have occurred together so far. We argue that adding a feature i is unnoticeable if the probability of reaching it by a random walker starting at the features originally present for node u and performing one step is significantly large. Formally, let $S_u = \{j \mid X_{uj} \neq 0\}$ be the set of all features originally present for node u. We consider addition of feature $i \notin S_u$ to node u as unnoticeable if

$$p(i \mid S_u) = \frac{1}{|S_u|} \sum_{j \in S_u} \frac{1}{d_j} \cdot E_{ij} > \sigma. \tag{3.11}$$

where d_j denotes the degree in the co-occurrence graph C. That is, given that the probabilistic walker has started at any feature $j \in S_u$, after performing one step it would reach the feature i at least with probability σ. In our experiments we simply picked σ to be half of the maximal achievable probably, i.e., $\sigma = 0.5 \cdot \frac{1}{|S_u|} \sum_{j \in S_u} 1/d_j$.

The above principle has two desirable effects: First, features i which have co-occurred with many of u's features (i.e., in other nodes) have a high probability; they are less noticeable when being added. Second, features i that only co-occur with features $j \in S_u$ that are not specific to the node u (e.g., features j which co-occur with almost every other feature; stopwords) have low probability; adding i would be noticeable. Thus, we obtain the desired result.

Using the above test, we only accept perturbations $G' = (\boldsymbol{A}', X')$ where the feature values fulfill

$$\forall u \in \mathcal{V} : \forall i \in \mathcal{F} : x'_{ui} = 1 \Rightarrow i \in S_u \vee p(i|S_u) > \sigma \tag{3.12}$$

In summary, to ensure unnoticeable perturbations, we update our problem definition to:

Problem 2. *Same as Problem 1 but replacing $\mathcal{P}^{G0}_{\Delta,\mathcal{A}}$ with the more restricted set $\hat{\mathcal{P}}^{G0}_{\Delta,\mathcal{A}}$ of graphs that additionally preserve the degree distribution (Eq. 3.10) and feature co-occurrence (Eq. 3.12).*

3.5 Generating Adversarial Graphs

Solving Problem 1/2 is highly challenging. While (continuous) bi-level problems for attacks have been addressed in the past by gradient computation based on first-order

KKT conditions [155, 132], such a solution is not possible in our case due to the data's discreteness and the large number of parameters θ. Therefore, we propose a sequential approach, where we first attack a *surrogate model*, thus, leading to an attacked graph. This graph is subsequently used to train the final model. Indeed, this approach can directly be considered as a check for transferability since we do not specifically focus on the used model but only on a surrogate one.

3.5.1 Surrogate Model

To obtain a tractable surrogate model that still captures the idea of graph convolutions, we perform a linearization of the model from Eq. 3.2. That is, we replace the non-linearity $\sigma(.)$ with a simple linear activation function, leading to:

$$Z' = \text{softmax}\left(\hat{\boldsymbol{A}}\,\hat{\boldsymbol{A}}\,\boldsymbol{X}\boldsymbol{W}^{(1)}\,\boldsymbol{W}^{(2)}\right) = \text{softmax}\left(\hat{\boldsymbol{A}}^2\,\boldsymbol{X}\boldsymbol{W}\right) \tag{3.13}$$

Since $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ are (free) parameters to be learned, they can be absorbed into a single matrix $\boldsymbol{W} \in \mathbb{R}^{D\times K}$.

Since our goal is to maximize the difference in the log-probabilities of the target v_0 (given a certain budget Δ), the instance-dependent normalization induced by the softmax can be ignored. Thus, the log-probabilities can simply be reduced to $\hat{\boldsymbol{A}}^2\,\boldsymbol{X}\boldsymbol{W}$. Accordingly, given the trained surrogate model on the (uncorrupted) input data with learned parameters $\boldsymbol{W}$, we define the surrogate loss

$$\mathcal{L}_s\left(\boldsymbol{A}, \boldsymbol{X}; \boldsymbol{W}, v_0\right) = \max_{c\neq c_{\text{old}}} [\hat{\boldsymbol{A}}^2\,\boldsymbol{X}\boldsymbol{W}]_{v_0,c} - [\hat{\boldsymbol{A}}^2\,\boldsymbol{X}\boldsymbol{W}]_{v_0,c_{\text{old}}} \tag{3.14}$$

and aim to solve $\underset{(\boldsymbol{A}',\boldsymbol{X}')\in\hat{\mathcal{P}}^{G0}_{\Delta,\mathcal{A}}}{\arg\max}\ \mathcal{L}_s\left(\boldsymbol{A}', \boldsymbol{X}'; \boldsymbol{W}, v_0\right)$.

While being much simpler, this problem is still intractable to solve exactly due to the discrete domain and the constraints. Thus, in the following we introduce a scalable greedy approximation scheme. For this, we define scoring functions that evaluate the surrogate loss from Eq. (3.14) obtained *after* adding/removing a feature $f = (u, i)$ or edge $e = (u, v)$ to an arbitrary graph $G = (\boldsymbol{A}, \boldsymbol{X})$:

$$s_{\text{struct}}(e; G, v_0) := \mathcal{L}_s(\boldsymbol{A}', \boldsymbol{X}; \boldsymbol{W}, v_0) \tag{3.15}$$

$$s_{\text{feat}}(f; G, v_0) := \mathcal{L}_s(\boldsymbol{A}, \boldsymbol{X}'; \boldsymbol{W}, v_0) \tag{3.16}$$

where $\boldsymbol{A}' := \boldsymbol{A} \pm e$ (i.e., $a'_{uv} = a'_{vu} = 1 - a_{uv}$)[4] and $\boldsymbol{X}' := \boldsymbol{X} \pm f$ (i.e., $x'_{ui} = 1 - x_{ui}$).

3.5.2 Approximate Solution

Algorithm 1 shows the pseudo-code. In detail, following a locally optimal strategy, we sequentially 'manipulate' the most promising element: either an entry from the adjacency

[4]Please note that by modifying a single element $e = (u, v)$ we always change two entries, a_{uv} and a_{vu}, of $\boldsymbol{A}$ since we are operating on an undirected graph.

Algorithm 1: NETTACK: Adversarial attacks on graphs

Input: Graph $G^{(0)} = (\boldsymbol{A}^{(0)}, \boldsymbol{X}^{(0)})$, target node v_0, attacker nodes $\mathcal{A}$, modification budget Δ

Output: Modified Graph $G' = (\boldsymbol{A}', \boldsymbol{X}')$

Train surrogate model on $G^{(0)}$ to obtain $\boldsymbol{W}$; `// Eq.` (3.13)
$t \leftarrow 0$;
while $\frac{1}{2}|\boldsymbol{A}^{(t)} - \boldsymbol{A}^{(0)}| + |\boldsymbol{X}^{(t)} - \boldsymbol{X}^{(0)}| < \Delta$ **do**
  $C_{\text{struct}} \leftarrow$ `candidate_edge_perturbations`$(\boldsymbol{A}^{(t)}, \mathcal{A})$;
  $e^* = (u^*, v^*) \leftarrow \underset{e \in C_{\text{struct}}}{\arg\max}\, s_{\text{struct}}\left(e; G^{(t)}, v_0\right)$;
  $C_{\text{feat}} \leftarrow$ `candidate_feature_perturbations`$(X^{(t)}, \mathcal{A})$;
  $f^* = (u^*, i^*) \leftarrow \underset{f \in C_{\text{feat}}}{\arg\max}\, s_{\text{feat}}\left(f; G^{(t)}, v_0\right)$;
  if $s_{struct}(e^*; G^{(t)}, v_0) > s_{feat}(f^*; G^{(t)}, v_0)$ **then** $G^{(t+1)} \leftarrow G^{(t)} \pm e^*$;
  else $G^{(t+1)} \leftarrow G^{(t)} \pm f^*$;
  $t \leftarrow t + 1$;
return : $G^{(t)}$
`// Train final graph model on the corrupted graph` $G^{(t)}$

matrix or a feature entry (taking the constraints into account). That is, given the current state of the graph $G^{(t)}$, we compute a candidate set C_{struct} of allowable elements (u, v) whose change from 0 to 1 (or vice versa; hence the $\pm$ sign in the pseudocode) does not violate the constraints imposed by $\hat{\mathcal{P}}^{G0}_{\Delta,\mathcal{A}}$. Among these elements we pick the one which obtains the highest difference in the log-probabilities, indicated by the score function $s_{\text{struct}}(e; G^{(t)}, v_0)$. Similarly, we compute the candidate set C_{feat} and the score function $s_{\text{feat}}(f; G^{(t)}, v_0)$ for every allowable feature manipulation of feature i and node u. Whichever change obtains the higher score is picked and the graph accordingly updated to $G^{(t+1)}$. This process is repeated until the budget Δ has been exceeded.

To make Algorithm 1 tractable, two core aspects have to hold: (i) an efficient computation of the score functions s_{struct} and s_{feat}, and (ii) an efficient check which edges and features are compliant with our constraints $\hat{\mathcal{P}}^{G0}_{\Delta,\mathcal{A}}$, thus, forming the sets C_{struct} and C_{feat}. In the following, we describe these two parts in detail.

3.5.3 Fast Computation of Score Functions

3.5.3.1 Structural attacks

We start by describing how to compute s_{struct}. For this, we have to compute the class prediction (in the surrogate model) of node v_0 *after* adding/removing an edge (m, n). Since we are now optimizing w.r.t. $\boldsymbol{A}$, the term $\boldsymbol{XW}$ in Eq. (3.14) is a constant — we substitute it with $\boldsymbol{C} = \boldsymbol{XW} \in \mathbb{R}^{N \times K}$. The log-probabilities of node v_0 are then given by $g_{v_0} = [\hat{\boldsymbol{A}}^2]_{v_0} \cdot \boldsymbol{C} \in \mathbb{R}^{1 \times K}$ where $[\hat{\boldsymbol{A}}^2]_{v_0}$ denotes a row vector. Thus, we only have to inspect how this row vector changes to determine the optimal edge manipulation.

Naively recomputing $[\hat{\boldsymbol{A}}^2]_{v_0}$ for every element from the candidate set, though, is not practicable. An important observation to alleviate this problem is that in the used two-layer GCN the prediction for each node is influenced by its two-hop neighborhood only. That is, the above row vector is zero for most of the elements. And even more importantly, we can derive an incremental update — we don't have to recompute the updated $[\hat{\boldsymbol{A}}^2]_{v_0}$ from scratch.

Theorem 1. *Given an adjacency matrix $\boldsymbol{A}$, and its corresponding matrices $\widetilde{\boldsymbol{A}}$, $\hat{\boldsymbol{A}}^2$, $\widetilde{\boldsymbol{D}}$. Denote with $\boldsymbol{A}'$ the adjacency matrix when adding or removing the element $e = (m, n)$ from $\boldsymbol{A}$. It holds:*

$$[\hat{\boldsymbol{A}'}^2]_{uv} = \frac{1}{\sqrt{\widetilde{d}'_u \widetilde{d}'_v}} \left(\sqrt{\widetilde{d}_u \widetilde{d}_v} [\hat{\boldsymbol{A}}^2]_{uv} - \frac{\widetilde{a}_{uv}}{\widetilde{d}_u} - \frac{a_{uv}}{\widetilde{d}_v} + \frac{a'_{uv}}{\widetilde{d}'_v} + \frac{\widetilde{a}'_{uv}}{\widetilde{d}'_u} - \right.$$
$$\left. - \frac{a_{um} a_{mv}}{\widetilde{d}_m} + \frac{a'_{um} a'_{mv}}{\widetilde{d}'_m} - \frac{a_{un} a_{nv}}{\widetilde{d}_n} + \frac{a'_{un} a'_{nv}}{\widetilde{d}'_n} \right) \tag{3.17}$$

where $\widetilde{d}'$, a', and $\widetilde{a}'$, are defined as (using the Iverson bracket $\mathbb{I}$):

$$\widetilde{d}'_k = \widetilde{d}_k + \mathbb{I}[k \in \{m, n\}] \cdot (1 - 2 \cdot a_{mn})$$
$$a'_{kl} = a_{kl} + \mathbb{I}[\{k, l\} = \{m, n\}] \cdot (1 - 2 \cdot a_{kl})$$
$$\widetilde{a}'_{kl} = \widetilde{a}_{kl} + \mathbb{I}[\{k, l\} = \{m, n\}] \cdot (1 - 2 \cdot \widetilde{a}_{kl})$$

Proof. Let S and S' be defined as $S = \sum_{k=1}^{N} \frac{a_{uk} a_{kv}}{\widetilde{d}_k}$ and $S' = \sum_{k=1}^{N} \frac{a'_{uk} a'_{kv}}{\widetilde{d}'_k}$. We have $[\hat{\boldsymbol{A}}]_{uv} = \frac{\widetilde{a}_{uv}}{\sqrt{\widetilde{d}_u \widetilde{d}_v}}$. If $u \neq v$, then

$$[\hat{\boldsymbol{A}}^2]_{uv} = \sum_{k=1}^{N} [\hat{\boldsymbol{A}}]_{uk} [\hat{\boldsymbol{A}}]_{kv} = \frac{\widetilde{a}_{uv}}{\widetilde{d}_u \sqrt{\widetilde{d}_u \widetilde{d}_v}} + \frac{\widetilde{a}_{uv}}{\widetilde{d}_v \sqrt{\widetilde{d}_u \widetilde{d}_v}} + \frac{1}{\sqrt{\widetilde{d}_u \widetilde{d}_v}} S.$$

Having the above equation for $\hat{\boldsymbol{A}'}^2$, we get

$$[\hat{\boldsymbol{A}'}^2]_{uv} \big(\sqrt{\widetilde{d}'_u \widetilde{d}'_v} \big) - [\hat{\boldsymbol{A}}^2]_{uv} \big(\sqrt{\widetilde{d}_u \widetilde{d}_v} \big) =$$
$$\big[\frac{\widetilde{a}'_{uv}}{\widetilde{d}'_u} - \frac{\widetilde{a}_{uv}}{\widetilde{d}_u} \big] + \big[\frac{\widetilde{a}'_{uv}}{\widetilde{d}'_v} - \frac{\widetilde{a}_{uv}}{\widetilde{d}_v} \big] + (S' - S).$$

After replacing $S' - S = -\frac{a_{um} a_{mv}}{\widetilde{d}_m} + \frac{a'_{um} a'_{mv}}{\widetilde{d}'_m} - \frac{a_{un} a_{nv}}{\widetilde{d}_n} + \frac{a'_{un} a'_{nv}}{\widetilde{d}'_n}$ in the above equation, it is straightforward to derive Eq. 3.17. Deriving this equation for the case $u = v$ is similar. Eq. 3.17 encompasses both cases. □

Eq. (3.17) enables us to update the entries in $\hat{\boldsymbol{A}}^2$ in *constant time*; and in a sparse and incremental manner. Remember that all $\widetilde{a}_{uv}$, a_{uv}, and a'_{uv} are either 1 or 0, and their corresponding matrices are sparse. Given this highly efficient update of $[\hat{\boldsymbol{A}}^2]_{v_0}$ to $[\hat{\boldsymbol{A}'}^2]_{v_0}$, the updated log-probabilities and, thus, the final score according to Eq. (3.15) can be easily computed.

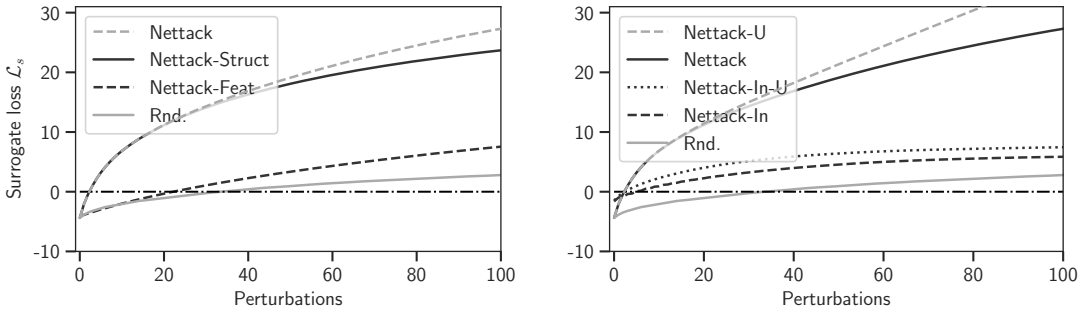


Figure 3.2: Average surrogate loss for increasing number of perturbations. Different variants of our method on CORA-ML. Larger is better.

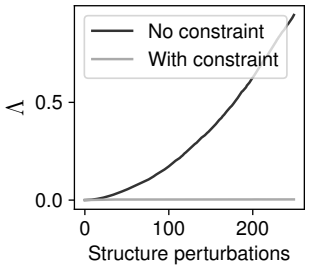


Figure 3.3: Change in test statistic Λ (degree distr.)

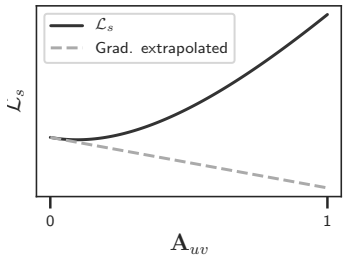


Figure 3.4: Example for loss when extrapolating gradient vs. actual loss

3.5.3.2 Feature Attacks

The feature attacks are much easier to realize. Indeed, by fixing the class $c \neq c_{\text{old}}$ with currently largest log-probability score $[\hat{\boldsymbol{A}}^2 \boldsymbol{X}\boldsymbol{W}]_{v_0 c}$, the problem is linear in $\boldsymbol{X}$ and every entry of $\boldsymbol{X}$ acts independently. Thus, to find the best node and feature (u^*, i^*) we only need to compute the gradients

$$\begin{aligned} \Upsilon_{ui} &= \frac{\partial}{\partial x_{ui}} \big([\hat{\boldsymbol{A}}^2 \boldsymbol{X}\boldsymbol{W}]_{v_0 c} - [\hat{\boldsymbol{A}}^2 \boldsymbol{X}\boldsymbol{W}]_{v_0 c_{\text{old}}} \big) \\ &= [\hat{\boldsymbol{A}}^2]_{v_0 u} \left([\boldsymbol{W}]_{ic} - [\boldsymbol{W}]_{ic_{\text{old}}} \right) \end{aligned} \tag{3.18}$$

and subsequently pick the one with the highest absolute value that points into an allowable direction (e.g., if the feature was 0, the gradient needs to point into the positives). The value of the score function s_{feat} for this best element is then simply obtained by adding $|\Upsilon_{ui}|$ to the current value of the loss function:

$$\mathcal{L}_s(\boldsymbol{A}, \boldsymbol{X}; \boldsymbol{W}, v_0) + |\Upsilon_{ui}| \cdot \mathbb{I}[(2 \cdot x_{ui} - 1) \cdot \Upsilon_{ui} < 0]$$

All this can be done in *constant time* per feature. The elements where the gradient points outside the allowable direction should not be perturbed since they would only hinder the attack — thus, the old score stays unchanged.

3.5.4 Fast Computation of Candidate Sets

Last, we have to make sure that all perturbations are valid according to the constraints $\hat{\mathcal{P}}^{G0}_{\Delta,\mathcal{A}}$. For this, we defined the sets C_{struct} and C_{feat}. Clearly, the constraints introduced in Eq. 3.4 and 3.5 are easy to ensure. The budget constraint Δ is fulfilled by the process of the greedy approach, while the elements which can be perturbed according to Eq. 3.4 can be precomputed. Likewise, the node-feature combinations fulfilling the co-occurrence test of Eq. 3.12 can be precomputed. Thus, the set C_{feat} only needs to be instantiated once.

The significance test for the degree distribution, however, does not allow such a precomputation since the underlying degree distribution dynamically changes. How can we efficiently check whether a potential perturbation of the edge (m, n) still preserves a similar degree distribution? Indeed, since the individual degrees only interact additively, we can again derive a *constant time* incremental update of our test statistic Λ.

Theorem 2. *Given graph $G = (\boldsymbol{A}, \boldsymbol{X})$ and the multiset $\mathcal{D}_G$ (see below Eq. 3.6). Denote with $R^G = \sum_{d_i \in \mathcal{D}_G} \log d_i$ the sum of log degrees. Let $e = (m, n)$ be a candidate edge perturbation, and d_m and d_n the degrees of the nodes in G. For $G' = G \pm e$ we have:*

$$\alpha_{G'} = 1 + n^e \left[R^{G'} - n^e \log\left(d_{\min} - \tfrac{1}{2}\right) \right]^{-1} \tag{3.19}$$

$$l\left(\mathcal{D}_{G'}\right) = n^e \log \alpha_{G'} + n^e \alpha_{G'} \log d_{\min} - \left(\alpha_{G'} + 1\right) R^{G'} \tag{3.20}$$

where

$$\begin{aligned} x &= 1 - 2 \cdot a_{mn} \\ n^e &= |\mathcal{D}_G| + (\mathbb{I}[d_m + 1 - a_{mn} = d_{\min}] + \mathbb{I}[d_n + 1 - a_{mn} = d_{\min}]) \cdot x \\ R^{G'} &= R^G - \mathbb{I}[d_m \geq d_{\min}] \log d_m + \mathbb{I}[d_m + x \geq d_{\min}] \log(d_m + x) \\ &\quad - \mathbb{I}[d_n \geq d_{\min}] \log d_n + \mathbb{I}[d_n + x \geq d_{\min}] \log(d_n + x). \end{aligned} \tag{3.21}$$

Proof. Firstly, we show that if we incrementally compute n^e according to the update equation of Theorem 2, n^e will be equal to $|\mathcal{D}_{G'}|$. The term $\mathbb{I}[d_m + 1 - a_{mn} = d_{\min}] \cdot x$ will be activated (i.e., non-zero) only in two cases: 1) $a_{mn} = 1$ (i.e., $G' = G - e$), and $d_m = d_{\min}$, then $x < 0$ and the update equation actually removes node m from $\mathcal{D}_G$. 2) $a_{mn} = 0$ (i.e., $G' = G + e$), and $d_m = d_{\min} - 1$, then $x > 0$ and the update equation actually adds node m to $\mathcal{D}_G$. A similar argumentation is applicable for node n. Accordingly, we have that $n^e = |\mathcal{D}_{G'}|$.

Similarly, one can show the valid incremental update for $R^{G'}$ considering that only nodes with degree larger than $d_{\min}$ are considered and that $d_m + x$ is the new degree. Having incremental updates for n^e and $R^{G'}$, the updates for $\alpha_{G'}$ and $l(\mathcal{D}_{G'})$ follow easily from their definitions. □

Given $G^{(t)}$, we can now incrementally compute $l(\mathcal{D}_{G_e^{(t)}})$, where $G_e^{(t)} = G^{(t)} \pm e$. Equivalently we get incremental updates for $l(\mathcal{D}_{comb})$ after an edge perturbation. Since all r.h.s. of the equations above can be computed in constant time, also the test statistic

Class: neural networks				Class: theory				Class: probabilistic models			
constrained		**unconstrained**		**constrained**		**unconstrained**		**constrained**		**unconstrained**	
probabilistic	25	efforts	2	driven	3	designer	0	difference	2	calls	1
probability	38	david	0	increase	8	assist	0	solve	3	chemical	0
bayesian	28	averages	2	heuristic	4	disjunctive	7	previously	12	unseen	1
inference	27	accomplished	3	approach	56	interface	1	control	16	corporation	3
probabilities	20	generality	1	describes	20	driven	3	reported	1	fourier	1
observations	9	expectation	10	performing	7	refinement	0	represents	8	expressed	2
estimation	35	specifications	0	allow	11	refines	0	steps	5	robots	0
distributions	21	family	10	functional	2	starts	1	allowing	7	achieving	0
independence	5	uncertain	3	11	3	restrict	0	task	17	difference	2
variant	9	observations	9	acquisition	1	management	0	expressed	2	requirement	1

Table 3.1: Top-10 feature perturbations per class on CORA-ML

$\Lambda(G^{(0)}, G_e^{(t)})$ can be computed in *constant time*. Overall, the set of valid candidate edge perturbations at iteration t is $C_{\text{struct}} = \{e{=}(m,n) \mid \Lambda(G^{(0)}, G_e^{(t)}) < \tau \wedge (m \in \mathcal{A} \vee n \in \mathcal{A})\}$. Since $R^{G^{(t)}}$ can be incrementally updated to $R^{G^{(t+1)}}$ once the best edge perturbation has been performed, the full approach is highly efficient.

3.5.5 Complexity

The candidate set generation (i.e., which edges/features are allowed to change) and the score functions can be incrementally computed and exploit the graph's sparsity, thus, ensuring scalability. The runtime complexity of the algorithm can easily be determined as:

$$\mathcal{O}(\Delta \cdot |\mathcal{A}| \cdot (N \cdot th_{v_0} + D))$$

where th_{v_0} indicates the size of the two-hop neighborhood of the node v_0 during the run of the algorithm.

In every of the Δ many iterations, each attacker evaluates the potential edge perturbations (N at most) and feature perturbations (D at most). For the former, this requires to update the two-hop neighborhood of the target due to the two convolution layers. Assuming the graph is sparse, th_{v_0} is much smaller than N. The feature perturbations are done in constant time per feature. Since all constraints can be checked in constant time they do not affect the complexity.

3.6 Experiments

We explore how our attacks affect the surrogate model, and evaluate transferability to other models and for multiple datasets.

3.6.1 Setup

We use the well-known CORA-ML and CITESEER networks as in [22], and POLBLOGS [1]. The dataset characteristics are shown in Table A.1 (c.f. appendix). We split the network

in labeled (20%) and unlabeled nodes (80%). We further split the labeled nodes in equal parts *training* and *validation* sets to train our surrogate model. That is, we remove the labels from the validation set in the training procedure and use them as the stopping criterion (i.e., stop when validation error increases). The labels of the unlabeled nodes are never visible to the surrogate model during training.

We average over five different random initializations/ splits, where for each we perform the following steps. We first train our surrogate model on the labeled data and among all nodes from the test set that have been *correctly* classified, we select (i) the 10 nodes with highest margin of classification, i.e., they are clearly correctly classified, (ii) the 10 nodes with lowest margin (but still correctly classified) and (iii) 20 more nodes randomly. These will serve as the target nodes for our attacks.

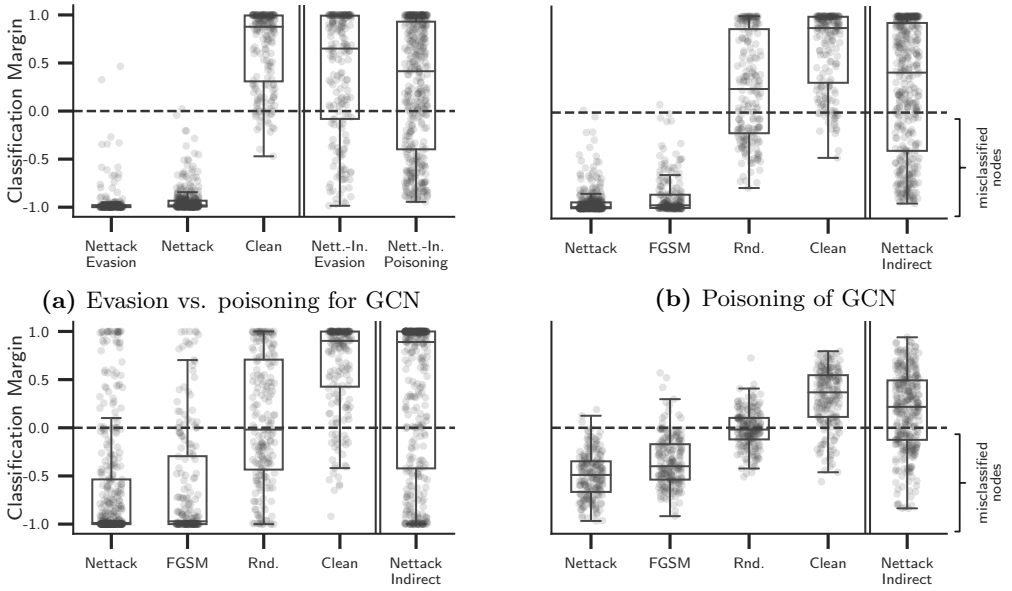
Then, we corrupt the input graph using the model proposed in this chapter, called Nettack for direct attacks, and Nettack-In for influence attacks, respectively. For the latter we pick as attackers the five neighboring nodes of the target that, if disconnected from the target, would lead to the largest decrease of the surrogate loss. If the target has less than five neighbors, we connect it to the nodes that lead to the strongest increase in surrogate loss until it has five neighbors.
Since no other competitors exist, we compare against two baselines: (i) Fast Gradient Sign Method (FGSM) [88] as a direct attack on v_0 (in our case also making sure that the result is still binary). (ii) Rnd is an attack in which we modify the structure of the graph. Given our target node v_0, in each step we randomly sample nodes u for which $c_{v_0} \neq c_u$ and add the edge u, v to the graph structure, assuming unequal class labels are hindering classification. Note that this means that (unlike Nettack) Rnd has access to *all* node labels, hence is a strong baseline.

3.6.2 Attacks on the Surrogate Model

We start by analyzing different variants of our method by inspecting their influence on the surrogate model. In Fig. 3.2 (left) we plot the surrogate loss when performing a specific number of perturbations. Note that once the surrogate loss is positive, we realized a successful misclassification. We analyze Nettack, and variants where we only manipulate features or only the graph structure. As seen, perturbations in the structure lead to a stronger change in the surrogate loss compared to feature attacks. Still, combining both is the most powerful, only requiring around 3 changes to obtain a misclassification. For comparison we have also added Rnd, which is clearly not able to achieve good performance.

In Fig. 3.2 (right) we analyze our method when using a direct vs. influencer attack. Clearly, direct attacks need fewer perturbations — still, influencer attacks are also possible, posing a high risk in real life scenarios. The figure also shows the result when *not* using our constraints as proposed in Sec. 3.4.2, indicated by the name Nettack-U. As seen, even when using our constraints, the attack is still successful. Thus, unnoticeable perturbations can be generated.

It is worth mentioning that the constraints are indeed necessary. Figure 3.3 shows the test statistic Λ of the resulting graph with or without our constraints. As seen the con-



(a) Evasion vs. poisoning for GCN

(b) Poisoning of GCN

(c) Poisoning of Column Network

(d) Poisoning of DeepWalk

Figure 3.5: Results on CORA-ML using different attack algorithms (perturbation budget $\Delta = d + 2$). Clean indicates the original data. Lower scores are better.

straint we impose has an effect on our attack; if not enforced, the power law distribution of the corrupted graph becomes more and more dissimilar to the original graph's. Similarly, Table 3.1 illustrates the result for the feature perturbations. For CORA-ML, the features correspond to the presence of *words* in the abstracts of papers. For each class (i.e., set of nodes with same label), we plot the top-10 features that have been manipulated by the techniques (these account for roughly 50% of all perturbations). Further, we report for each feature its original occurrence within the class. We see that the used features are indeed different — even more, the unconstrained version often uses words which are 'unlikely' for the class (indicated by the small numbers). Using such words can easily be noticed as manipulations, e.g., 'david' in neural networks or 'chemical' in probabilistic models. Our constraint ensures that the changes are more subtle.

Overall, we conclude that attacking the features and structure simultaneously is very powerful; and the introduced constraints do not hinder the attack while generating more realistic perturbations. Direct attacks are clearly easier than influencer attacks.

3.6.3 Transferability of Attacks

After exploring how our attack affects the (fixed) surrogate model, we will now find out whether our attacks are also successful on established deep learning models for graphs. For this, we pursue the approach from before and use a budget of $\Delta = d_{v_0} + 2$, where d_{v_0} is the degree of the target node we currently attack. This is motivated by the observation that high-degree nodes are more difficult to attack than low-degree ones. In the following we always report the score $X = z^*_{v_0,c_{\text{old}}} - \max_{c \neq c_{\text{old}}} z^*_{v_0,c}$ using the ground truth label

	Cora-ML			Citeseer			Polblogs		
	GCN	CLN	DeepWalk	GCN	CLN	DeepWalk	GCN	CLN	DeepWalk
Clean	0.90	0.82	0.84	0.88	0.71	0.76	0.94	0.82	0.93
Nettack	**0.01**	**0.16**	**0.02**	**0.02**	**0.20**	**0.01**	**0.06**	**0.46**	**0.06**
FGSM	0.03	0.18	0.10	0.07	0.23	0.05	0.41	0.54	0.37
Rnd	0.61	0.52	0.46	0.60	0.52	0.38	0.36	0.54	0.30
Nettack-In	0.64	0.67	0.65	0.62	0.56	0.48	0.86	0.62	0.91

Table 3.2: Classification accuracy of classification models when trained on datasets poisoned by different methods (perturbation budget $\Delta = d + 2$). Smaller is better.

c_{old} of the target. We call X the classification margin. *The smaller X, the better.* For values smaller than 0, the targets get misclassified. Note that this could even happen for the clean graph since the classification itself might not be perfect.

3.6.3.1 Evasion vs Poisoning Attack

In Figure 3.5a we evaluate Nettack's performance for two attack types: evasion attacks, where the model parameters (here of GCN [117]) are kept fix based on the clean graph; and poisoning attacks, where the model is retrained after the attack (averaged over 10 runs). In the plot, every dot represents one target node. As seen, direct attacks are extremely successful — even for the challenging poisoning case almost every target gets misclassified. We therefore conclude that our surrogate model and loss are a sufficient approximation of the true loss on the non-linear model *after* re-training on the perturbed data. Clearly, influencer attacks (shown right of the double-line) are harder but they still work in both cases. Since poisoning attacks are in general harder and match better the transductive learning scenario, we report in the following only these results.

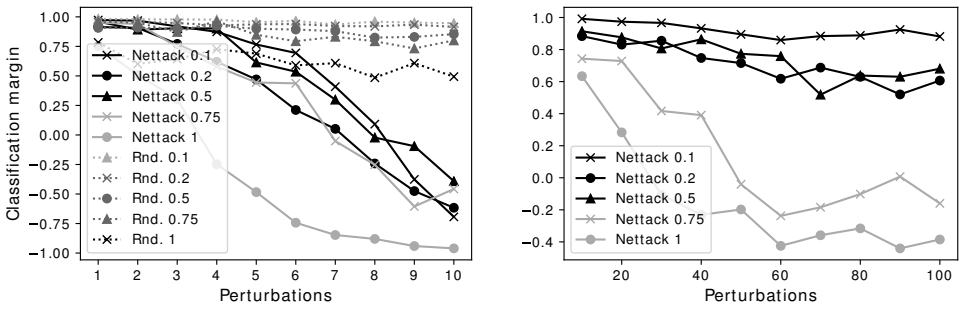
3.6.3.2 Comparison

Figure 3.5b and 3.5c show that the corruptions generated by Nettack transfer to different (semi-supervised) graph convolutional methods: GCN [117] and CLN [184]. Most remarkably, even the unsupervised model DeepWalk (DW) [182] is strongly affected by our perturbations (Figure 3.5d). Since DW only handles unattributed graphs, only structural attacks were performed. Following [182], node classification is performed by training a logistic regression on the learned embeddings. Overall, we see that direct attacks pose a much harder problem than influencer attacks. In these plots, we also compare against the two baselines Rnd and FGSM, both operating in the direct attack setting. As shown, Nettack outperforms both. Again note: All these results are obtained using a challenging poisoning attack (i.e., retraining of the model).

In Table 3.2 we summarize the results for different datasets and classification models. Here, we report the fraction of target nodes that get *correctly classified*. Our adversarial perturbations on the surrogate model are transferable to all three models an on all

	[1;5]	[6;10]	[11;20]	[21;100]	[100; ∞)
Clean Acc.	0.878	0.823	1.0	1.0	1.0
NETTACK	0.003	0.009	0.014	0.036	0.05

Table 3.3: Success rate of NETTACK for varying node degrees



(a) Direct attack **(b)** Influence attack

Figure 3.6: Attacks with limited knowledge about the data

datasets we evaluated. Not surprisingly, influencer attacks lead to a lower decrease in performance compared to direct attacks.

We see that FGSM performs worse than NETTACK, and we argue that this comes from the fact that gradient methods are not optimal for discrete data. Fig. 3.4 shows why this is the case: we plot the loss resulting from extrapolating using the gradient vs. the actual change in loss when changing an entry in $\boldsymbol{A}$. Often the gradients do not approximate the loss well – in the example depicted not even the sign of the changes are the same. One key advantage of NETTACK is that we can *precisely* and efficiently compute the change in $\mathcal{L}_s$.

Last, in Table 3.3, we also analyze how the structure of the target, i.e., its degree, affects the performance. As seen, high degree nodes are slightly harder to attack: they have higher classification accuracy both in the clean graph and in the attacked graph.

3.6.3.3 Limited Knowledge

In the previous experiments, we have assumed *full knowledge* of the input graph (but only a fraction of labels), which is a reasonable assumption for a worst-case attack. In Fig. 3.6 we analyze the result when having limited knowledge: Given a target node v_0, we provide our model only *subgraphs* of increasing size relative to the size of CORA-ML. We constructed these subgraphs by selecting nodes with increasing distance from v_0, i.e., we first selected 1-hop neighbors, then 2-hop neighbors and so on, until we have reached the desired graph size. We then perturb the subgraphs using the attack strategy proposed in this paper. These perturbations are then taken over to *full* graph, where we train GCN. *Note that* NETTACK *has always only seen the subgraph; and its surrogate model is also only trained based on it.*

Fig. 3.6 shows the result for a direct attack. As seen, even if only 10% of the graph is observed, we can still significantly attack it. Clearly, if the attacker knows the full graph, the fewest number of perturbations is required. For comparison we include the RND attack, also only operating on the subgraphs. In Fig. 3.6 we see the influence attack. Here we require more perturbations and 75% of the graph size for our attack to succeed. Still, this experiment indicates that full knowledge is not required.

3.7 Patterns of Adversarial Attacks

Our experiments in the previous section (as well as the experiments in [276]) show that adversarial attacks transfer from our surrogate model to different node classification algorithms. However, it is yet unclear what makes these adversarial attacks harmful to a variety of classification models. If we can find out what makes an edge insertion or deletion a strong adversarial change, we can use this knowledge to *detect* adversarial attacks and/or make graph neural networks more robust.

In this section we explore some patterns and regularities (e.g., *when inserting an edge, connect nodes from different classes*) and perform hypothesis testing to determine whether the patterns we identify are statistically significant. We focus on *direct structure attacks*, i.e., allowing only edge insertions and deletions to the target node, since these changes have the strongest effect on the classification (c.f. Fig. 3.2 and Table 3.2).

3.7.1 Statistical Analysis

In order to find patterns in structure attacks, we consider metrics based on the nodes incident to the edges that are being inserted or removed by the attacker. We consider two classes of metrics: (1) *absolute metrics* $m(w)$ that capture network metrics such as the node degree or centrality measures, and (2) *relative metrics* $m_{v_0}(w)$ that compare both nodes incident to the perturbed edge (e.g., the difference of the degrees of both nodes).

We consider our attacks to exhibit a pattern w.r.t. a metric m if its distribution among the nodes incident to inserted/removed edges significantly differs from some *baseline* that can be observed in the original (unperturbed) graph. An example (which we will test later) could be that the adversary connects nodes from different classes more often than the edges present in the original graph.

For our analysis, we uniformly sample $n = 400$ nodes $\mathcal{V}_s = \{v_1, \ldots, v_n\}$ from the graph and perform 5 direct structure perturbations per sampled node using NETTACK. For absolute metrics $m(w)$, the baseline is the distribution of the metric among the sampled nodes $\mathcal{V}_s$, which we then compare to the distribution of the nodes incident to edges inserted/removed by the attacker. For relative metrics (e.g., whether two nodes are from the same class) the baseline is the distribution among the neighbors of the sampled nodes. More formally, for a relative metric $m_{v_i}(w)$, the baseline is given by the targets' neighbors $\mathcal{N}(v_i)$ as $\frac{1}{|\mathcal{N}(v_i)|} \sum_{w \in \mathcal{N}(v_i)} m_{v_i}(w)$.

The metrics for all targets nodes $\mathcal{V}_s$, and nodes incident to inserted/removed edges can be viewed as samples of underlying distributions. Hence, we use hypothesis testing to

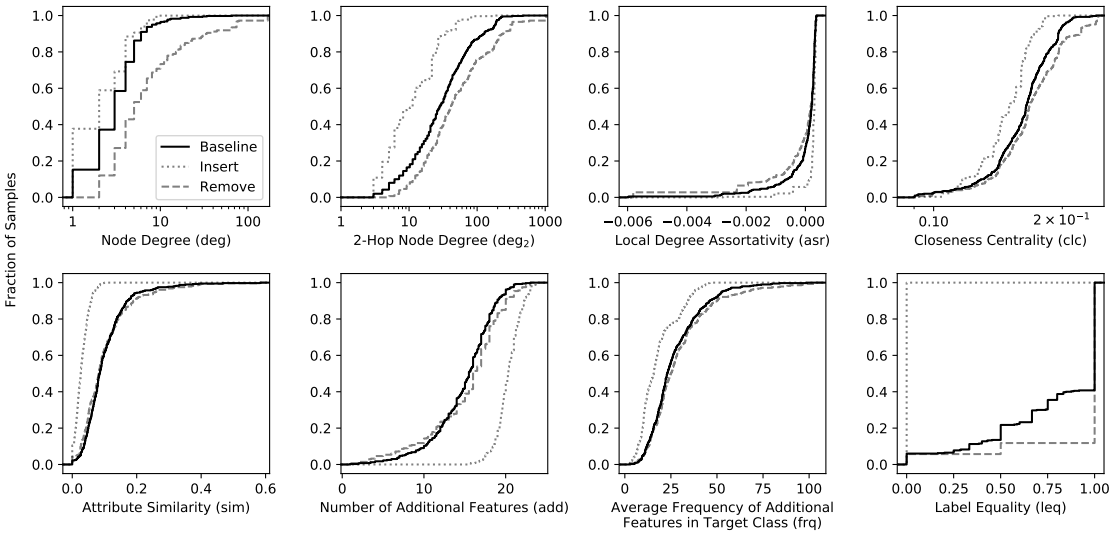


Figure 3.7: CDFs for distributions of different metrics when performing direct structure attacks on Cora-ML.

identify statistically significant deviations from the baseline. We use the nonparametric Mann-Whitney U-test [150] since the underlying distributions differ for the metrics and very rarely resemble a normal distribution (e.g., many properties on graphs follow a powerlaw distribution). Let X be a random variable distributed according to the baseline and Y a random variable describing the metric for inserted/removed edges. We formulate the null hypothesis $H_0 : \Pr[Y > X] \odot \frac{1}{2}$ ($\odot \in \{\leq, \geq\}$). When rejecting H_0 on a significance level $\alpha = 0.01$, we conclude that Y is stochastically larger/smaller than X to a significant extent.

Fig. 3.7 shows the results of our analysis for various metrics when attacking Cora-ML (the insights from the other datasets are similar). One qualitative insight we can draw from the figure is that the adversary tends to *insert* edges to nodes with a lower (two-hop) degree than the nodes in $\mathcal{V}_s$, which matches the results in [276]. The upper row of Fig. 3.7 displays absolute metrics which are defined for a node $v \in \mathcal{V}$ given an adjacency matrix $\boldsymbol{A}$:

$$\begin{aligned} \mathrm{asr}(v) &= \frac{r+1}{|\mathcal{V}|} - \frac{1}{|\mathcal{N}(v)|} \sum_{w \in \mathcal{N}(v)} |\deg(v) - \deg(w)| \\ \mathrm{clc}(v) &= \frac{|\mathcal{V}| - 1}{\sum_{w \in \mathcal{V} \setminus \{v\}} \mathrm{path}_{\min}(v, w)}, \end{aligned} \tag{3.22}$$

where we define the 2-hop node degree to be the sum of node degrees in a node's two-hop neighborhood and $\deg(v)$ is the degree of node v. The definition of the local degree assortativity is taken from [215], where r is the graph's degree assortativity as described in [168]. The lower row in Fig. 3.7 shows relative metrics. For a node w and target v, they can be formalized with an attribute matrix $\boldsymbol{X}$, a vector $\boldsymbol{y}$ with true labels of all

nodes, and a vector $\boldsymbol{f}_{y_v}$ with frequencies of all features among nodes assigned to the target class y_v:

$$\begin{aligned} \text{sim}_v(w) &= \frac{\boldsymbol{x}_v^T \boldsymbol{x}_w}{||\boldsymbol{x}_v + \boldsymbol{x}_w||_1 - \boldsymbol{x}_v^T \boldsymbol{x}_w} \qquad \text{add}_v(w) = ||\max(\boldsymbol{x}_w - \boldsymbol{x}_v, 0)||_0 \\ \text{frq}_v(w) &= \frac{\max\{\boldsymbol{x}_w - \boldsymbol{x}_v, 0\}^T \boldsymbol{f}_{y_v}}{||\max\{\boldsymbol{x}_w - \boldsymbol{x}_v, 0\}||_0} \qquad \text{leq}_v(w) = \mathbb{I}[y_v = y_w] \end{aligned} \tag{3.23}$$

For all metrics in Fig. 3.7, the appropriate null hypothesis can be rejected ($\alpha = 0.01$) when comparing the metric among the nodes to which edges are inserted to the baseline. However, we observe that finding significant patterns for the nodes incident to removed edges is generally hard since these nodes are inherently restricted to the original neighbors of the targets. For the metrics of removed edges, we can reject the null hypothesis for (2-hop) degree, closeness centrality, and label equality.

Unsurprisingly, the seemingly most prominent pattern is the label equality: edges are always inserted to nodes with a different label (and removed mostly between nodes with the same label). However, this alone cannot explain the harmfulness of Nettack's perturbations since the random baseline does exactly that but shows a much weaker effect on the classification as Nettack. Moreover, we found that when inserting adversarial edges to a target node v_0, the nodes the target is being connected to by the adversary often (around 95% and 88% on Cora-ML and Citeseer, respectively) share the same label (which is however different from the target node's label).

3.7.2 Estimating the Importance of Patterns via Ranking

In the previous section we have analyzed the statistical significance of patterns in adversarial attacks. In this section, we analyze the perturbations from a different perspective. We use the fact that Nettack assigns each (potential) perturbation a numerical score indicating their adversarial harmfulness and aim to *predict* a potential perturbation's adversarial impact using a linear model based on the values of the metrics introduced in the previous section. The rationale is that we can use these predictions to imitate Nettack and perform adversarial attacks to evaluate whether our patterns have successfully captured what makes adversarial attacks harmful to graph neural networks. More formally, the general idea to determine the importance of each pattern/metric is the following: for a specific target node v, we characterize all N possible perturbations via feature vectors $\boldsymbol{m}^{(1)}, \ldots, \boldsymbol{m}^{(N)}$. These vectors can be obtained by computing all metrics for the nodes in the graph. Further, we use Nettack to compute scores $s^{(1)}, \ldots, s^{(N)}$ for all perturbations as given by Eq. 3.15. The higher this score, the more suitable a perturbation is expected to be for confusing a classifier. Hence, the setting for our quantitative analysis is a classical supervised learning setting.

The fundamental idea is to train a linear model $f(\boldsymbol{m} \mid \boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{m}$ to predict the perturbations' scores. When preprocessing the features to follow roughly equal distributions, the relevance r_i of metric i is given as $0 \leq r_i = \frac{|\theta_i|}{||\boldsymbol{\theta}||_1} \leq 1$. The goal is to identify the metrics where r_i is sufficiently different from 0, i.e., absolute values above some threshold ε.

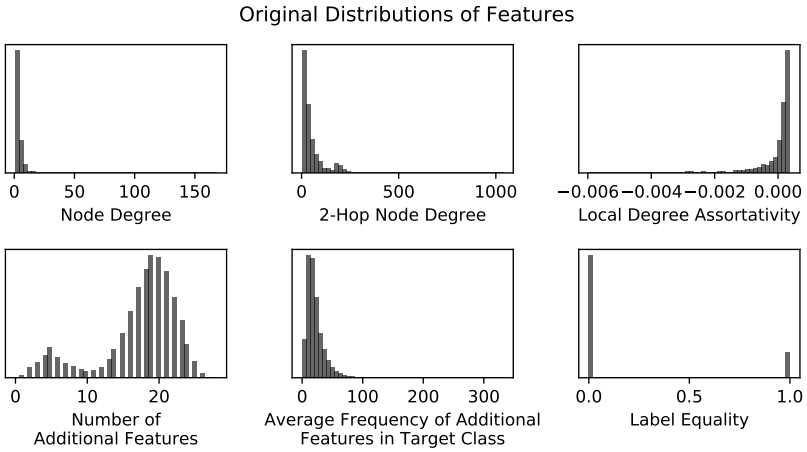


Figure 3.8: Distributions of relevant features before transformation.

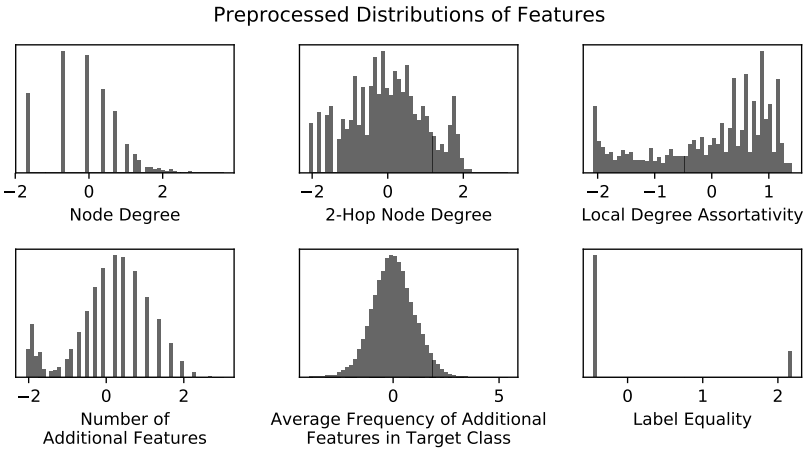


Figure 3.9: Result of Yeo-Johnson power transformation and standardization. Compared to Fig. 3.8, the histograms resemble normal distributions much more.

3.7.2.1 Feature Preprocessing

Overall, we are considering almost two dozen metrics to characterize perturbations (see Table B.1 in the appendix). However, they follow different families of distributions (e.g., powerlaw or normal distributions) and simple standardization to obtain distributions with zero mean and unit variance is not sufficient. In order to interpret the weights of a linear model in a useful way, we use the Yeo-Johnson power transformation (YJPT) [251]. This transformation depends on a parameter λ to perform a nonlinear transformation such that the data is roughly normally distributed [251].

For each metric independently, we use the YJPT and subsequent standardization (with parameters μ, σ) yields roughly standard normally distributed data. In Fig. 3.9 we can see the effect of the YJPT. Especially for the distributions that follow a power law (e.g., node degree), the processed distributions follow (almost) normal distributions.

The advantage of this nonlinear data transformation is not only better interpretability of the parameters of a linear model; also, such a model can be transferred to different graphs where the metrics' domains might be entirely different. When recomputing the preprocessing parameters λ, μ, σ for the a graph, the metrics still follow approximately the same (standard normal) distributions.

3.7.2.2 Ranking of Scores

The supervised learning setting to predict scores for perturbations suggests conventional linear regression. However, this is ill-suited for our problem, since we are generally only interested in the few potential perturbations with the highest scores. Moreover, we are not necessarily interested in the scores themselves but rather their order. Thus, we rephrase the problem as identifying the (few) most promising perturbations. That is, we want to learn to *rank* the perturbations according to their scores while the learned model's primary goal is to rank the most promising perturbations correctly.

We can achieve this by modifying the loss function while the model remains linear. Instead of the (root) mean squared error, we use the loss introduced in [242] to perform *listwise ranking*: in our case, the model is trained to predict scores for perturbations such that they are ranked correctly (per target node $v \in \mathcal{V}_s$, we refer to this set of perturbations as *batch*). Thus, the setup is the following: let $\mathcal{M}$ be a set of n batches $\boldsymbol{M}^{(1)}, \dots, \boldsymbol{M}^{(n)}$ with b_i perturbations each. Further, let $\mathcal{S}$ be a set of these batches' corresponding NETTACK scores $\boldsymbol{s}^{(1)}, \dots, \boldsymbol{s}^{(n)}$ for all perturbations. The vector $\boldsymbol{r}^{(i)}$ then describes the ranks $0, \dots, b_i - 1$ for all scores within the batch i. Given the (linear) model $f(\boldsymbol{m} \,|\, \boldsymbol{\theta})$, the loss $\mathcal{L}(\boldsymbol{\theta})$ is defined as follows (adapted from [242]):

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \left(\sum_{j=0}^{b_i-1} d(j) \left(f_j^{(i)} - \log \sum_{k=j}^{b_i-1} \exp(f_k^{(i)}) \right) \right) \quad \text{with} \quad f_j^{(i)} = f(\boldsymbol{m}^{(i)}_{r_j^{(i)}} \,|\, \boldsymbol{\theta}) \tag{3.24}$$

As shortly touched upon in [143], we introduce a quadratic *position discount factor* $d(j) = \frac{1}{(j+1)^2}$. As a result, the loss function is focused on predicting the ranks of the most important perturbations correctly.

3.7.2.3 Relevance of Metrics for Structure Perturbations

We evaluate the relevance of the metrics by training linear ranking models on CORA-ML, CITESEER, POLBLOGS, and PUBMED. For each graph, we sample 300 target nodes at random and compute the scores for all possible perturbations. We then train on perturbations and their scores computed by NETTACK. Here we focus only on perturbations that insert edges, since our statistical analysis indicates that these are most relevant. Further, strong L1-regularization encourages sparsity in the parameters, leaving only the few most relevant metrics with non-zero parameters.

The trained models' parameters are listed in Table B.1 (in the appendix). Generally, they are similar for the different graphs. Still, we find no more than half a dozen metrics to be of major importance (i.e., with absolute value of the corresponding parameters

Metric	deg	$\deg_2$	asr	add	frq	leq
Coefficient	+0.0190	−0.2937	+0.0190	+0.1632	−0.2173	−0.2435

Table 3.4: Coefficients of linear model for most relevant metrics when trained on Cora-ML.

$\gg 0$). These metrics are listed in Table 3.4 — the coefficients of the linear model are obtained when training on these metrics only. Especially note the strong linear correlation between the degree and the 2-hop degree here, resulting in an unexpected slightly positive coefficient for the degree. Surprisingly, all relative network metrics (e.g., the difference in node degree between a node and the target) are close to irrelevant. This corresponds to our finding that there often exist few nodes to which a large share of edges is inserted. This indicates that nodes to connect to depend on the adversary's target only to a small extent. Experiments on Polblogs (where no features are available) show that Nettack inserts edges to the same nodes almost for all targets. In principle, a defender could use such nodes as 'honeypots' to identify adversaries trying to fool the classification system.

Somewhat surprisingly, our analysis reveals that numerous computationally expensive metrics (e.g., closeness centrality, PageRank) have only little relevance in predicting the scores of perturbations. Thus, algorithms building on the patterns we have identified may discard these metrics, improving their scalability at very little (expected) drop in performance.

3.7.2.4 Relevance of Metrics for Feature Perturbations

As a proof of concept, we also explore training a linear model to predict the scores of feature perturbations. When training with the perturbations to insert features, the most promising features to insert are characterized by a relatively high overall frequency in the dataset, a very low frequency when considering only the nodes with the same label as the target, and a relatively high degree in the feature co-occurrence graph. When analyzing perturbations to remove features, exactly the opposite can be observed. Again, these findings are consistent across Cora-ML, Citeseer, and PubMed.

3.8 Scalable Attacks using Patterns in Perturbations

In this section we evaluate the patterns identified in the previous section. That is, we perform attacks using the linear model on the metrics identified as relevant. By being merely based on Nettack's patterns and by eliminating the need to train any kind of surrogate model, we expect to trade a slight decrease in effectiveness for a significant improvement in runtime performance. Hence, we introduce Fasttack.

Similarly to Nettack, our scalable algorithm Fasttack computes perturbations iteratively. Therefore, we can easily enforce the budget constraint (Eq. (3.5)) and restrict the algorithm to only perform unnoticeable perturbations (Eqs. (3.10) and (3.12)).

The general idea of FASTTACK is to use a pre-trained model to rank perturbations by their (predicted) impact on the classification of our surrogate model. We train the linear model as described in Sec. 3.7.2, using only the metrics we identified as relevant (see Table 3.4). Then, the algorithm iteratively chooses the best perturbation that still fulfills all constraints.

FASTTACK has two phases: a setup and an attack phase. During setup, the preprocessing parameters (YJPT + standardization) for the metrics are computed; while those for absolute metrics can be computed directly, we sample a set of node pairs to estimate the distribution of the relative metrics. Therefore, the setup phase only needs to be executed once per graph no matter how many nodes are attacked. During the attack phase, FASTTACK iteratively chooses the perturbation that maximizes its objective function (see Sec. 3.8.1).

Eventually, we need to consider that FASTTACK does not have access to all true labels while some metrics require labels for all nodes. Therefore, we use GCN (any other node classification algorithm could be used) to predict labels from a small subset of labeled nodes (note that this prediction can be done during the setup phase). When computing metrics based on the nodes' class memberships, we simply use class with the highest (log) probability as predicted by GCN. However, when evaluating the label equality between a node w and the target v, we use the softmax output vectors $\boldsymbol{z}_w$ and $\boldsymbol{z}_v$. With $\boldsymbol{z}_w^T \boldsymbol{z}_v$, we approximate the probability that the w and v share the same label. Lastly, all nodes FASTTACK connects the target to share the same (predicted) class label (as pre-computed with GCN), and this class label is always different from the target's class label.

A summary of FASTTACK's attack phase is depicted in Algorithm 4 in the appendix.

3.8.1 Adversarial Ratio

Up to this point, we have only evaluated how the most promising perturbations can be identified; we have not yet considered whether the notion of the best perturbations changes while modifying the graph. Indeed we notice that NETTACK's perturbation behavior changes over the course of the attack: when attacking a target node, NETTACK tends to initially insert adversarial edges, remove benign edges as soon as the number of perturbations equals the initial node degree, and afterwards only insert (adversarial) edges (c.f. Fig. 3.10a).

Given a target node v with incident edges $\mathcal{E}_v$, we therefore define the *adversarial ratio* as in Eq. 3.25 as the ratio of 'adversarial' edges $\mathcal{E}_v^-$ and 'benign' edges $\mathcal{E}_v^+$ (where $\mathcal{E}_v^- \cap \mathcal{E}_v^+ = \emptyset$ and $\mathcal{E}_v^- \cup \mathcal{E}_v^+ = \mathcal{E}_v$). We refer to an adversarial edge as an edge that benefits an adversary trying to confuse a classifier; likewise, a benign edge is beneficial to the classifier. Naturally, an adversary aims to maximize this ratio — which can be achieved by increasing the numerator (i.e., inserting an adversarial edge) or decreasing the denominator (i.e., removing a benign edge).

However, in practice we cannot exactly compute the adversarial ratio since edges cannot trivially be categorized as adversarial or benign. Hence, for an edge e, we introduce the (estimated) *adversarial probability* $p_e \in [0, 1]$ that indicates the probability of e (to

be inserted or removed) being adversarial. Likewise, $1 - p_e$ is the edge's probability of being benign. p_e can be computed by making use of the model $f(\boldsymbol{m} \mid \boldsymbol{\theta})$ which predicts the usefulness of perturbations (recall that each perturbation is coupled with a unique edge). First, the model's predicted scores for all perturbations are scaled into the range $[-1, +1]$. Second, we use a rescaled sigmoid function $\sigma(z) = \frac{1}{1+e^{-zk}}$ with $k = 9/2$ (chosen experimentally) to obtain what we refer to as the adversarial probability.

By introducing the sigmoid function as nonlinearity, we capture the intuition that a high (low) score predicted by the model indicates a high (low) probability of the edge being adversarial (benign). In practice, we found that the adversarial ratio $\xi(v)$ of a target v can be reasonably approximated as

$$\xi(v) = \log \frac{1 + |\mathcal{E}_v^-|}{1 + |\mathcal{E}_v^+|} \approx \log \left(1 + \sum_{e \in \mathcal{E}_v} p_e\right) - \log \left(1 + \sum_{e \in \mathcal{E}_v} 1 - p_e\right) = \xi'(\mathcal{E}_v) \tag{3.25}$$

In the attack phase, FASTTACK iteratively (and greedily) chooses the edge e^* from all edges associated with perturbations as

$$e^* = \arg\max_e \xi'(\mathcal{E}_v \oplus \{e\}) \tag{3.26}$$

where $\oplus$ is the symmetric difference, i.e., edge removal if the edge is present and otherwise insertion. Fig. 3.10 shows that, indeed, the adversarial ratio helps to reasonably approximate NETTACK's behavior.

3.8.2 Complexity

When analyzing the theoretical runtime complexity of FASTTACK, we refer to N as the number of nodes, M the number of edges, D the number of features, and Δ the number of perturbations. The entire algorithm's complexity is then given as $\mathcal{O}(M + ND + N \log(N) + \Delta)$. Here, the term $\mathcal{O}(M + ND)$ is caused by the setup phase and the computation of the metrics for the target node. The term $\mathcal{O}(N \log N)$ results from sorting perturbations by their score. Eventually, each perturbation can be computed in constant time as the adversarial ratio can be updated in constant time. Hence, the attack itself only requires $\mathcal{O}(\Delta)$ operations.

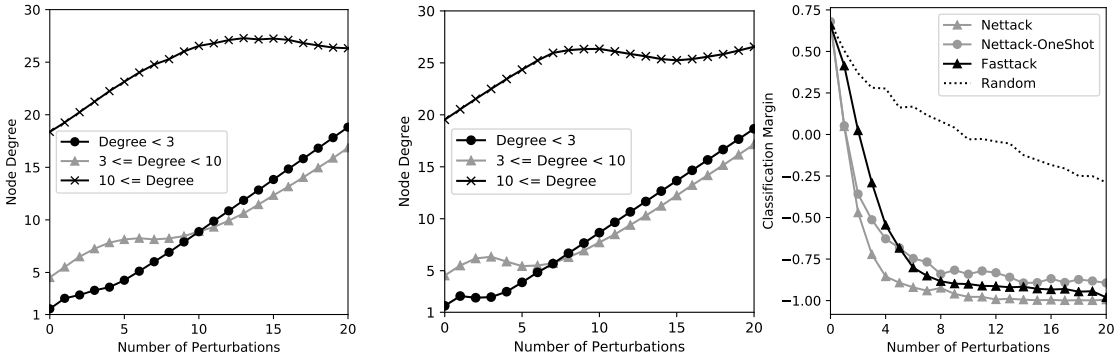
In most cases, however, graphs are sparse, i.e., $M \approx N$, just as the attribute matrices, i.e., $ND \approx N$. Further, we normally consider $\Delta \ll N$, hence a more suitable theoretical runtime complexity of FASTTACK is simply given as $\mathcal{O}(N \log N)$.

3.8.3 Experimental Evaluation

We want to explore the effectiveness of the identified patterns by comparing FASTTACK to NETTACK and a random attack algorithm.

3.8.3.1 Effectiveness

At first, we evaluate FASTTACK's behavior over the course of multiple edge perturbations. For this, we use CORA-ML and split its nodes into 10% training, 10% validation, and



(a) Attack using NETTACK **(b)** Attack using FASTTACK **(c)** Avg. attack performance.

Figure 3.10: Degree of nodes with different initial degrees when attacked in CORA-ML (**(a)** and **(b)**), and average reduction of classification margin for different attacks **(c)**.

80% test set. We then randomly sample 25 nodes from the test set which we attack with FASTTACK, NETTACK, NETTACK-ONESHOT, and a random approach. Here, NETTACK-ONESHOT is a modification of NETTACK which computes the scores for perturbations only once at the beginning and chooses the perturbations with the highest scores for an attack (while satisfying the same constraints as NETTACK and FASTTACK). In contrast to all other algorithms, the random approach has access to all true labels and inserts edges only between nodes with different label and removes edges only between nodes with the same one (other works refer to this as "DICE" (delete internally, connect externally), e.g., [276]). The model that FASTTACK uses for ranking perturbations is trained on 300 random nodes from the test set, disjoint from the attacked nodes.

We perform 20 perturbations for each target node and use GCN to predict the labels of the attacked nodes after every perturbation. Fig. 3.10c displays the average classification margins for the attacked nodes. While NETTACK leads (on average) to a stronger decrease in classification margins in the first few perturbations, FASTTACK is almost as strong with an increasing number of perturbations, and clearly stronger than the random baseline. Generally, our scalable algorithm needs 1–2 perturbations more to lead to misclassification (with significant margin). Surprisingly FASTTACK is able to surpass NETTACK-ONESHOT in the long run (i.e., for $\Delta \geq 5$). This highlights the effectiveness of the adversarial ratio introduced above.

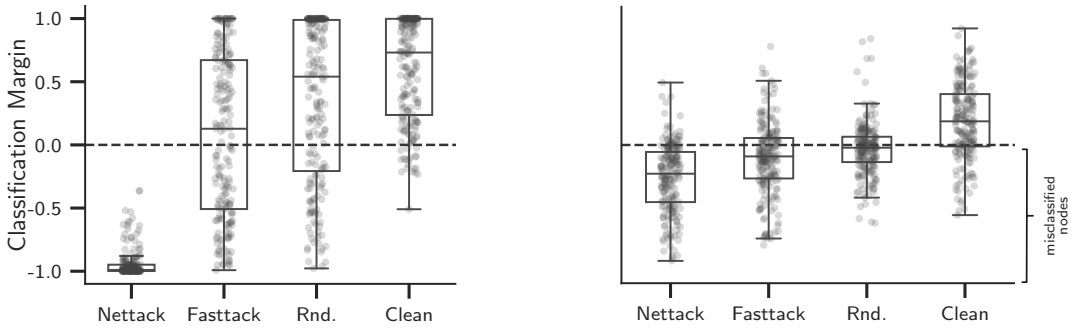
3.8.3.2 Transferability

In this section, we want to focus on FASTTACK's ability to generalize to different deep learning models for classification.

For this, we use CORA-ML, CITESEER, POLBLOGS, and PUBMED where we choose 5 different splits of nodes per graph (10% training, 10% validation, 80% test). For each split, we choose 40 target nodes from the test set as described in Section 3.6 and attack these using FASTTACK, NETTACK, and the random attack algorithm introduced above.

	Cora-ML			Citeseer			Polblogs			PubMed		
	GCN	CLN	DW	GCN	CLN	DW	GCN	CLN	DW	GCN	CLN	DW
Clean	0.88	0.83	0.82	0.86	0.85	0.73	0.90	0.89	0.94	0.89	0.85	0.80
Random	0.71	0.73	0.41	0.69	0.76	0.44	**0.40**	**0.53**	**0.29**	0.68	0.79	**0.42**
Fasttack	**0.41**	**0.62**	**0.35**	**0.34**	**0.59**	**0.36**	0.43	0.70	0.58	**0.53**	**0.78**	0.56
Nettack	0.05	0.35	0.15	0.07	0.39	0.16	0.16	0.52	0.26	0.00	0.55	0.16

Table 3.5: Classification accuracy of classification models when trained on datasets poisoned by Fasttack (perturbation budget $\Delta = d$). Smaller is better.



(a) PubMed: Attack GCN **(b)** Citeseer: Attack DW

Figure 3.11: Results of poisoning attacks with perturbation budget $\Delta = d$. Clean indicates the original data. Lower scores are better.

In each attack, we perform $\Delta = d_{v_0}$ perturbations (as compared to $\Delta = d_{v_0} + 2$ in Section 3.6), where d_{v_0} is the degree of the respective target node.

For Fasttack, we use the splits for Cora-ML and train 5 different linear models as outlined in Sec. 3.7.2.2. To obtain a set of nodes for training, we use the test sets of each split (separately) and uniformly sample 300 nodes (disjoint from the target nodes). When attacking Cora-ML, we always use the model trained on the respective split. For all other graphs, we use a transferred model, where we average the parameters of the models trained on the different splits of Cora-ML, resulting in the parameter vector listed in Table 3.4. By transferring the models trained on Cora-ML, we implicitly test for transferability of the patterns we have identified.

Table 3.5 displays the results of this experiment when measuring the classification accuracy among the target nodes for different deep learning models for classification. While, not surprisingly, Nettack leads to the strongest decrease in performance, Fasttack is able to outperform the random approach (which has access to *all* class labels) in most cases. On Polblogs, Fasttack achieves weaker results than on the other datasets. A likely cause is the model transferred from Cora-ML: while Cora-ML has node attributes, Polblogs does not (and hence Fasttack is missing values for these metrics). Fig. 3.11 further highlights the effectiveness of Fasttack – remember again that the model used for ranking the perturbations is trained on an entirely different graph. This highlights the meaningfulness of the patterns we identified.

3.8.3.3 Runtime

After deriving the theoretical runtime complexity of FASTTACK in Sec. 3.8.2, we lastly want to evaluate its practical runtime performance. For this, we again use a pre-trained model for FASTTACK. FASTTACK and NETTACK then attack random nodes in CORA-ML, and POLBLOGS, and PUBMED while performing 20 structure perturbations every time. The results of this experiments are displayed in Fig. 3.12. They are obtained by measuring the runtime when running on a CPU (Quad-Core Intel Core i7 Skylake, 2.9 GHz) and 16 GB of RAM.

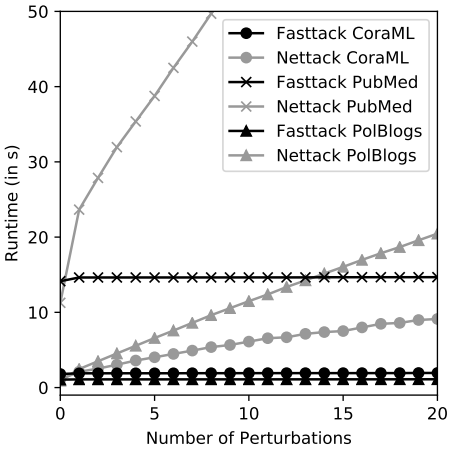


Figure 3.12: Runtime comparison.

FASTTACK significantly outperforms NETTACK in terms of runtime, allowing it to scale to much larger graphs – even though we include initial label training via GCN (prior to any perturbations) in the runtime analysis of Fig. 3.12. The slope of the plots indicate that runtime performance of FASTTACK itself is markedly faster than NETTACK. Most notably, FASTTACK is much more suitable to attack large graphs and high-degree nodes. We conclude that, indeed, a slight drop in effectiveness allows FASTTACK to be very scalable.

3.9 Conclusion

We presented a method for adversarial attacks to (attributed) graphs, specifically focusing on the task of node classification via graph convolutional networks. Our attacks target the nodes' features and the graph structure. Exploiting the relational nature of the data, we proposed direct and influencer attacks. To ensure unnoticeable changes even in a discrete, relational domain, we proposed to preserve the graph's degree distribution and feature co-occurrences. Our algorithm enables efficient perturbations in a discrete domain. Based on our extensive experiments we can conclude that even the challenging poisoning attack is possible with our approach. The classification performance is consistently reduced, even when only partial knowledge of the graph is available or the attack is restricted to a few influencers. Even more, the attacks generalize to other node classification models.

We further identify statistically significant patterns in the adversarial perturbations of NETTACK. We exploit these to design FASTTACK — a highly scalable adversarial attack algorithm. FASTTACK's effectiveness in reducing the classification performance of graphs it was not trained on highlights the meaningfulness of the patterns we dicovered. This represents the first leap towards methods being able to detect and prevent adversarial attacks on graph neural networks.

Retrospective

In this chapter we propose the first adversarial attack method for graph neural networks. As we have shown and as confirmed by a number of later studies [e.g., 268, 69, 237] the attack method is highly effective at reducing the performance of different GNNs and even node embedding methods. In hindsight, we can identify a number of open questions, limitations, and potential improvements. While our attack method focuses on poisoning attacks, it does not explicitly account for the bilevel nature of the corresponding optimization problem. We address this limitation in our follow-up study, which we present in the following chapter. Moreover, with the recent Open Graph Benchmark (OGB) [106] we have much larger high-quality node classification datasets on which we could evaluate our attack method. An aspect which has not been analyzed yet is that the surrogate model used in this chapter is effectively a logistic regression model on the node attributes smoothed via two message passing steps with $\hat{\boldsymbol{A}}$. Thus, the surrogate model is convex (in the node attributes), which could be exploited by attack or defense methods. Finally, the unnoticeability constraint we propose in addition to the budget constraint is based on a global property of the graph (the degree distribution), while the attack is local, i.e., targets individual nodes. How to develop unnoticeability constraints on graphs which better fit the nature of the attacks is still an open question.

4 Adversarial Attacks on Graph Neural Networks via Meta Learning

4.1 Introduction

In the previous chapter we have shown that graph neural networks are vulnerable to adversarial attacks both at test time (*evasion*) as well as training time (*poisoning* attacks). A core strength of graph neural networks – exploiting the information in a node's neighborhood to improve classification – is also a major vulnerability: because of these propagation effects, an attacker can change a single node's prediction without even changing any of its attributes or edges. This is because the foundational assumption that all samples are *independent* of each other does not hold for node classification. Network effects such as homophily [145] support the classification, while on the other hand they enable indirect adversarial attacks.

The attack methods on node classification models presented in the previous chapter existing attacks are *local*, that is, aim to provoke misclassification of a specific *single* node, e.g., a person in a social network. In this chapter, we propose the first algorithm for poisoning attacks that is able to compromise the *global* node classification performance of a model, e.g., as measured by the classification accuracy on the whole test set. We show that even under restrictive attack settings and without access to the target classifier, our attacks can render it near-useless for use in production (i.e., on test data).

In contrast to the previous chapter, here we explicitly account for the bilevel nature of poisoning attacks. We do so by basing our approach on the principle of meta learning, which has traditionally been used for hyperparameter optimization [14], or, more recently, few-shot learning [77]. In essence, we turn the gradient-based optimization procedure of deep learning models upside down and treat the input data – the graph at hand – as a hyperparameter to learn.

4.2 Related Work

Adversarial attacks on machine learning models have been studied both in the machine learning and security community and for many different model types [155]. It is important to distinguish attacks from outliers; while the latter naturally occur in graphs [21], adversarial examples are deliberately created with the goal to mislead machine learning models and often designed to be unnoticeable. Deep neural networks are highly sensitive to these small adversarial perturbations to the data [212, 88]. The vast majority of attacks and defenses assume the data instances to be independent and continuous. This assumption clearly does not hold for node classification and many other tasks on graphs.

Works on adversarial attacks for graph learning tasks are generally sparse. [46] have measured the changes in the resulting graph clustering when injecting noise to a bipartite graph that represent DNS queries. However, their focus is not on generating attacks in a principled way. [219] consider adversarial noise in the node features in order to improve robustness of collective classification via associative Markov networks.

Only recently researchers have started to study adversarial attacks on deep learning for graphs. Dai et al. [56] consider *test-time* (i.e., evasion) attacks on graph classification (i.e., classification of graphs themselves) and node classification. However, they do not consider poisoning (i.e., training-time) attacks or evaluate transferability of their attacks, and restrict the attacks to edge deletions only. Moreover, they focus on targeted attacks, i.e., attacks designed to change the prediction of a single node. Zügner et al. [273] consider both test-time and training-time attacks on node classification models. They circumvent explicitly tackling the bilevel optimization problem underlying poisoning attacks by performing their attacks based on a (static) surrogate model and evaluating their impact by training a classifier on the data modified by their algorithm. In contrast to [56], their attacks can both insert and remove edges, as well as modify node attributes in the form of binary vectors. Again, their algorithm is suited only to targeted attacks on single nodes; the problem of training-time attacks on the overall performance of node classification models remains unexplored. Bojchevski and Günnemann [23] propose poisoning attacks on a different task: unsupervised node representation learning (or node embeddings). They exploit perturbation theory to maximize the loss obtained after training DeepWalk. In this chapter, we focus on semi-supervised learning.

Meta-learning [216, 167], or *learning to learn*, is the task of optimizing the learning algorithm itself; e.g., by optimizing the hyperparameters [14], learning to update the parameters of a neural network [197, 13], or the activation function of a model [2]. Gradient-based hyperparameter optimization works by differentiating the training phase of a model to obtain the gradients w.r.t. the hyperparameters to optimize.

The key idea of this chapter is to use meta-learning for the opposite: modifying the training data to *worsen* the performance after training (i.e., training-time or poisoning attacks). [165] demonstrate that meta learning can indeed be used to create training-time attacks on simple, linear classification models. On continuous data, they report little success when attacking deep neural networks, and on discrete datasets, they do not consider deep learning models or problems with more than two classes. Like most works on adversarial attacks, they assume the data instances to be independent. In this chapter, for the first time, we propose an algorithm for global attacks on (deep) node classification models at training time. In contrast to [273], we explicitly tackle the bilevel optimization problem of poisoning attacks using meta learning.

4.3 Problem Formulation

We consider the task of (semi-supervised) node classification. Given a single (attributed) graph and a set of labeled nodes, the goal is to infer the class labels of the unlabeled nodes. Formally, let $G = (A, X)$ be an attributed graph with adjacency matrix $A \in$

$\{0,1\}^{N\times N}$ and node attribute matrix $X \in \mathbb{R}^{N\times D}$, where N is the number of nodes and D the dimension of the node feature vectors. W.l.o.g., we assume the node IDs to be $\mathcal{V} = \{1,\ldots,N\}$.

Given the set of labeled nodes $\mathcal{V}_L \subseteq \mathcal{V}$, where nodes are assigned exactly one class in $\mathcal{C} = \{c_1, c_2, ..., c_K\}$, the goal is to learn a function f_θ, which maps each node $v \in \mathcal{V}$ to exactly one of the K classes in $\mathcal{C}$ (or in a probabilistic formulation: to the K-simplex). Note that this is an instance of transductive learning, since *all* test samples (i.e., the unlabeled nodes) as well as their attributes and edges (but not their class labels!) are known and used during training [41]. The parameters θ of the function f_θ are generally learned by minimizing a loss function $\mathcal{L}_{\text{train}}$ (e.g., cross-entropy) on the labeled training nodes:

$$\theta^* = \arg\min_\theta \mathcal{L}_{\text{train}}(f_\theta(G)), \tag{4.1}$$

where we overload the notation of f_θ to indicate that we feed in the whole graph G.

4.3.1 Attack Model

Adversarial attacks are small deliberate perturbations of data samples in order to achieve the outcome desired by the attacker when applied to the machine learning model at hand. The attacker is constrained in the knowledge they have about the data and the model they attack, as well as the adversarial perturbations they can perform.

Attacker's goal. In our work, the attacker's goal is to increase the misclassification rate (i.e., one minus the accuracy) of a node classification algorithm achieved after training on the data (i.e., graph) modified by our algorithm. In contrast to [273] and [56], our algorithm is designed for *global* attacks reducing the overall classification performance of a model. That is, the goal is to have the test samples classified as *any* class different from the true class.

Attacker's knowledge. The attacker can have different levels of knowledge about the training data, i.e. the graph G, the target machine learning model $\mathcal{M}$, and the trained model parameters θ. In our work, we focus on limited-knowledge attacks where the attacker has no knowledge about the classification model and its trained weights, but the same knowledge about the data as the classifier. In other words, the attacker can observe all nodes' attributes, the graph structure, as well as the labels of the subset $\mathcal{V}_L$ and uses a surrogate model to modify the data. Besides assuming knowledge about the full data, we also perform experiments where only a subset of the data is given. Afterwards, this modified data is used to train deep neural networks to degrade their performance.

Attacker's capability. In order to be effective and remain undiscovered, adversarial attacks should be *unnoticeable*. To account for this, we largely follow the attacker capabilities from Chapter Chapter 3. First, we impose a budget constraint Δ on the attacks, i.e., limit the number of changes $\|A - \hat{A}\|_0 \leq \Delta$ (here we have 2Δ since we assume the

graph to be symmetric). Furthermore, we make sure that no node becomes disconnected (i.e. a singleton) during the attack. One of the most fundamental properties of a graph is its degree distribution. Any significant changes to it are very likely to be noticed; to prevent such large changes to the degree distribution, we employ [273]'s unnoticeability constraint on the degree distribution. Essentially, it ensures that the graph's degree distribution can only marginally be modified by the attacker. The authors also derive an efficient way to check for violations of the constraint so that it adds only minimal computational overhead to the attacks. While in this chapter we focus on changing the graph structure only, our algorithm can easily be adapted to change the node features as well. We summarize all these constraints and denote the set of admissible perturbations on the data as $\Phi(G)$, where G is the graph at hand.

4.3.2 Overall Goal

Poisoning attacks can be mathematically formulated as a bilevel optimization problem:

$$\min_{\hat{G} \in \Phi(G)} \quad \mathcal{L}_{\text{atk}}(f_{\theta^*}(\hat{G})) \quad s.t. \quad \theta^* = \arg\min_{\theta} \quad \mathcal{L}_{\text{train}}(f_\theta(\hat{G})). \tag{4.2}$$

$\mathcal{L}_{\text{atk}}$ is the loss function the attacker aims to optimize. In our case of global and unspecific (regarding the type of misclassification) attacks, the attacker tries to decrease the *generalization performance of the model on the unlabeled nodes*. Since the test data's labels are not available, we cannot directly optimize this loss. One way to approach this is to maximize the loss on the labeled (training) nodes $\mathcal{L}_{\text{train}}$, arguing that if a model has a high training error, it is very likely to also generalize poorly (the opposite is not true; when overfitting on the training data, a high generalization loss can correspond to a low training loss). Thus, our first attack option is to choose $\mathcal{L}_{\text{atk}} = -\mathcal{L}_{\text{train}}$.

Recall that semi-supervised node classification is an instance of *transductive* learning: all data samples (i.e., nodes) and their attributes are known at training time (but not all labels!). We can use this insight to obtain a second variant of $\mathcal{L}_{\text{atk}}$. The attacker can learn a model on the labeled data to estimate the labels $\hat{C}_U$ of the unlabeled nodes $\mathcal{V}_U = \mathcal{V} \backslash \mathcal{V}_L$. The attacker can now perform self-learning, i.e. use these predicted labels and compute the loss of a model on the unlabeled nodes, yielding our second option $\mathcal{L}_{\text{atk}} = -\mathcal{L}_{\text{self}}$ where $\mathcal{L}_{\text{self}} = \mathcal{L}(\mathcal{V}_U, \hat{C}_U)$. Note that, at all times, only the labels of the labeled nodes are used for training; $\mathcal{L}_{\text{self}}$ is only used to estimate the generalization loss after training. In our experimental evaluation, we compare both versions of $\mathcal{L}_{\text{atk}}$ outlined above.

Importantly, notice the bilevel nature of the problem formulation in Eq. (4.2): the attacker aims to maximize the classification loss achieved *after* optimizing the model parameters on the modified (poisoned) graph $\hat{G}$. Optimizing such a bilevel problem is highly challenging by itself. Even worse, in our graph setting the data and the action space of the attacker are *discrete*: the graph structure is $A = \{0, 1\}^{N \times N}$, and the possible actions are edge insertions and deletions. This makes the problem even more difficult in two ways. First, the action space is vast; given a budget of Δ perturbations, the number of possible attacks is, ignoring symmetry, $\binom{N^2}{\Delta}$ and thus in $O(N^{2\Delta})$; exhaustive

search is clearly infeasible. Second, a discrete data domain means that we cannot use gradient-based methods such as gradient descent to make *small* (real-valued) updates on the data to optimize a loss.

4.4 Graph Structure Poisoning via Meta-Learning

4.4.1 Poisoning via Meta-gradients

In this chapter, we tackle the bilevel problem described in Eq. (4.2) using meta-gradients, which have traditionally been used in meta-learning. The field of meta-learning (or learning to learn) tries to make the process of learning machine learning models more time and/or data efficient, e.g., by finding suitable hyperparameter configurations [14] or initial weights that enable rapid adaptation to new tasks or domains in few-shot learning [77].

Meta-gradients (e.g., gradients w.r.t. hyperparameters) are obtained by backpropagating *through* the learning phase of a differentiable model (typically a neural network). The core idea behind our adversarial attack algorithm is **to treat the graph structure matrix as a hyperparameter** and compute the gradient of the attacker's loss *after training* with respect to it:

$$\nabla_G^{\text{meta}} := \nabla_G \, \mathcal{L}_{\text{atk}}(f_{\theta^*}(G)) \quad s.t. \quad \theta^* = \text{opt}_\theta(\mathcal{L}_{\text{train}}(f_\theta(G))), \tag{4.3}$$

where $\text{opt}(\cdot)$ is a differentiable optimization procedure (e.g. gradient descent and its stochastic variants) and $\mathcal{L}_{\text{train}}$ the training loss. Notice the similarity of the meta-gradient to the bi-level formulation in Eq. (4.2); the meta-gradient indicates how the attacker loss $\mathcal{L}_{\text{atk}}$ *after training* will change for small perturbations on the data, which is exactly what a poisoning attacker needs to know.

As an illustration, consider an example where we instantiate opt with vanilla gradient descent with learning rate α starting from some intial parameters θ_0

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(G)) \tag{4.4}$$

The attacker's loss after training for T steps is $\mathcal{L}_{\text{atk}}(f_{\theta_T}(G))$. The meta-gradient can be expressed by unrolling the training procedure:

$$\nabla_G^{\text{meta}} = \nabla_G \mathcal{L}_{\text{atk}}(f_{\theta_T}(G)) = \nabla_f \mathcal{L}_{\text{atk}}(f_{\theta_T}(G)) \cdot \left[\nabla_G f_{\theta_T}(G) + \nabla_{\theta_T} f_{\theta_T}(G) \cdot \nabla_G \theta_T\right], \text{where}$$
$$\nabla_G \theta_{t+1} = \nabla_G \theta_t - \alpha \nabla_G \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(G)) \tag{4.5}$$

Note that the parameters θ_t itself depend on the graph G (see Eq. 4.4); they are *not fixed*. Thus, the derivative w.r.t. the graph has to be taken into account, chaining back until θ_0. Given this, the attacker can use the meta-gradient to perform a meta update M on the data to minimize $\mathcal{L}_{\text{atk}}$:

$$G^{(k+1)} \leftarrow M(G^{(k)}) \tag{4.6}$$

Algorithm 2: Poisoning attack on GNNs with meta gradients and self-training.

Input: Graph $G = (A, X)$, modification budget Δ, number of training iterations T, training class labels C_L

Output: Modified graph $\hat{G} = (\hat{A}, X)$

$\hat{\theta} \leftarrow$ train surrogate model on the input graph using known labels C_L;
$\hat{C}_U \leftarrow$ predict labels of unlabeled nodes using $\hat{\theta}$;
$\hat{A} \leftarrow A$;
while $\|\hat{A} - A\|_0 < 2\Delta$ **do**
  randomly initialize θ_0;
  for t *in* $0 \dots T-1$ **do**
    $\theta_{t+1} \leftarrow \text{step}\left(\theta_t, \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{A}, X)); C_L\right)$; `// update e.g. via gradient descent`
  `// Compute meta gradient via backprop through the training procedure`
  $\nabla^{\text{meta}}_{\hat{A}} \leftarrow \nabla_{\hat{A}} \mathcal{L}_{\text{self}}(f_{\theta_T}(\hat{A}, X); \hat{C}_U)$;
  $S \leftarrow \nabla^{\text{meta}}_{\hat{A}} \odot (-2\hat{A} + 1)$; `// Flip gradient sign of node pairs with edge`
  $e' \leftarrow$ maximum entry (u, v) in S that fulfills constraints $\Phi(G)$;
  $\hat{A} \leftarrow$ insert or remove edge e' to/from $\hat{A}$;
$\hat{G} \leftarrow (\hat{A}, X)$;
return : $\hat{G}$

The final poisoned data $G^{(\Delta)}$ is obtained after performing Δ meta updates. A straightforward way to instantiate M is (meta) gradient descent with some step size β: $M(G) = G - \beta \nabla_G \mathcal{L}_{\text{atk}}(f_{\theta_T}(G)))$.

It has to be noted that such a gradient-based update rule is neither possible nor well-suited for problems with discrete data (such as graphs). Due to the discreteness, the gradients are not defined. Thus, in our approach we simply relax the data's discreteness condition. However, we still perform discrete updates (actions) since the above simple gradient update would lead to dense (and continuous) adjacency matrices; not desired and not efficient to handle. Thus, in the following section, we propose a greedy approach to preserve the data's sparsity and discreteness.

4.4.2 Greedy Poisoning Attacks via Meta Gradients

We assume that the attacker does not have access to the target classifier's parameters, outputs, or even knowledge about its architecture; the attacker thus uses a *surrogate* model to perform the poisoning attacks. Afterwards the poisoned data is used to train deep learning models for node classification (e.g. a GCN) to evaluate the performance degradation due to the attack. We use the same surrogate model as [273], which is a linearized two-layer graph convolutional network:

$$f_\theta(A, X) = \text{softmax}(\hat{A}^2 X W), \tag{4.7}$$

where $\hat{A} = D^{-1/2} \tilde{A} D^{-1/2}$, $\tilde{A} = A + I$, A is the adjacency matrix, X are the node features, D the diagonal matrix of the node degrees, and $\theta = \{W\}$ the set of learnable parameters. In contrast to [273] we do not linearize the output (softmax) layer.

Algorithm 3: GNN poisoning with approximate meta gradients, self-training.

Input: Graph $G = (A, X)$, modification budget Δ, number of training iterations T, gradient weighting λ, training class labels C_L
Output: Modified graph $\hat{G} = (\hat{A}, X)$

$\hat{\theta} \leftarrow$ train surrogate model on the input graph using known labels C_L;
$\hat{C}_U \leftarrow$ predict labels of unlabeled nodes using $\hat{\theta}$;
$\hat{A} \leftarrow A$;
while $\|\hat{A} - A\|_0 < 2\Delta$ **do**
  randomly initialize θ_0;
  $\nabla^{\text{meta}}_{\hat{A}} \leftarrow \lambda \nabla_{\hat{A}} \mathcal{L}_{\text{train}}(f_{\theta_0}(\hat{A}; X); C_L) + (1-\lambda) \nabla_{\hat{A}} \mathcal{L}_{\text{self}}(f_{\theta_0}(\hat{A}; X); \hat{C}_U)$
  for t *in* $0 \dots T-1$ **do**
    $\theta_{t+1} \leftarrow \text{step}\,(\theta_t, \nabla_{\theta_t} \mathcal{L}_{\text{train}}(f_{\theta_t}(\hat{A}, X)); C_L)$; `// update e.g. via gradient descent`
    $\tilde{\theta}_{t+1} \leftarrow \text{stop\_gradient}(\theta_{t+1})$; `// no backprop through training`
    $\nabla^{\text{meta}}_{\hat{A}} \leftarrow \nabla^{\text{meta}}_{\hat{A}} + \lambda \nabla_{\hat{A}} \mathcal{L}_{\text{train}}(f_{\tilde{\theta}_{t+1}}(\hat{A}; X); C_L) + (1-\lambda) \nabla_{\hat{A}} \mathcal{L}_{\text{self}}(f_{\tilde{\theta}_{t+1}}(\hat{A}; X); \hat{C}_U)$
  $S \leftarrow \nabla^{\text{meta}}_{\hat{A}} \odot (-2\hat{A} + 1)$; `// Flip gradient sign of node pairs with edge`
  $e' \leftarrow$ maximum entry (u, v) in S that fulfills constraints $\Phi(G)$;
  $\hat{A} \leftarrow$ insert or remove edge e' to/from $\hat{A}$;
$\hat{G} \leftarrow (\hat{A}, X)$;
return : $\hat{G}$

Note that we only perform changes to the graph structure A, hence we treat the node attributes X as a constant during our attacks. For clarity, we replace G with A in the meta gradient formulation.

We define a score function $S : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ that assigns each possible action a numerical value indicating its (estimated) impact on the attacker objective $\mathcal{L}_{\text{atk}}$. Given the meta-gradient for a node pair (u, v), we define $S(u, v) = \nabla^{\text{meta}}_{a_{uv}} \cdot (-2 \cdot a_{uv} + 1)$ where a_{uv} is the entry at position (u, v) in the adjacency matrix A. We essentially flip the sign of the meta-gradients for connected node pairs as this yields the gradient for a change in the negative direction (i.e., removing the edge).

We greedily pick the perturbation $e' = (u', v')$ with the highest score one at a time

$$e' = \underset{e=(u,v):\, M(A,e) \in \Phi(G)}{\arg\max} S(u, v), \tag{4.8}$$

where $M(A, e) \in \Phi(G)$ ensures that we only perform changes compliant with our attack constraints (e.g., unnoticeability). The meta update function $M(A, e)$ inserts the edge $e = (i, j)$ by setting $a_{ij} = 1$ if nodes (i, j) are currently not connected and otherwise deletes the edge by setting $a_{ij} = 0$. We show a summary of our algorithm in Alg. 2.

4.4.3 Approximating Meta-Gradients

Computing the meta gradients is expensive both from a computational and a memory point-of-view. A simple first-order approximation is

$$\nabla^{\text{meta}}_{A} = \nabla_A \mathcal{L}_{\text{atk}}(f_{\theta_T}(A)) \approx \nabla_A \mathcal{L}_{\text{atk}}(f_{\tilde{\theta}_T}(A)) = \nabla_f \mathcal{L}_{\text{atk}}(f_{\tilde{\theta}_T}(A)) \cdot \nabla_A f_{\tilde{\theta}_T}(A). \tag{4.9}$$

	Cora-ML		Citeseer	
	GCN	**CLN**	**GCN**	**CLN**
Clean	16.6±0.3	17.3±0.3	28.5±1.0	28.3±0.8
A-Meta-Train	21.2±0.9	20.3±0.3	31.8±0.8	29.8±0.5
A-Meta-Self	21.8±0.7	18.9±0.3	28.6±0.4	28.5±0.4
A-Meta-Both	22.5±0.6	19.2±0.3	28.9±0.4	28.8±0.4

Table 4.1: Misclassification rate (in %) for different meta-gradient heuristics with 5% perturbed edges.

We denote by $\tilde{\theta}_t$ the parameters at time t *independent* of the data A (and $\tilde{\theta}_{t-1}$), i.e., $\nabla_A \tilde{\theta}_t = 0$; the gradient is thus not propagated through $\tilde{\theta}_t$. This corresponds to taking the gradient of the attack loss $\mathcal{L}_{\text{atk}}$ w.r.t. the data, after training the model for T steps. We compare against this baseline in our experiments; as also done in [273]. However, unlike the meta-gradient, this approximation completely disregards the training dynamics.

[171] propose a heuristic of the meta gradient in which they update the initial weights θ_0 on a line towards the local optimum θ_T to achieve faster convergence in a multi-task learning setting: $\nabla_{\theta_0}^{\text{meta}} \approx \sum_{t=1}^{T} \nabla_{\tilde{\theta}_t} \mathcal{L}_{\text{train}}(f_{\tilde{\theta}_t}(A; X))$. Again, they assume $\tilde{\theta}_t$ to be independent of $\tilde{\theta}_{t-1}$. While there is no direct connection to the formulation of the meta gradient in Eq. (4.5), there is an intuition behind it: the heuristic meta gradient is the direction, in which, *on average*, we have observed the strongest increase in the training loss during the training procedure. The authors' experimental evaluation further indicates that this heuristic achieves similar results as the meta gradient while being much more efficient to compute (see Sec. 4.4.4 for a discussion on complexity).

When we adapt this to our adversarial attack setting on graphs, we obtain $\nabla_A^{\text{meta}} \approx \sum_{t=1}^{T} \nabla_A \mathcal{L}_{\text{train}}(f_{\tilde{\theta}_t}(A; X))$. We can view this as a heuristic of the meta gradient when $\mathcal{L}_{\text{atk}} = -\mathcal{L}_{\text{train}}$. Likewise, again taking the transductive learning setting into account, we can use self-learning to estimate the loss on the unlabeled nodes, replacing $\mathcal{L}_{\text{train}}$ by $\mathcal{L}_{\text{self}}$. Indeed, we combine these two views

$$\nabla_A^{\text{meta}} \approx \Sigma_{t=1}^{T} \lambda \nabla_A \mathcal{L}_{\text{train}}(f_{\tilde{\theta}_t}(A; X)) + (1-\lambda) \nabla_A \mathcal{L}_{\text{self}}(f_{\tilde{\theta}_t}(A; X)), \tag{4.10}$$

where λ can be used to weight the two objectives. This approximation has a much smaller memory footprint than the exact meta gradient since we don't have to store the whole training trajectory $\tilde{\theta}_1, \ldots, \tilde{\theta}_T$ in memory; additionally, there there are no second-order derivatives to be computed. A summary of our algorithm can be found in Alg. 3.

4.4.4 Complexity analysis

In our attack we handle both edge insertions and deletions, i.e. each element in the adjacency matrix $A \in \{0,1\}^{N \times N}$ can be changed. This means that without further optimization, the (approximate) meta gradient for each node pair has to be computed, leading to a baseline memory and computational complexity of $O(N^2)$. For the meta gradient computation we additionally have to store the entire weight trajectory during training, adding $O(T \cdot |\theta|)$ to the memory cost, where T is the number of inner training

	Cora-ML			Citeseer			Polblogs			Avg.
Attack	**GCN**	**CLN**	**DeepWalk**	**GCN**	**CLN**	**DeepWalk**	**GCN**	**CLN**	**DeepWalk**	**rank**
Clean	16.6±0.3	17.3±0.3	20.3±1.0	28.5±0.9	28.3±0.9	34.8±1.4	6.4±0.6	7.6±0.5	5.3±0.5	7.4
DICE	18.0±0.4	18.0±0.2	22.8±0.3	28.9±0.3	29.1±0.3	39.1±0.4	11.2±1.1	11.2±0.8	10.2±0.6	5.0
First-order	17.2±0.3	17.6±0.2	20.7±0.2	28.3±0.3	28.4±0.3	34.0±0.3	7.8±0.9	7.6±0.5	7.9±0.6	7.1
Nettack*	-	-	-	31.9±0.3	30.2±0.4	**41.2±0.4**	-	-	-	-
A-Meta-Train	21.8±0.9	20.5±0.3	25.0±0.6	31.9±0.7	30.1±0.5	32.7±0.5	11.9±2.8	12.9±2.5	5.8±0.2	4.7
A-Meta-Both	20.7±0.4	19.0±0.3	**28.5±0.5**	28.6±0.4	28.7±0.4	34.4±0.4	19.8±0.8	16.5±1.3	21.5±1.9	4.3
Meta-Train	22.0±1.2	**21.7±0.4**	26.1±0.6	30.3±1.0	29.0±0.6	36.0±0.2	16.3±2.9	**18.7±2.3**	14.5±4.2	3.2
Meta-Self	**24.5±1.0**	20.3±0.4	28.1±0.6	**34.6±0.7**	**32.2±0.6**	34.6±0.7	**22.5±0.8**	17.9±1.7	**59.0±3.0**	**2.3**
Meta w/ Oracle	21.0±0.5	21.6±0.3	27.8±0.7	34.2±0.9	32.9±0.6	36.1±0.7	25.6±1.9	19.1±1.4	52.3±2.8	2.0

* Did not finish within three days on Cora-ML and PolBlogs

Table 4.2: Misclassification rate (in %) with 5% perturbed edges.

steps and $|\theta|$ the number of weights. Thus, memory complexity of our meta gradient attack is $O(N^2 + T \cdot |\theta|)$. The second-order derivatives at each step T in the meta gradient formulation can be computed in $O(N^2)$ using Hessian-vector products, leading to a computational complexity of $O(T \cdot N^2)$.

For the meta gradient heuristics, the computational complexity is similar since we have to evaluate the gradient w.r.t. the adjacency matrix at every training step. However, the training trajectory of the weights does not have to be kept in memory, yielding a memory complexity of $O(N^2)$. This is highly beneficial, as memory (especially on GPUs) is limited.

The computational and memory complexity of our adversarial attacks implies that (as-is) it can be executed for graphs with roughly $20K$ nodes using a commodity GPU. The complexity, however, can be drastically reduced by pre-filtering the elements in the adjacency matrix for which the (meta) gradient needs to be computed, since only a fraction of entries in the adjacency matrix are promising candidate perturbations. We leave such performance optimization for future work.

4.5 Experiments

Setup. We evaluate our approach on the well-known Citeseer [201], Cora-ML [152], and PolBlogs [1] datasets; an overview is given in Table A.1. We split the datasets into labeled (10%) and unlabeled (90%) nodes. The labels of the unlabeled nodes are never visible to the attacker or the classifiers and are only used to evaluate the generalization performance of the models.

We evaluate the transferability of adversarial attacks by training deep node classification models on the modified (poisoned) data. For this purpose, we use Graph Convolutional Networks (GCN) [117] and Column Networks (CLN) [184]. Both are models utilizing the message passing framework (a.k.a. graph convolution) and trained in a semi-supervised way. We further evaluate the node classification performance achieved by training a standard logistic regression model on the node embeddings learned by DeepWalk [182]. DeepWalk itself is trained in an unsupervised way and without node

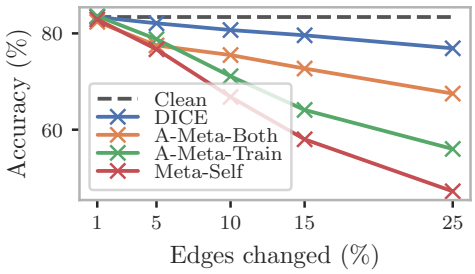


Figure 4.1: Change in accuracy of GCN on CORA-ML for increasing number of perturbations.

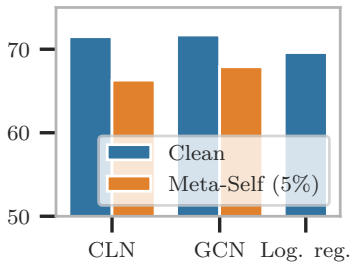


Figure 4.2: Comparison with logistic regression baseline on CITESEER.

attributes or graph convolutions; thus, this is arguably an even more difficult transfer task.

We repeat all of our attacks on five different splits of labeled/unlabeled nodes and train all target classifiers ten times per attack (using the split that was used to create the attack). In our tables, the uncertainty indicates 95 % confidence intervals of the mean obtained via bootstrapping. For our meta-gradient approaches, we compute the meta-gradient $\nabla_A^{\text{meta}} \mathcal{L}_{\text{atk}}(f_{\theta_T}(A; X))$ by using gradient descent with momentum for 100 iterations. We refer to our meta-gradient approach with self-training as Meta-Self and to the variant without self-training as Meta-Train. Similarly, we refer to our approximations as A-Meta-Self (with λ=0), A-Meta-Train (λ=1), and A-Meta-Both (λ=0.5).

Comparing meta-gradient heuristics. First, we analyze the different meta gradient heuristics described in Section 4.4.3. The results can be seen in Table 4.1. All principles successfully increase the misclassification rate (i.e., 1 − accuracy on unlabeled nodes) obtained on the test data, compared to the results obtained with the unperturbed graph. Since A-Meta-Self consistently shows a weaker performance than A-Meta-Both, we do not further consider A-Meta-Self in the following.

Comparison with competing methods. We compare our meta-gradient approach as well as its approximations with various baselines and Nettack [273]. DICE [35] ('delete internally, connect externally') is a baseline where, for each perturbation, we randomly choose whether to insert or remove an edge. Edges are only removed between nodes from the same class, and only inserted between nodes from different classes. This baseline has *all* true class labels (train and test) available and thus more knowledge than all competing methods. First-order refers to the approximation proposed by [77], i.e., ignoring all second-order derivatives. Note that Nettack is *not* designed for global attacks. In order to be able to compare to them, for each perturbation we randomly select one target node from the unlabeled nodes and attack it using Nettack while considering *all* nodes in the network. In this case, its time and memory complexity is $O(N^3)$ and thus it was not feasible to run it on any but the sparsest dataset. Meta w/ Oracle corresponds to our

	Cora-ML		Citeseer	
	GCN	**CLN**	**GCN**	**CLN**
Clean	16.6±0.3	17.3±0.3	28.5±0.8	28.3±0.8
A-Meta-Sub	21.4±0.7	22.4±0.4	30.9±0.7	31.4±0.7
Meta-Sub	21.2±0.6	20.8±0.3	28.7±0.3	31.4±0.5

Figure 4.3: Poisoning results with limited knowledge about the graph (i.e., on a subgraph) after 10% changes.

	Cora			Citeseer			PolBlogs			Avg.
Attack	**GCN**	**CLN**	**DeepWalk**	**GCN**	**CLN**	**DeepWalk**	**GCN**	**CLN**	**DeepWalk**	**rank**
Clean	16.6±0.3	17.3±0.3	20.3±0.9	28.5±0.8	28.3±0.8	34.8±1.3	6.4±0.5	7.6±0.5	5.3±0.5	3.0
A-Meta-Both	21.6±0.6	18.9±0.3	27.8±0.2	31.6±0.4	30.3±0.6	40.7±0.4	17.8±1.9	13.9±1.4	11.0±0.5	1.8
Meta-Self	29.7±2.2	20.1±0.4	31.5±1.2	29.9±0.7	32.7±0.8	45.6±0.7	17.4±0.8	14.6±1.2	16.8±1.9	1.2

Table 4.3: Misclassification rate (in %) with 5% perturbed edges and only training the surrogate model for $T = 10$ iterations to obtain the (meta-) gradients.

meta-gradient approach when supplied with *all* true class labels on the test data – this only serves as a reference point since it cannot be carried out in real scenarios where the test nodes' labels are unknown. For all methods, we enforce the *unnoticeability* constraint introduced by [273], which ensures that the graph's degree distribution changes only slightly.

In Table 4.2 we see the misclassification rates (i.e., 1 - accuracy on unlabeled nodes) achieved by changing 5% of E_{LCC} edges according to the different methods (larger is better, except for the average rank). That is, each method is allowed to modify 5% of E_{LCC}, i.e., the number of edges present in the graph before the attack. We present similar tables for 1% and 10% changes in Appendix C. Our meta-gradient with self-training (Meta-Self) produces the strongest drop in performance across all models and datasets as indicated by the *average rank*. Changing only 5% of the edges leads to a relative increase of up to 48% in the misclassification rate of GCN on Cora-ML.

Remarkably, our memory efficient meta-gradient approximations lead to strong increases in misclassifications as well. They outperform both baselines and are in many cases even on par with the more expensive meta-gradient. In Table 4.3 we can see that using only $T = 10$ training iterations of the surrogate models for computing the meta gradient (or its approximations) can significantly hurt the performance across models and datasets. Moreover, in Table C.1 in Appendix C we show that our heuristic is successful at attacking PubMed, a dataset with roughly 20K nodes.

In Fig. 4.1 we see the drop in classification performance of GCN on Cora-ML for increasing numbers of edge insertions/deletions (similar plots for the remaining datasets and models are provided in Appendix C). Meta-Self is even able to reduce the classification accuracy below 50%. Fig. 4.2 shows the classification accuracy of GCN and CLN as well as a baseline operating on the node attributes only, i.e., ignoring the graph. Not surprisingly, deep models achieve higher accuracy than the baseline when trained on the clean Citeseer graph – exploiting the network information improves classification.

	$\boldsymbol{W}$	$\tilde{\boldsymbol{W}}$
$\boldsymbol{A}$	0.85	0.52
$\tilde{\boldsymbol{A}}$	0.83	0.49

Table 4.4: Accuracy of clean/ corrupted graph and weights.

	$c_i{=}c_j$	$c_i{\neq}c_j$
DEL	15.3	3.9
INS	9.4	71.4

Table 4.5: Share (%) of edge deletions (DEL) and insertions (INS) by Meta-Self.

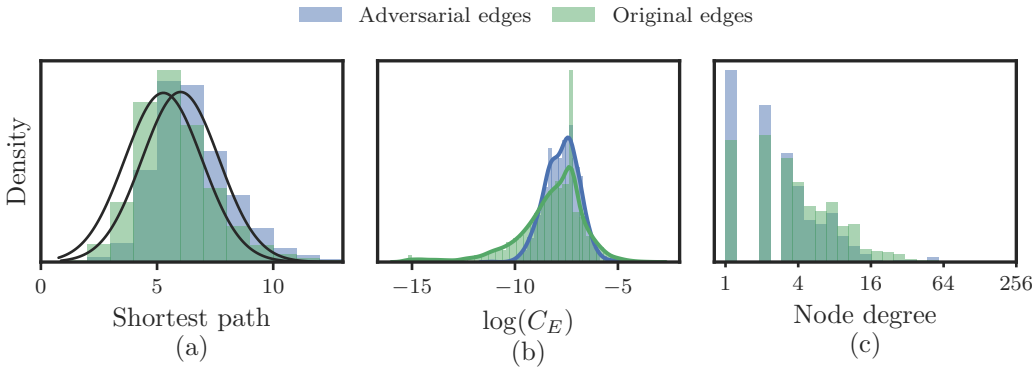


Figure 4.4: Analysis of adversarially inserted edges.

However, by only perturbing 5% of the edges, we obtain the opposite: GCN and CLN perform *worse* than the baseline – the graph structure now hurts classification.

Impact of graph structure and trained weights. Another interesting property of our attacks can be seen in Table 4.4, where $\boldsymbol{W}$ and $\tilde{\boldsymbol{W}}$ correspond to the weights trained on the clean Cora-ML network $\boldsymbol{A}$ and a version $\tilde{\boldsymbol{A}}$ poisoned by our algorithm (here with even 25 % modified edges), respectively. Note that the classification accuracy barely changes when modifying the underlying network for a given set of trained weights; even when applying the clean weights $\boldsymbol{W}$ on the highly corrupted $\tilde{\boldsymbol{A}}$, the performance drops only marginally. Likewise, even the *clean* graph $\boldsymbol{A}$ only leads to a low accuracy when using it with the weights $\tilde{\boldsymbol{W}}$. This result emphasizes the importance of the training procedure for the performance of graph models and shows that our poisoning attack works by *derailing* the training procedure from the start, i.e. leading to 'bad' weights.

Analysis of attacks. An interesting question to ask is *why* the adversarial changes created by our meta-gradient approach are so destructive, and what patterns they follow. If we can find out what makes an edge insertion or deletion a strong adversarial change, we can circumvent expensive meta-gradient computations or even use this knowledge to detect adversarial attacks.

In Fig. 4.4 we compare edges inserted by our meta-gradient approach to the edges originally present in the Cora-ML network. Fig. 4.4 (a) shows the shortest path lengths between nodes pairs *before* being connected by adversarially inserted edges vs. shortest path lengths between all nodes in the original graph. In Fig. 4.4 (b) we compare the

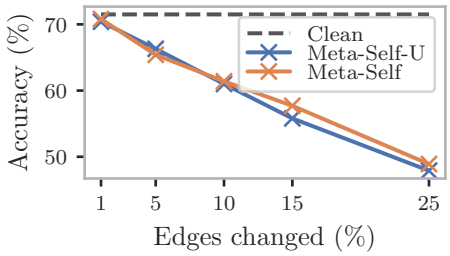


Figure 4.5: Change in accuracy of GCN on CITESEER with and without enforcing unnoticeability constraints ('Meta-Self-U').

edge betweenness centrality (C_E) of adversarially inserted edges to the centrality of edges present in the original graph. In (c) we see the node degree distributions of the original graph and the node degrees of the nodes that are picked for adversarial edges. For all three measures no clear distinction can be made. There is a slight tendency for the algorithm to connect nodes that have higher-than-average shortest paths and low degrees, though.

As we can see in Table 4.5, roughly 80% of our meta attack's perturbations are edge insertions (INS). As expected by the homophily assumption, in most cases edges inserted connect nodes from different classes and edges deleted connect same-class nodes. However, as the comparison with the DICE baseline shows, this by itself can also not explain the destructive performance of the meta-gradient.

Unnoticeability constraint. In all our experiments, we enforce the unnoticeability constraint on the degree distribution proposed by Zügner et al. [273]. Singleton nodes are never admissible, even when not enforcing unnoticeability. In Fig. 4.5 we show that this constraint does not significantly limit the destructive performance of our attacks. Thus we conclude that these constraints should always be enforced, since they improve unnoticeability while at the same time our attacks remain effective.

Attacks with changes to the node features. While the focus of our work is poisoning attacks by modifying the graph structure, our method can be applied to node feature attacks as well. The most straightforward case is when the node features are binary, since then we can use the same greedy algorithm as for the graph structure (ignoring the degree distribution constraints). Among the datasets we evaluated, CITESEER has binary node features, hence in Table 4.6 we display the results when attacking both node features and graph structure (while the total number of perturbations stays the same). We can observe that the impact of the combined attacks is slightly lower than the structure-only attack. We attribute this to the fact that we assign the same 'cost' to structure and feature changes, but arguably we expect a structure perturbation to have a stronger effect on performance than a feature perturbation. Future work can provide a framework where structure and feature changes impose a different 'cost' on the attacker. When the node features are continuous, there also needs to be some tuning of the meta step size and considerations whether multiple features per instance can be changed in a single step.

	Citeseer	
	GCN	**CLN**
Clean	28.5±0.9	28.3±0.8
Meta-Self with features	37.2±1.1	34.2±0.7
Meta-Self	38.6±1.0	35.3±0.7

Table 4.6: Misclassification rate (in %) with 10% perturbations in edges / node features. For Meta-Self with features, at each step the perturbation (edge or feature) is selected that has the highest meta gradient score.

Limited knowledge about the graph structure. In the experiments described above, the attacker has full knowledge about the graph structure and all node attributes (as typical in a transductive setting). We also tested our algorithm on a *sub-graph* of CORA-ML and CITESEER. That is, we select the 10% labeled nodes and randomly select neighbors of these until we have a subgraph with number of nodes $n = 0.3N$. We run our attacks on this small subgraph, and afterwards plug in the perturbations into the original graphs to train GCN and CLN as before. Table 4.3 summarizes the results: Even in this highly restricted setting, our attacks consistently increase misclassification rate across datasets and models, highlighting the effectiveness of our method.

4.6 Conclusion

We propose an algorithm for training-time adversarial attacks on (attributed) graphs, focusing on the task of node classification. We use meta-gradients to solve the bilevel optimization problem underlying the challenging class of poisoning adversarial attacks. Our experiments show that attacks created using our meta-gradient approach consistently lead to a strong decrease in classification performance of graph convolutional models and even transfer to unsupervised models. Remarkably, even small perturbations to a graph based on our approach can lead to graph neural networks performing worse than a baseline ignoring all relational information. We further propose approximations of the meta-gradients that are less expensive to compute and, in many cases, have a similarly destructive impact on the training of node classification models.

Retrospective

This study presented above complements the one from Chapter 3 (i) by explicitly addressing the bilevel nature of poisoning attacks and (ii) by focusing on global attacks on the overall performance of a GNN. Since we model the adjacency matrix as a dense $N \times N$ matrix, scalability is a main limitation of the approach presented in this chapter. Geisler et al. [81] present scalable attacks based on randomized block coordinate descent; combining their attack with the one presented above could be an interesting direction for future work. Moreover, the attack loss we use in this chapter is cross entropy; Geisler et al. [81] show that this is a suboptimal choice for global attacks since the attacker tends

to spend part of the budget on further attacking nodes which are already misclassified. In this chapter we highlight some intriguing properties of our poisoning attacks. First, the attacks have almost no effect on a model which was trained on a non-perturbed graph. Second, a relatively small amount of perturbations is required to lead to worse performance of a GNN than a linear structure-agnostic classifier. In [81], the authors present a similar finding on large graphs where only around 2% of perturbed edges suffice to reduce the GNN performance below an MLP on the node attributes. Later studies [e.g., 260, 249, 111] proposed defenses against the attacks presented above, which effectively aim to reverse the adversarial perturbations by filtering out edges between nodes with very dissimilar attributes.

5 Adversarial Attacks on GNNs: Retrospective

The study presented in Chapter 3 was the first to investigate the adversarial robustness properties of GNNs. It effectively kicked off a new research field, which presents us with the unique opportunity to provide an overview on how this field developed in this retrospective section. With the sheer amount and rate of new publications this section is not intended to be exhaustive, but aims to show broader trends that have been happening in the field. For more comprehensive surveys, see [209], [110], or the very recent book chapter by Günnemann [94]. The GNN robustness benchmark by Zheng et al. [266] is a recent contribution aiming to improve reproducibility and comparability of attacks and defenses on GNNs.

First and foremost, a variety of attack methods for ML on graphs have been proposed. They differ not only in terms of the methodology used but also in which models and tasks are tageted, their scalability, whether they are local or global attacks, and whether they target the training phase (poisoning) or the inference phase (evasion).

Xu et al. [245] propose poisoning and evasion attacks based on projected gradient descent. In [244], the authors propose a black-box attack method based on the graph spectrum which does not need access to the model parameters or node labels. Dai et al. [56]'s attack method is based on reinforcement learning and can be used to attack graph as well as node classification models. In the case of node classification attacks, their method is limited to remove edges from the graph. Recently, there have been fewer works on GNN attacks on node classification without additional focus, e.g., on scalability or specific application domains. Many recent studies also consider attacks in different settings and tasks such as node embeddings, graph classification, or link prediction. Moreover, there as a challenge on GNN robustness on a large real-world graph at KDD 2020 KDD Cup 2020[1]. This reflects a general shift towards more diverse and realistic attack scenarios in recent years.

Scalability. Scalability has been a major limitation of many GNN attacks, especially global ones. This is because we typically need to compute some score (e.g., based on (meta-) gradients) w.r.t. all $N \times N$ entries of the adjacency matrix, which is typically infeasbile for graphs with roughly $N > 20,000$ on current GPUs. This is much smaller than most real-world graphs; most graphs in the recently published Open Graph Benchmark (OGB) [106], which go up to 100M nodes. At the same time, scalable GNN attacks are rare, and thus our knowledge of GNNs' robustness behavior on large graphs is limited.

[1] `https://www.biendata.xyz/competition/kddcup_2020_formal/`

Further research in this direction would be important to improve our understanding and guide researchers to develop more robust methods. In the following, we mention recent scalable GNN attack methods.

Geisler et al. [81] propose a scalable global attack for GNNs based on projected block coordinate descent, which is loosely based on [245]'s projected gradient descent attack. This means that they only compute the gradient w.r.t. a small subset of adjacency matrix entries; randomization of the selected entries ensures exploration and diversity of the attacks. They show that GNNs on large graph datasets are remarkably nonrobust to global attacks. While white-box attacks are useful to find out about the *worst-case* behavior of a model, complete knowledge of the graph structure and all node features is not very realistic. An open question for follow-up work is how to design reasonable limitations of an attacker's knowledge on large-scale datasets.

Feng et al. [75] partition the graph into subgraphs and optimize the attack via ADMM. However, their surrogate model is only a single-layer linearized GCN, which limits the expressivity of the attacks; further, their attack method is still $O(N^2)$.

Li et al. [136] propose a scalable local attack on the SGC [236] model, where they use a heuristic to select only a small subset of nodes for the attack generation; this reduces memory and computation times. Similarly, Wang et al. [230] propose a scalable local attack which works by inserting adversarial nodes into the existing graph and connecting them to the existing nodes. The existing nodes' edges and features are not changed.

Tasks. After initial studies of GNN robustness on the task of node classification, researchers also explored different graph ML tasks and specific application domains. These include **(i)** node representation learning; **(ii)** graph classification; and, very recently **(iii)** neural solvers for combinatorial optimization.

For **(i)** node representation learning, Bojchevski and Günnemann [23] develop a poisoning attack on the DeepWalk [182] node embedding method, which exploits the graph spectrum and leads to reduced usefulness of the learned embeddings for downstream tasks such as node classification. Similarly, Sun et al. [210]'s attack targets DeepWalk embeddings, whereas their attack is gradient-based. Chang et al. [40]'s attack also takes into account node feature information. A typical downstream task of node representation models is link prediction. Chen et al. [44] specifically target this task by attacking a graph autoencoder (GAE) [116] based on gradient information w.r.t. the adjacency matrix. Lin et al. [141] propose a greedy attack on the SEAL GNN model for link prediction [259]. One particular application of link prediction are knowledge graphs. Zhang et al. [258] propose an adversarial attack to increase or decrease certain facts' predicted plausibility scores.

(ii) Graph classification is another typical task in ML for graphs. Here, instead of predicting properties of individual nodes or edges, we aim to classify graphs as a whole. Dai et al. [56] propse a reinforcement-learning-based attack on graph classification models. Tang et al. [213]'s method attacks graph pooling models for graph prediction via a surrogate model. Zhang et al. [262] propose backdoor attacks on graph classification models, where goal is to have graphs classified a certain way when *trigger* subgraphs

are added. At the same time the test accuracy of the model should not be affected. In their recent study, Wan et al. [226] attack graph classification models using Bayesian optimization and genetic algorithms.

Very recently, Geisler et al. [82] propose adversarial attacks on **(iii)** neural solvers for combinatorial optimization [e.g., 200, 185], which are often based on GNN models. The authors propose attacks for the SATisfiability (SAT) and traveling salesperson (TSP) problems. This highlights the relevance and applicability of GNN robustness study to diverse models, applications, and domains.

Another interesting case is the study of Hsieh and Li [105]'s method called NetFense. The method is effectively an *adversarial attack* method based on Nettack whose goal is to *defend* the GNN against privacy attacks. That is, they perturb the graph structure in such a way that predicting certain private attributes becomes harder. They achieve this by making changes to the input graph in order to make the output of a classifier predicting the sensitive attribute as close as possible to a 50/50 prediction. At the same time, their method aims to maintain the performance of the GNN in predicting the class of the node.

Part III

Robustness Certification of Graph Neural Networks

6 Certifiable Robustness and Robust Training for Graph Convolutional Networks

6.1 Introduction

As shown in Part II, graph neural network models for node classification are vulnerable to adversarial attacks: Even only slight deliberate perturbations of the nodes' features or the graph structure can lead to completely wrong predictions. Such negative results significantly hinder the applicability of these models. The results become unreliable and such problems open the door for attackers that can exploit these vulnerabilities.

So far, no effective mechanisms are available, which (i) prevent that small changes to the data lead to completely different predictions in a GNN, or (ii) that can verify whether a given GNN is robust w.r.t. a specific perturbation model. This is critical, since especially in domains where graph-based learning is used (e.g. the Web) adversaries are omnipresent, e.g., manipulating online reviews and product websites [104]. One of the core challenges is that in a GNN a node's prediction is also affected when perturbing *other* nodes in the graph – making the space of possible perturbations large. How to make sure that small changes to the input data do not have a dramatic effect to a GNN?

In this chapter, we shed light on this problem by proposing the first method for *provable robustness* of GNNs. More precisely, we focus on graph convolutional networks and potential perturbations of the node attributes, where we provide:

1) *Certificates:* Given a trained GNN, we can give robustness certificates that state that a node is robust w.r.t. a certain space of perturbations. If the certificate holds, it is guaranteed that *no* perturbation (in the considered space) exists which will change the node's prediction. Furthermore, we also provide non-robustness certificates that, when they hold, state whether a node is not robust; realized by providing an adversarial example.

2) *Robust Training:* We propose a learning principle that improves the robustness of the GNN (i.e., making it less sensitive to perturbations) while still ensuring high accuracy for node classification. Specifically, we exploit the semi-supervised nature of the GNN learning task, thus, taking also the unlabeled nodes into account.

In contrast to existing works on provable robustness for classical neural networks/robust training (e.g. [234, 189, 101]), we tackle various additional challenges: Being the

first approach for graphs, we have to deal with perturbations of multiple instances simultaneously. For this, we introduce a novel space of perturbations where the perturbation budget is constrained locally and globally. Moreover, since the considered data domains are often discrete/binary attributes, we tackle challenging L_0 constraints on the perturbations. Lastly, we exploit a crucial aspect of semi-supervised learning by taking also the unlabeled nodes into account for robust training.

The key idea we will exploit in our approach is to estimate the *worst-case* change in the predictions obtained by the GNN under the space of perturbations. If the worst possible change is small, the GNN is robust. Since, however, this worst-case cannot be computed efficiently, we provide *bounds* on this value, providing conservative estimates. More technically, we derive relaxations of the GNN and the perturbations space, enabling efficient computation.

Besides the two core technical contributions mentioned above, we further perform extensive experiments:

3) *Experiments:* We show on various graph datasets that GNNs trained in the traditional way are not robust, i.e. only few of the nodes can be certified to be robust, respectively many are certifiably non-robust even with small perturbation budgets. In contrast, using our robust training we can dramatically improve robustness increasing it by in some cases by factor of four.

Overall, using our method, significantly improves the reliability of GNNs, thus, being highly beneficial when, e.g., using them in real production systems or scientific applications.

6.2 Related Work

The sensitivity of machine learning models w.r.t. adversarial perturbations has been studied extensively [88] . Only recently, however, researchers have started to investigate adversarial attacks on graph neural networks [273, 56, 276] and node embeddings [23]. All of these works focus on generating adversarial examples. In contrast, we provide the first work to certify and improve the robustness of GNNs. As shown in [273], both perturbations to the node attributes as well as the graph structure are harmful. In this chapter, we focus on perturbations of the node attributes; structure perturbations are studied in Chapter 7.

For 'classical' neural networks various heuristic approaches have been proposed to improve the the robustness to adversarial examples [180]. However, such heuristics are often broken by new attack methods, leading to an arms race. As an alternative, recent works have considered certifiable robustness [234, 189, 101, 55] providing guarantees that no perturbation w.r.t. a specific perturbation space will change an instance's prediction.

For this chapter, specifically the class of methods based on convex relaxations are of relevance [234, 189]. They construct a convex relaxation for computing a lower bound on the worst-case margin achievable over all possible perturbations. This bound serves as a certificate of robustness. Solving such convex optimization problems can often been

done efficiently, and by exploiting duality it enables to even train a robust model [234]. As already mentioned, our work differs significantly from the existing methods since (i) it considers the novel GNN domain with its relational dependencies, (ii) it handles a discrete/binary data domain, while existing works have only handled continuous data; thus also leading to very different constraints on the perturbations, and (iii) we propose a novel robust training procedure which specifically exploits the semi-supervised learning setting of GNNs, i.e. using the unlabeled nodes as well.

6.3 Preliminaries

We consider the task of (semi-supervised) node classification in a single large graph having binary node features. Let $G = (\boldsymbol{A}, \boldsymbol{X})$ be an attributed graph, where $\boldsymbol{A} \in \{0,1\}^{N \times N}$ is the adjacency matrix and $\boldsymbol{X} \in \{0,1\}^{N \times D}$ represents the nodes' features. W.l.o.g. we assume the node-ids to be $\mathcal{V} = \{1, \ldots, N\}$. Given a subset $\mathcal{V}_L \subseteq \mathcal{V}$ of labeled nodes, with class labels from $\mathcal{C} = \{1, 2, \ldots, K\}$, the goal of node classification is to learn a function $f : \mathcal{V} \to \mathcal{C}$ which maps each node $v \in \mathcal{V}$ to one class in $\mathcal{C}$. In this chapter, we focus on node classification employing graph neural networks. In particular, we consider graph convolutional networks where the latent representations $\boldsymbol{H}^{(l)}$ at layer l are of the form

$$\boldsymbol{H}^{(l)} = \sigma^{(l)}\left(\hat{\boldsymbol{A}}^{(l-1)}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l-1)} + \boldsymbol{b}^{(l-1)}\right) \quad \text{for } l = 2, ..., L \tag{6.1}$$

where $\boldsymbol{H}^{(1)} = \boldsymbol{X}$ and with activation functions given by

$$\sigma^{(L)}(\cdot) = \text{softmax}(\cdot), \qquad \sigma^{(l)}(\cdot) = \text{ReLU}(\cdot) \qquad \text{for } l = 2, ..., L-1.$$

The output $\boldsymbol{H}^{(L)}_{vc}$ denotes the probability of assigning node v to class c. The $\hat{\boldsymbol{A}}^{(l)}$ are the message passing matrices that define how the activations are propagated in the network. In GCN [117], for example, $\hat{\boldsymbol{A}}^{(1)} = ... = \hat{\boldsymbol{A}}^{(L-1)} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$, where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_{N \times N}$ and $\tilde{\boldsymbol{D}}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. The $\boldsymbol{W}^{(\cdot)}$ and $\boldsymbol{b}^{(\cdot)}$ are the trainable weights of the graph neural network, usually simply learned by minimizing the cross-entropy loss on the given labeled training nodes $\mathcal{V}_L$.

Notations: We denote with $\mathcal{N}_l(t)$ the l-hop neighborhood of a node t, i.e. all nodes which are reachable with l hops (or less) from node t, including the node t itself. Given a matrix $\boldsymbol{X}$, we denote its positive part with $[\boldsymbol{X}]_+ = \max(\boldsymbol{X}, 0)$ where the max is applied entry-wise. Similarly, the negative part is $[\boldsymbol{X}]_- = -\min(\boldsymbol{X}, 0)$, which are non-negative numbers. All matrix norms $||\boldsymbol{X}||_p$ used in this chapter are meant to be entry-wise, i.e. flattening $\boldsymbol{X}$ to a vector and applying the corresponding vector norm. We denote with $h^{(l)}$ the dimensionality of the latent space in layer l, i.e. $\boldsymbol{H}^{(l)} \in \mathbb{R}^{N \times h^{(l)}}$. $\boldsymbol{X}_{i:}$ denotes the i-th row of a matrix $\boldsymbol{X}$ and $\boldsymbol{X}_{:j}$ its j-th column.

6.4 Certifying Robustness for Graph Convolutional Networks

Our first goal is to derive an efficient principle for robustness certificates. That is, given an already trained GNN and a specific node t under consideration (called *target* node),

our goal is to provide a certificate which guarantees that the prediction made for node t *will not change* even if the data gets perturbed (given a specific perturbation budget). That is, if the certificate is provided, the prediction for this node is robust under *any* admissible perturbations. Unlike existing works, we cannot restrict perturbations to the instance itself due to the relational dependencies.

However, we can exploit one key insight: for a GNN with L layers, the output $\boldsymbol{H}_{t:}^{(L)}$ of node t depends only on the nodes in its $L-1$ hop neighborhood $\mathcal{N}_{L-1}(t)$. Therefore, instead of operating with Eq. (6.1), we can 'slice' the matrices $\boldsymbol{X}$ and $\hat{\boldsymbol{A}}^{(l)}$ at each step to only contain the entries that are required to compute the output for the target node t.[1] This step drastically improves scalability – reducing not only the size of the neural network but also the potential perturbations we have to consider later on. We define the matrix slices for a given target t as follows:[2]

$$\dot{\boldsymbol{A}}^{(l)} = \hat{\boldsymbol{A}}^{(l)}_{\mathcal{N}_{L-l}(t),\mathcal{N}_{L-l+1}(t)} \text{ for } l = 1, ..., L-1, \qquad \dot{\boldsymbol{X}} = \boldsymbol{X}_{\mathcal{N}_{L-1}(t):} \tag{6.2}$$

where the set indexing corresponds to slicing the rows and columns of a matrix, i.e., $\hat{\boldsymbol{A}}_{\mathcal{N}_2(t),\mathcal{N}_1(t)}$ contains the rows corresponding to the two-hop neighbors of node t and the columns corresponding to its one-hop neighbors. As it becomes clear, for increasing l (i.e., depth in the network), the slices of $\hat{\boldsymbol{A}}^{(l)}$ become smaller, and at the final step we only need the target node's one-hop neighbors.

Overall, we only need to consider the following *sliced* GNN:

$$\hat{\boldsymbol{H}}^{(l)} = \dot{\boldsymbol{A}}^{(l-1)} \boldsymbol{H}^{(l-1)} \boldsymbol{W}^{(l-1)} + \boldsymbol{b}^{(l-1)} \qquad \text{for } l = 2, ..., L \tag{6.3}$$

$$\boldsymbol{H}^{(l)}_{nj} = \max\left\{\hat{\boldsymbol{H}}^{(l)}_{nj}, 0\right\} \qquad \text{for } l = 2, ..., L-1 \tag{6.4}$$

and $\boldsymbol{H}^{(1)} = \dot{\boldsymbol{X}}$. Here, we replaced the ReLU activation by its analytical form, and we denoted with $\hat{\boldsymbol{H}}^{(l)}$ the input before applying the ReLU, and with $\boldsymbol{H}^{(l)}$ the corresponding output. Note that the matrices are getting smaller in size – with $\hat{\boldsymbol{H}}^{(L)}$ actually reducing to a vector that represents the predicted log probabilities (logits) for node t only. Note that we also omitted the softmax activation function in the final layer L since for the final classification decision it is sufficient to consider the largest value of $\hat{\boldsymbol{H}}^{(L)}$. Overall, we denote the output of this sliced GNN as $f^t_\theta(\dot{\boldsymbol{X}}, \dot{\boldsymbol{A}}) = \hat{\boldsymbol{H}}^{(L)} \in \mathbb{R}^K$. Here θ is the set of all parameters, i.e., $\theta = \{\boldsymbol{W}^{(\cdot)}, \boldsymbol{b}^{(\cdot)}\}$.

6.4.1 Robustness Certificates for GNNs

Given this set-up, we are now ready to define our actual task: We aim to verify whether no admissible perturbation changes the prediction of the target node t. Formally we aim to solve:

[1] Note that the shapes of $\boldsymbol{W}$ and $\boldsymbol{b}$ do not change.

[2] To avoid clutter in the notation, since our method certifies robustness with respect to a specific node t, we omit explicitly mentioning the target node t in the following.

Problem 3. *Given a graph G, a target node t, and an GNN with parameters θ. Let y^* denote the class of node t (e.g. given by the ground truth or predicted). The worst case margin between classes y^* and y achievable under some set $\mathcal{X}_{q,Q}(\dot{\boldsymbol{X}})$ of admissible perturbations to the node attributes is given by*

$$m^t(y^*, y) := \underset{\tilde{\boldsymbol{X}}}{\text{minimize}} \; f_\theta^t(\tilde{\boldsymbol{X}}, \dot{\boldsymbol{A}})_{y^*} - f_\theta^t(\tilde{\boldsymbol{X}}, \dot{\boldsymbol{A}})_y \quad \text{subject to } \tilde{\boldsymbol{X}} \in \mathcal{X}_{q,Q}(\dot{\boldsymbol{X}}) \tag{6.5}$$

If $m^t(y^*, y) > 0$ for all $y \neq y^*$, the GNN is certifiably robust w.r.t. node t and $\mathcal{X}_{q,Q}$.

If the minimum in Eq. (6.5) is positive, it means that there exists *no* adversarial example (within our defined admissible perturbations) that leads to the classifier changing its prediction to the other class y – i.e. the logits of class y^* are always larger than the one of y.

Setting reasonable constraints to adversarial attacks is important to obtain certificates that reflect realistic attacks. Works for *classical* neural networks have constrained the adversarial examples to lie on a small ϵ-ball around the original sample measured by, e.g., the infinity-norm or L2-norm [234, 189, 55], often e.g., $\epsilon < 0.1$ This is clearly not practical in our binary setting as an $\epsilon < 1$ would mean that *no* attribute can be changed. To allow reasonable perturbations in a binary/discrete setting one has to allow much larger changes than the ϵ-balls considered so far.

Therefore, motivated by the existing works on adversarial attacks to graphs [273], we consider a more realistic scenario: We define the set of admissible perturbations by limiting the *number* of changes to the original attributes – i.e. we assume a perturbation budget $Q \in \mathbb{N}$ and measure the L_0 norm in the change to $\dot{\boldsymbol{X}}$. It is important to note that in a graph setting an adversary can attack the target node by also changing the node attributes of its $L-1$ hop neighborhood. Thus, Q acts as a *global* perturbation budget.

However, since changing many attributes for a single node might not be desired, we also allow to limit the number of perturbations *locally* – i.e. *for each node* in the $L-1$ hop neighborhood we can consider a budget of $q \in \mathbb{N}$. Overall, in this chapter we consider admissible perturbations of the form:

$$\mathcal{X}_{q,Q}(\dot{\boldsymbol{X}}) = \Big\{ \tilde{\boldsymbol{X}} \;\Big|\; \tilde{\boldsymbol{X}}_{nj} \in \{0,1\} \wedge \|\tilde{\boldsymbol{X}} - \dot{\boldsymbol{X}}\|_0 \leq Q \wedge \|\tilde{\boldsymbol{X}}_{n:} - \dot{\boldsymbol{X}}_{n:}\|_0 \leq q \; \forall n \in \mathcal{N}_{L-1} \Big\}. \tag{6.6}$$

Challenges There are two major obstacles preventing us from efficiently finding the minimum in Eq. (6.5). First, our data domain is discrete, making optimization often intractable. Second, our function (i.e., the GNN) f_θ^t is nonconvex due to the nonlinear activation functions in the neural network. But there is hope: As we will show, we can efficiently find *lower bounds* on the minimum of the original problem by performing specific relaxations of (i) the neural network, and (ii) the data domain. This means that if the lower bound is positive, we are certain that our classifier is robust w.r.t.

the set of admissible perturbations. Remarkably, we will even see that our relaxation has an optimal solution which is integral. That is, we obtain an optimal solution (i.e. perturbation) which is binary – thus, we can effectively handle the discrete data domain.

6.4.2 Convex Relaxations

To make the objective function in Eq. (6.5) convex, we have to find a convex relaxation of the ReLU activation function. While there are many ways to achieve this, we follow the approach of [234] in this chapter. The core idea is (i) to treat the matrices $\boldsymbol{H}^{(\cdot)}$ and $\hat{\boldsymbol{H}}^{(\cdot)}$ in Eqs. (6.3,6.4) no longer as deterministic but as *variables* one can optimize over (besides optimizing over $\tilde{\boldsymbol{X}}$). In this view, Eqs. (6.3,6.4) become constraints the variables have to fulfill. Then, (ii) we relax the non-linear ReLU constraint of Eq. (6.4) by a set of convex ones.

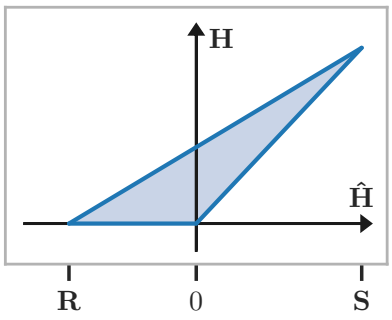


Figure 6.1: Convex relaxation of ReLU

In detail: Consider Eq. (6.4). Here, $\hat{\boldsymbol{H}}_{nj}^{(l)}$ denotes the input to the ReLU activation function. Let us assume we have given some lower bounds $\boldsymbol{R}_{nj}^{(l)}$ and upper bounds $\boldsymbol{S}_{nj}^{(l)}$ on this input based on the possible perturbations (in Section 6.4.5 we will discuss how to find these bounds). We denote with $\mathcal{I}^{(l)}$ the set of all tuples (n, j) in layer l for which the lower and upper bounds differ in their sign, i.e., $\boldsymbol{R}_{nj}^{(l)} < 0 < \boldsymbol{S}_{nj}^{(l)}$. We denote with $\mathcal{I}_+^{(l)}$ and $\mathcal{I}_-^{(l)}$ the tuples where both bounds are non-negative and non-positive, respectively.

Consider the case $\mathcal{I}^{(l)}$: We relax Eq. (6.4) using a convex envelope:

$$\begin{aligned} \boldsymbol{H}_{nj}^{(l)} \geq \hat{\boldsymbol{H}}_{nj}^{(l)}, \qquad \boldsymbol{H}_{nj}^{(l)} \geq 0, \\ \boldsymbol{H}_{nj}^{(l)} \left(\boldsymbol{S}_{nj}^{(l)} - \boldsymbol{R}_{nj}^{(l)} \right) \leq \boldsymbol{S}_{nj}^{(l)} \left(\hat{\boldsymbol{H}}_{nj}^{(l)} - \boldsymbol{R}_{nj}^{(l)} \right) \qquad \text{if } (n,j) \in \mathcal{I}^{(l)} \end{aligned}$$

The idea is illustrated in the figure on the right. Note that $\boldsymbol{H}_{nj}^{(l)}$ is no longer the deterministic output of the ReLU given its input but it is a *variable*. For a given input, the variable is constrained to lie on a vertical line above the input and below the upper line of the envelope.

Accordingly, but more simply, for the cases $\mathcal{I}_+^{(l)}$ and $\mathcal{I}_-^{(l)}$ we get:

$$\boldsymbol{H}_{nj}^{(l)} = \hat{\boldsymbol{H}}_{nj}^{(l)} \qquad \text{if } (n,j) \in \mathcal{I}_+^{(l)} \qquad \text{and} \qquad \boldsymbol{H}_{nj}^{(l)} = 0 \qquad \text{if } (n,j) \in \mathcal{I}_-^{(l)}$$

which are actually not relaxations but exact conditions. Overall, Eq. (6.4) has now been replaced by a set of linear (i.e. convex) constraints. Together with the linear constraints of Eq. (6.3) they determine the set of admissible $\boldsymbol{H}^{(\cdot)}$ and $\hat{\boldsymbol{H}}^{(\cdot)}$ we can optimize over. We denote the collection of these matrices that fulfill these constraints by $\mathcal{Z}_{q,Q}(\tilde{\boldsymbol{X}})$. Note that this set depends on $\tilde{\boldsymbol{X}}$ since $\boldsymbol{H}^{(1)} = \tilde{\boldsymbol{X}}$.

Overall, our problem becomes:

$$\tilde{m}^t(y^*, y) := \underset{\tilde{\boldsymbol{X}}, \boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}}{\text{minimize}} \; \hat{\boldsymbol{H}}^{(L)}_{y^*} - \hat{\boldsymbol{H}}^{(L)}_{y} = \boldsymbol{c}^\top \hat{\boldsymbol{H}}^{(L)} \quad \text{subject to } \tilde{\boldsymbol{X}} \in \mathcal{X}_{q,Q}(\dot{\boldsymbol{X}}) \;,\; [\boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}] \in \mathcal{Z}_{q,Q}(\tilde{\boldsymbol{X}}) \tag{6.7}$$

Here we introduced the constant vector $\boldsymbol{c} = \boldsymbol{e}_{y^*} - \boldsymbol{e}_y$, which is 1 at position y^*, -1 at y, and 0 else. This notation clearly shows that the objective function is a simple linear function.

Corollary 1. *The minimum in Eq. (6.7) is a lower bound on the minimum of the problem in Eq. (6.5), i.e.* $\tilde{m}^t(y^*, y) \leq m^t(y^*, y)$.

Proof. Let $\tilde{\boldsymbol{X}}$ be the perturbation obtained by Problem 3, and $[\boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}]$ the resulting *exact* representations based on Eq. (6.3)+(6.4). By construction, $[\boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}] \in \mathcal{Z}_{q,Q}(\dot{\boldsymbol{X}})$. Since Eq. (6.7) optimizes over the full set $\mathcal{Z}_{q,Q}(\dot{\boldsymbol{X}})$ its minimum can not be larger. □

From Corollary 1 it follows that if $\tilde{m}^t(y^*, y) > 0$ for all $y \neq y^*$, the GNN is robust at node t. Directly solving Eq. (6.7), however, is still intractable due to the discrete data domain.

As one core contribution, we will show that we can find the optimal solution in a tractable way. We proceed in two steps: (i) We first find a suitable continuous, convex relaxation of the discrete domain of possible adversarial examples. (ii) We show that the relaxed problem has an optimal solution which is integral; thus, by our specific construction the solution is binary.

More precisely, we relax the set $\mathcal{X}_{q,Q}(\dot{\boldsymbol{X}})$ to:

$$\hat{\mathcal{X}}_{q,Q}(\dot{\boldsymbol{X}}) = \Big\{ \tilde{\boldsymbol{X}} \;\Big|\; \tilde{\boldsymbol{X}}_{nj} \in [0,1] \wedge \|\tilde{\boldsymbol{X}} - \dot{\boldsymbol{X}}\|_1 \leq Q \wedge \|\tilde{\boldsymbol{X}}_{n:} - \dot{\boldsymbol{X}}_{n:}\|_1 \leq q \; \forall n \in \mathcal{N}_{L-1} \Big\} \tag{6.8}$$

Note that the entries of $\tilde{\boldsymbol{X}}$ are now continuous between 0 and 1, and we have replaced the L_0 norm with the L_1 norm. This leads to:

$$\hat{m}^t(y^*, y) := \underset{\tilde{\boldsymbol{X}}, \boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}}{\text{minimize}} \; \hat{\boldsymbol{H}}^{(L)}_{y^*} - \hat{\boldsymbol{H}}^{(L)}_{y} = \boldsymbol{c}^\top \hat{\boldsymbol{H}}^{(L)} \quad \text{subject to } \tilde{\boldsymbol{X}} \in \hat{\mathcal{X}}_{q,Q}(\dot{\boldsymbol{X}}) \;,\; [\boldsymbol{H}^{(\cdot)}, \hat{\boldsymbol{H}}^{(\cdot)}] \in \mathcal{Z}_{q,Q}(\tilde{\boldsymbol{X}}) \tag{6.9}$$

It is worth mentioning that Eq. (6.9) is a linear problem since besides the linear objective function also all constraints are linear. We provide the explicit form of this linear program in the appendix. Accordingly, Eq. (6.9) can be solved optimally in a tractable way. Since $\hat{\mathcal{X}}_{q,Q}(\dot{\boldsymbol{X}}) \supset \mathcal{X}_{q,Q}(\dot{\boldsymbol{X}})$, we trivially have $\hat{m}^t(y^*, y) \leq \tilde{m}^t(y^*, y)$. But even more, we obtain:

Theorem 3. *The minimum in Eq. (6.7) is equal to the minimum in Eq. (6.9), i.e.* $\tilde{m}^t(y^*, y) = \hat{m}^t(y^*, y)$.

We will proof this theorem later (see Sec. 6.4.4) since it requires some further results. In summary, using Theorem 3, we can indeed handle the discrete data domain/discrete perturbations exactly and tractably by simply solving Eq. (6.9) instead of Eq. (6.7).

6.4.3 Efficient Lower Bounds via the Dual

In order to provide a robustness *guarantee* w.r.t. the perturbations on $\dot{\boldsymbol{X}}$, we have to find the *minimum* of the linear program in Eq. (6.9) to ensure that we have covered the worst case. While it is possible to solve linear programs 'efficiently' using highly optimized linear program solvers, the potentially large number of variables in a GNN makes this approach rather slow. As an alternative, we can consider the dual of the linear program [234]. There, *any* dual-feasible solution is a lower bound on the minimum of the primal problem. That is, if we find *any* dual-feasible solution for which the objective function of the dual is positive, we know that the minimum of the primal problem has to be positive as well, guaranteeing robustness of the GNN w.r.t. any perturbation in the set.

Theorem 4. *The dual of Eq. (6.9) is equivalent to:*

$$\begin{aligned} \underset{\boldsymbol{\Omega},\boldsymbol{\eta},\rho}{\text{maximize}} \quad & g^t_{q,Q}\left(\dot{\boldsymbol{X}}, \boldsymbol{c}, \boldsymbol{\Omega}, \boldsymbol{\eta}, \rho\right) \\ \text{subject to} \quad & \\ & \boldsymbol{\Omega}^{(l)} \in [0,1]^{|\mathcal{N}_{L-l}| \times h^{(l)}} \text{ for } l = L-1, ..., 2, \\ & \boldsymbol{\eta} \in \mathbb{R}^{|\mathcal{N}_{L-1}|}_{\geq 0}, \quad \rho \in \mathbb{R}_{\geq 0} \end{aligned} \tag{6.10}$$

where

$$\begin{aligned} g^t_{q,Q}(...) = & \sum_{l=2}^{L-1} \sum_{(n,j)\in\mathcal{I}^{(l)}} \frac{\boldsymbol{S}^{(l)}_{nj}\boldsymbol{R}^{(l)}_{nj}}{\boldsymbol{S}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj}} \left[\hat{\boldsymbol{\Phi}}^{(l)}_{nj}\right]_+ - \sum_{l=1}^{L-1} \mathbf{1}^\top \boldsymbol{\Phi}^{(l+1)} \boldsymbol{b}^{(l)} \\ & - \operatorname{Tr}\left[\dot{\boldsymbol{X}}^\top \hat{\boldsymbol{\Phi}}^{(1)}\right] - \|\boldsymbol{\Psi}\|_1 - q \cdot \sum_n \boldsymbol{\eta}_n - Q \cdot \rho \end{aligned}$$

and

$$\begin{aligned} & \boldsymbol{\Phi}^{(L)} = -\boldsymbol{c} \in \mathbb{R}^k \\ & \hat{\boldsymbol{\Phi}}^{(l)} = \dot{\boldsymbol{A}}^{(l)\top} \boldsymbol{\Phi}^{(l+1)} \boldsymbol{W}^{(l)\top} \in \mathbb{R}^{|\mathcal{N}_{L-l}| \times h^{(l)}} \text{ for } l = L-1, ..., 1 \\ & \boldsymbol{\Phi}^{(l)}_{nj} = \begin{cases} 0 & \text{if } (n,j) \in \mathcal{I}^{(l)}_- \\ \hat{\boldsymbol{\Phi}}_{nj} & \text{if } (n,j) \in \mathcal{I}^{(l)}_+ \\ \frac{\boldsymbol{S}^{(l)}_{nj}}{\boldsymbol{S}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj}} \left[\hat{\boldsymbol{\Phi}}^{(l)}_{nj}\right]_+ - \boldsymbol{\Omega}^{(l)}_{nj} \left[\hat{\boldsymbol{\Phi}}^{(l)}_{nj}\right]_- & \text{if } (n,j) \in \mathcal{I}^{(l)} \end{cases} \\ & \qquad\qquad\qquad\qquad\qquad\qquad \text{for } l = L-1, ..., 2 \\ & \boldsymbol{\Psi}_{nd} = \max\left\{\boldsymbol{\Delta}_{nd} - (\boldsymbol{\eta}_n + \rho), 0\right\} \\ & \boldsymbol{\Delta}_{nd} = \left[\hat{\boldsymbol{\Phi}}^{(1)}_{nd}\right]_+ \cdot (1 - \dot{\boldsymbol{X}}_{nd}) + \left[\hat{\boldsymbol{\Phi}}^{(1)}_{nd}\right]_- \cdot \dot{\boldsymbol{X}}_{nd} \end{aligned}$$

The proof is given in the appendix. Note that parts of the dual problem in Theorem 4 have a similar form to the problem in [234]. For instance, we can interpret this dual problem as a *backward* pass on a GNN, where the $\hat{\mathbf{\Phi}}^{(l)}$ and $\mathbf{\Phi}^{(l)}$ are the hidden representations of the respective nodes in the graph. Crucially different, however, is the propagation in the dual problem with the message passing matrices $\dot{\boldsymbol{A}}$ coming from the GNN formulation where neighboring nodes influence each other. Furthermore, our novel perturbation constraints from Eq. (6.8) lead to the dual variables $\boldsymbol{\eta}$ and ρ, which have their origin in the local (q) and global (Q) constraints, respectively. Note that, in principle, our framework allows for different budgets q per node. The term $\mathbf{\Psi}$ has its origin in the constraint $\tilde{\boldsymbol{X}}_{nj} \in [0, 1]$.

While on the first look, the above dual problem seems rather complicated, its specific form makes it amenable for *easy optimization*. The variables $\mathbf{\Omega}, \boldsymbol{\eta}, \rho$ have only simple, element-wise constraints (e.g., clipping between $[0, 1]$). All other terms are just deterministic assignments. Thus, straightforward optimization using (projected) gradient ascent in combination with any modern automatic differentiation framework (e.g. TensorFlow, PyTorch) is possible.

Furthermore, while in the above dual we need to optimize over $\boldsymbol{\eta}$ and ρ, it turns out that we can simplify it even further: for any feasible $\mathbf{\Omega}$, we get an optimal closed-form solution for $\boldsymbol{\eta}, \rho$.

Theorem 5. *Given the dual problem from Theorem 4 and any dual-feasible value for $\mathbf{\Omega}$. For each node $n \in \mathcal{N}_{L-1}$, let S_n be the set of dimensions d corresponding to the q largest values from the vector $\boldsymbol{\Delta}_{n:}$ (ties broken arbitrarily). Further, denote with $o_n = \min_{d \in S_n} \Delta_{nd}$ the smallest of these values. The optimal ρ that maximizes the dual is the Q-th largest value from $[\Delta_{nd}]_{n \in \mathcal{N}_{L-1}, d \in S_n}$. For later use we denote with S_Q the set of tuples (n, d) corresponding to these Q-largest values. Moreover, the optimal $\boldsymbol{\eta}_n$ is $\boldsymbol{\eta}_n = \max\{0, o_n - \rho\}$.*

The proof is given in the appendix. Using Theo. 5, we obtain an even more compact dual where we only have to optimize over $\mathbf{\Omega}$. Importantly, the calculations done in Theo. 5 are also available in many modern automatic differentiation frameworks (i.e. we can back-propagate through them). Thus, we still get very efficient (and easy to implement) optimization.

Default value As mentioned before, it is not required to solve the dual problem optimally. Any dual-feasible solution leads to a lower bound on the original problem. Specifically, we can also just evaluate the function $g^t_{q,Q}$ *once* given a single instantiation for $\mathbf{\Omega}$. This makes the computation of robustness certificates extremely fast. For example, adopting the result of [234], instead of optimizing over $\mathbf{\Omega}$ we can set it to

$$\mathbf{\Omega}^{(l)}_{nj} = \boldsymbol{S}^{(l)}_{nj} \cdot (\boldsymbol{S}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj})^{-1}, \tag{6.11}$$

which is dual-feasible, and still obtain strong robustness certificates. In our experimental section, we compare the results obtained using this default value to results for optimizing over $\mathbf{\Omega}$. Note that using Theo. 5 we always ensure to use the optimal $\boldsymbol{\eta}, \rho$ w.r.t. $\mathbf{\Omega}$.

6.4.4 Primal Solutions and Certificates

Based on the above results, we can now prove the following:

Corollary 2. *Eq.* (6.9) *is an integral linear program with respect to the variables* $\tilde{\boldsymbol{X}}$.

The proof is given in the appendix. Using this result, it is now straightforward to prove Theo. 3 from the beginning.

Proof. Since Eq. (6.9) has an optimal (thus, feasible) solution where $\tilde{\boldsymbol{X}}$ is integral, we have $\tilde{\boldsymbol{X}} \in \hat{\mathcal{X}}_{q,Q}(\dot{\boldsymbol{X}})$ and, thus, $\tilde{\boldsymbol{X}}$ has to be binary to be integral. Since in this case the L_1 constraints are equivalent to the L_0 constraints, it follows that $\tilde{\boldsymbol{X}} \in \mathcal{X}_{q,Q}(\dot{\boldsymbol{X}})$. Thus, this optimal solution of Eq. 6.9 is feasible for Eq. 6.7 as well. Together with $\hat{m}^t(y^*, y) \leq \tilde{m}^t(y^*, y)$ it follows that $\hat{m}^t(y^*, y) = \tilde{m}^t(y^*, y)$. □

In the proof of Corollary 2, we have seen that in the optimal solution, the set $\{(n, d) \in S_Q \mid \boldsymbol{\Delta}_{nd} > 0\} =: P$ indicates those elements which are perturbed. That is, we constructed the worst-case perturbation. Clearly, this mechanism can also be used even if $\boldsymbol{\Omega}$ (and, thus, $\boldsymbol{\Delta}$) is not optimal: simply perturbing the elements in P. In this case, of course, the primal solution might not be optimal and we cannot use it for a robustness certificate. However, since the resulting perturbation is primal feasible (regarding the set $\mathcal{X}_{q,Q}(\dot{\boldsymbol{X}})$), we can use it for our non-robustness certificate: After constructing the perturbation $\tilde{\boldsymbol{X}}$ based on P, we pass it through the *exact* GNN, i.e. we evaluate Eq. (6.5). If the value is negative, we found a harmful perturbation, certifying non-robustness.

In summary By considering the dual program, we obtain robustness certificates if the obtained (dual) values are positive for every $y \neq y^*$. In contrast, by constructing the primal feasible perturbation using P, we obtain non-robustness certificates if the obtained (exact, primal) values are negative for one $y \neq y^*$. For some nodes, neither of these certificates can be given. We analyze this aspect in more detail in our experiments.

6.4.5 Activation Bounds

One crucial component of our method, the computation of the bounds $\boldsymbol{R}^{(l)}$ and $\boldsymbol{S}^{(l)}$ on the activations in the relaxed GNN, remains to be defined. Again, existing bounds for classical neural networks are not applicable since they neither consider L_0 constraints nor do they take neighboring instances into account. Obtaining good upper and lower bounds is crucial to obtain robustness certificates, as tighter bounds lead to lower relaxation error of the GNN activations.

While in Sec. 6.4.3, we relax the discreteness condition of the node attributes $\dot{\boldsymbol{X}}$ in the linear program, it turns out that for the bounds the binary nature of the data can be exploited. More precisely, for every node $m \in \mathcal{N}_{L-2}(t)$, we compute the upper bound

$\boldsymbol{S}^{(2)}_{mj}$ in the second layer for latent dimension j as

$$\begin{aligned}\boldsymbol{S}^{(2)}_{mj} &= \text{sum\_top\_Q}\left([\dot{\boldsymbol{A}}^{(1)}_{mn}\hat{\boldsymbol{S}}^{(2)}_{nji}]_{n\in\mathcal{N}_1(m),i\in\{1,\dots,q\}}\right) + \dot{\boldsymbol{H}}^{(2)}_{mj} \\ \hat{\boldsymbol{S}}^{(2)}_{nji} &= \text{i-th\_largest}\left((1-\dot{\boldsymbol{X}}_{n:})\odot\left[\boldsymbol{W}^{(1)}_{:j}\right]_+ + \dot{\boldsymbol{X}}_{n:}\odot\left[\boldsymbol{W}^{(1)}_{:j}\right]_-\right)\end{aligned} \tag{6.12}$$

Here, i-th_largest$(\cdot)$ denotes the selection of the i-th largest element from the corresponding vector, and sum_top_Q$(\cdot)$ the sum of the Q largest elements from the corresponding list. The first term of the sum in Eq. (6.12) is an upper bound on the *change/increase* in the first hidden layer's activations of node m and hidden dimension j for any admissible perturbation on the attributes $\dot{\boldsymbol{X}}$. The second term are the hidden activations obtained for the (un-perturbed) input $\dot{\boldsymbol{X}}$, i.e. $\dot{\boldsymbol{H}}^{(2)}_{mj} = \dot{\boldsymbol{A}}^{(1)}\dot{\boldsymbol{X}}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}$. In sum we have an upper bound on the hidden activations in the first hidden layer for the perturbed input $\tilde{\boldsymbol{X}}$. Note that, reflecting the interdependence of nodes in the graph, the bounds of a node m depend on the attributes of its neighbors n.

Likewise for the lower bound we use:

$$\begin{aligned}\boldsymbol{R}^{(2)}_{mj} &= \text{-}\,\text{sum\_top\_Q}\left([\dot{\boldsymbol{A}}^{(1)}_{mn}\hat{\boldsymbol{R}}^{(2)}_{nji}]_{n\in\mathcal{N}_1(m),i\in\{1,\dots,q\}}\right) + \dot{\boldsymbol{H}}^{(2)}_{mj} \\ \hat{\boldsymbol{R}}^{(2)}_{nji} &= \text{i-th\_largest}\left(\dot{\boldsymbol{X}}_{n:}\odot\left[\boldsymbol{W}^{(1)}_{:j}\right]_+ + (1-\dot{\boldsymbol{X}}_{n:})\odot\left[\boldsymbol{W}^{(1)}_{:j}\right]_-\right)\end{aligned} \tag{6.13}$$

We need to compute the bounds for each node in the $L-2$ hop neighborhood of the target, i.e., for a GNN with a single hidden layer ($L=3$) we have $\boldsymbol{R}^{(2)}, \boldsymbol{S}^{(2)} \in \mathbb{R}^{\mathcal{N}_1(t)\times h^{(2)}}$.

Corollary 3. *Eqs. (6.12) and (6.13) are valid, and the tightest possible, lower/upper bounds w.r.t. the set of admissible perturbations.*

The proof is in the appendix. For the remaining layers, since the input to them is no longer binary, we adapt the bounds proposed in [189]. Generalized to the GNN we therefore obtain:

$$\begin{aligned}\boldsymbol{R}^{(l)} &= \dot{\boldsymbol{A}}^{(l-1)}\left(\boldsymbol{R}^{(l-1)}\left[\boldsymbol{W}^{(l-1)}\right]_+ - \boldsymbol{S}^{(l-1)}\left[\boldsymbol{W}^{(l-1)}\right]_-\right) \\ \boldsymbol{S}^{(l)} &= \dot{\boldsymbol{A}}^{(l-1)}\left(\boldsymbol{S}^{(l-1)}\left[\boldsymbol{W}^{(l-1)}\right]_+ - \boldsymbol{R}^{(l-1)}\left[\boldsymbol{W}^{(l-1)}\right]_-\right) \\ &\text{for } l = 3,\dots,L-1.\end{aligned}$$

Intuitively, for the upper bounds we assume that the activations in the previous layer take their respective upper bound wherever we have positive weights, and their lower bounds whenever we have negative weights (and the lower bounds are analogous to this). While there exist more computationally involved algorithms to compute more accurate bounds [234], we leave adaptation of such bounds to the graph domain for future work.

It is important to note that all bounds can be computed highly efficiently and one can even back-propagate through them – important aspects for the robust training (Sec. 6.5).

Specifically, one can compute Eqs. (6.12) and (6.13) for all $m \in \mathcal{V}$ (!) and all j *together* in time $\mathcal{O}(h^{(2)} \cdot (N \cdot D + E \cdot q))$ where E is the number of edges in the graph. Note that $\hat{\boldsymbol{R}}^{(2)}_{nj:}$ can be computed in time $\mathcal{O}(D)$ by unordered partial sorting; overall leading to the complexity $\mathcal{O}(N \cdot h^{(2)} \cdot D)$. Likewise the sum of top Q elements can be computed in time $\mathcal{O}(\mathcal{N}_1(m) \cdot q)$ for every $1 \leq j \leq h^{(2)}$ and $m \in \mathcal{V}$, together leading to $\mathcal{O}(E \cdot q \cdot h^{(2)})$.

6.5 Robust Training of GNNs

While being able to certify robustness of a given GNN by itself is extremely valuable for being able to trust the model's output in real-world applications, it is also highly desirable to train classifiers that are (certifiably) robust to adversarial attacks. In this section we show how to use our findings from before to train robust GNNs.

Recall that the value of the dual g can be interpreted as a lower bound on the margin between the two considered classes. As a shortcut, we denote with $\boldsymbol{p}^t_\theta(y, \boldsymbol{\Omega}^{(\cdot)}) = \left[-g^t_{q,Q}\left(\dot{\boldsymbol{X}}, \boldsymbol{c}^k, \boldsymbol{\Omega}^k\right)\right]_{1 \leq k \leq K}$ the K-dimensional vector containing the (negative) dual objective function values for any class k compared to the given class y, i.e. $\boldsymbol{c}^k = \boldsymbol{e}_y - \boldsymbol{e}_k$. That is, node t with class y^*_t is certifiably robust if $\boldsymbol{p}^t_\theta < 0$ for all entries (except the entry at y^*_t which is always 0). Here, θ denotes the parameters of the GNN.

First consider the training objective typically used to train GNNs for node classification:

$$\underset{\theta}{\text{minimize}} \quad \sum_{t \in \mathcal{V}_L} \mathcal{L}\left(f^t_\theta(\dot{\boldsymbol{X}}, \dot{\boldsymbol{A}}), y^*_t\right), \tag{6.14}$$

where $\mathcal{L}$ is the cross entropy function (operating on the logits) and $\mathcal{V}_L$ the set of labeled nodes in the graph. y^*_t denotes the (known) class label of node t.

To improve robustness, in [234] (for classical neural networks) it has been proposed to instead optimize

$$\underset{\theta, \{\boldsymbol{\Omega}^{t,k}\}_{t \in \mathcal{V}_L, 1 \leq k \leq K}}{\text{minimize}} \quad \sum_{t \in \mathcal{V}_L} \mathcal{L}\left(\boldsymbol{p}^t_\theta(y^*_t, \boldsymbol{\Omega}^{t,\cdot}), y^*_t\right) \tag{6.15}$$

which is an upper bound on the worst-case loss achievable. Note that we can omit optimizing over $\boldsymbol{\Omega}$ by setting it to Eq. (6.11). We refer to the loss function in Eq. (6.15) as *robust cross entropy* loss.

One common issue with deep learning models is overconfidence [127], i.e., the models predicting effectively a probability of 1 for one and 0 for the other classes. Applied to Eq. (6.15), this means that the vector $\boldsymbol{p}^t_\theta$ is pushed to contain very large negative numbers: the predictions will not only be robust but also very certain even under the worst perturbation. To facilitate *true* robustness and not false certainty in our model's predictions, we therefore propose an alternative robust loss that we refer to as *robust hinge loss*:

$$\hat{\mathcal{L}}_M(\boldsymbol{p}, y^*) = \sum_{k \neq y*} \max\{0, \boldsymbol{p}_k + M\}. \tag{6.16}$$

This loss is positive if $-\boldsymbol{p}^t_{\theta k} = g^t_{q,Q}\left(\dot{\boldsymbol{X}}, \boldsymbol{c}^k, \boldsymbol{\Omega}^k\right) < M$; and zero otherwise. Put simply: If the loss is zero, the node t is certifiably robust – in this case even guaranteeing a margin of at least M to the decision boundary. Importantly, realizing even larger margins (for the worst-case) is not 'rewarded'.

We combine the *robust hinge loss* with standard cross entropy to obtain the following robust optimization problem

$$\min_{\theta,\boldsymbol{\Omega}} \sum_{t\in\mathcal{V}_L} \hat{\mathcal{L}}_M\left(\boldsymbol{p}^t_\theta(y^*_t, \boldsymbol{\Omega}^{t,\cdot}), y^*_t\right) + \mathcal{L}\left(f^t_\theta(\dot{\boldsymbol{X}}, \dot{\boldsymbol{A}}), y^*_t\right). \tag{6.17}$$

Note that the cross entropy term is operating on the *exact*, non-relaxed GNN, which is a strong advantage over the robust cross entropy loss that only uses the *relaxed* GNN. Thus, we are using the exact GNN model for the node predictions, while the relaxed GNN is only used to ensure robustness. Effectively, if all nodes are robust, the term $\hat{\mathcal{L}}_M$ becomes zero, thus, reducing to the standard cross-entropy loss on the exact GNN (with robustness guarantee).

Robustness in the semi-supervised setting While Eq. (6.17) improves the robustness regarding the labeled nodes, we do not consider the given unlabeled nodes. How to handle the semi-supervised setting which is prevalent in the graph domain, ensuring also robustness for the unlabeled nodes? Note that for the unlabeled nodes, we do not necessarily want robustness certificates with a very large margin (i.e., strongly negative $\boldsymbol{p}^t_\theta$) since the classifier's prediction may be wrong in the first place; this would mean that we encourage the classifier to make very certain predictions even when the predictions are wrong. Instead, we want to reflect in our model that some unlabeled nodes might be close to the decision boundary and not make overconfident predictions in these cases.

Our robust hinge loss provides a natural way to incorporate these goals. By setting a smaller margin M_2 for the unlabeled nodes, we can train our classifier to be robust, but does not encourage worst-case logit differences larger than the specified M_2. Importantly, this does *not* mean that the classifier will be less certain in general, since the cross entropy term is unchanged and if the classifier is already robust, the robust hinge loss is 0. Overall:

$$\begin{aligned}\min_{\theta,\boldsymbol{\Omega}} &\sum_{t\in\mathcal{V}_L} \hat{\mathcal{L}}_{M_1}\left(\boldsymbol{p}^t_\theta(y^*_t, \boldsymbol{\Omega}^{t,\cdot}), y^*_t\right) + \mathcal{L}\left(f^t_\theta(\dot{\boldsymbol{X}}, \dot{\boldsymbol{A}}), y^*_t\right) \\ &+ \sum_{t\in\mathcal{V}\setminus\mathcal{V}_L} \hat{\mathcal{L}}_{M_2}\left(\boldsymbol{p}^t_\theta(\tilde{y}_t, \boldsymbol{\Omega}^{t,\cdot}), \tilde{y}_t\right)\end{aligned} \tag{6.18}$$

where $\tilde{y}_t = \arg\max_k f^t_\theta(\dot{\boldsymbol{X}}, \dot{\boldsymbol{A}})_k$ is the predicted label for node t. Note again that the unlabeled nodes are used for robustness purposes only – making it very different to the principle of self-training (see below). Overall, Eq. (6.18) aims to correctly classify all labeled nodes using the exact GNN, while making sure that *every* node has at least a margin of M_* from the decision boundary even under *worst-case perturbations*.

Eq. (6.18) can be optimized as is. In practice, however, we proceed as follows: We first train the GNN on the *labeled* nodes using Eq. (6.17) until convergence. Then we train on *all* nodes using Eq. (6.18) until convergence.

Discussion Note that the above idea is not applicable to the robust cross entropy loss from Eq. (6.15). One might argue that one could use a GNN trained using Eq. (6.15) to compute predictions for all (or some of the) unlabeled nodes. Then, treating these predictions as the correct (soft-)labels for the nodes and recursively apply the training. This has two undesired effects: If the prediction is very uncertain (i.e. the soft-labels are flat), Eq. (6.15) tries to find a GNN where the worst-case margin exactly matches these uncertain labels (since this minimizes the cross-entropy). The GNN will be forced to keep the prediction uncertain for such instances even if it could do better. On the other hand, if the prediction is very certain (i.e. very peaky), Eq. (6.15) tries to make sure that even in the worst-case the prediction has such high certainty – thus being overconfident in the prediction (which might even be wrong in the first place). Indeed, this case mimics the idea of self-training: In self-training, we first train our model on the labeled nodes. Subsequently, we use the predicted classes of (some of) the unlabeled nodes, pretending these are their true labels; and continue training with them as well. Self-training, however, serves an orthogonal purpose and, in principle, can be used with any of the above models.

Summary When training the GNN, the lower and upper activation bounds are treated as a function of θ, i.e. they are updated accordingly. While this can be done efficiently as discussed in Sec. 6.4.5, it is still the least efficient part of our model and future work might consider incremental computations. Overall, since the dual program in Theorem 4 and the upper/lower activations bounds are differentiable, we can train a robust GNN with gradient descent and standard deep learning libraries. Note again that by setting $\mathbf{\Omega}$ to its default value, we actually only have to optimize over θ – like in standard training. Furthermore, computing $\boldsymbol{p}^t_\theta$ for the default parameters has roughly the same cost as evaluating a usual (sliced) GNN K many times, i.e., it is very efficient.

6.6 Experimental Evaluation

Our experimental contributions are twofold. (i) We evaluate the robustness of traditionally trained GNNs using, and thus analyzing, our certification method. (ii) We show that our robust training procedure can dramatically improve GNNs' robustness while sacrificing only minimal accuracy on the unlabeled nodes.

We evaluate our method on the widely used and publicly available datasets Cora-ML[152], Citeseer[201], and PubMed [201] (see Table A.1 for more information). For every dataset, we allow *local* (i.e., per-node) changes to the node attributes amounting to 1% of the attribute dimension, i.e., $q = 0.01D$. Q is analyzed in detail in the experiments reflecting different perturbation spaces.

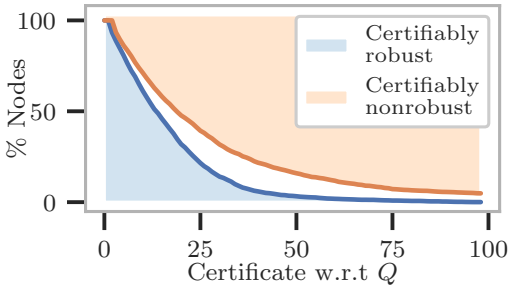


Figure 6.2: Certificates for a GNN with standard training on CORA-ML.

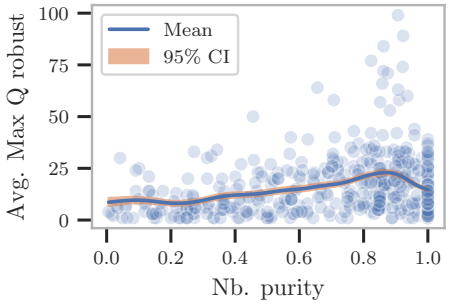


Figure 6.3: Neighborhood purity correlates with robustness.

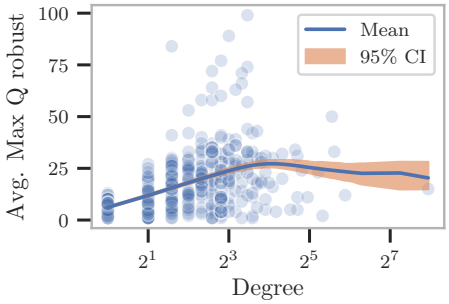


Figure 6.4: Robustness of nodes vs. their degrees.

We refer to the traditional training of GNNs as *Cross Entropy* (short *CE*), to the robust variant of cross entropy as *Robust Cross Entropy (RCE)*, and to our hinge loss variants as *Robust Hinge Loss (RH)* and *Robust Hinge Loss with Unlabeled (RH-U)*, where the latter enforces a margin loss also on the unlabeled nodes. We set M_1, i.e., the margin on the training nodes to $\log(0.9/0.1)$ and M_2 to $\log(0.6/0.4)$ for the unlabeled nodes (*RH-U* only). This means that we train the GNN to (correctly) classify the labeled nodes with output probability of 90% *in the worst case*, and the unlabeled nodes with 60%, reflecting that we do not want our model to be overconfident on the unlabeled nodes. Please note that we do not need to compare against graph adversarial attack models such as [273] since our method gives provable guarantees on the robustness.

While our method can be used for any GNN of the form in Eq. (6.1), we study the well-established GCN [117], which has shown to outperform many more complicated models. Following [117], we consider GCNs with one hidden layer (i.e., $L = 3$), and choose a latent dimensionality of 32. We split the datasets into 10% labeled and 90% unlabeled nodes. See the appendix for further details.

6.6.1 Certificates: Robustness of GNNs

We first start to investigate our (non-)robustness certificates by analyzing GNNs trained using standard cross entropy training. Figure 6.2 shows the main result: for varying Q

we report the percentage of nodes (train+test) which are certifiable robust/non-robust on Cora-ML. We can make two important observations: (i) Our certificates are often very tight. That is, the white area (nodes for which we cannot give any – robustness or non-robustness – certificate) is rather small. Indeed, for any given Q, *at most* 30% of the nodes cannot be certified across all datasets and despite no robust training, highlighting the tightness of our bounds and relaxations and the effectiveness of our certification method. (ii) GNNs trained traditionally are only certifiably robust up to very small perturbations. At $Q = 12$, less than 55% of the nodes are certifiably robust on Cora-ML. In case of Citeseer even less than 20% (Table 6.1; training: CE). Even worse, at this point already *two thirds* (for Citeseer) and a quarter (Cora-ML) of the nodes are certifiably non-robust (i.e. we can find adversarial examples), confirming the issues reported in [273]. PubMed behaves similarly (as we will see later, e.g., in Table 6.1). In our experiments, the labeled nodes are on average more robust than the unlabeled nodes, which is not surprising given that the classifier was not trained using the labels of the latter.

We also investigate what contributes to certain nodes being more robust than others. In Figure 6.3 we see that neighborhood purity (i.e. the share of nodes in a respective node's two-hop neighborhood that is assigned the same class by the classifier) plays an important role. On Cora-ML, almost *all* nodes that are certifiably robust above $Q \geq 50$ have a neighborhood purity of at least 80%. When analyzing the degree (Figure 6.4), it seems that nodes with a medium degree are most robust. While counterintuitive at first, having many neighbors also means a large surface for adversarial attacks. Nodes with low degree, in contrast, might be affected more strongly since each node in its neighborhood has a larger influence.

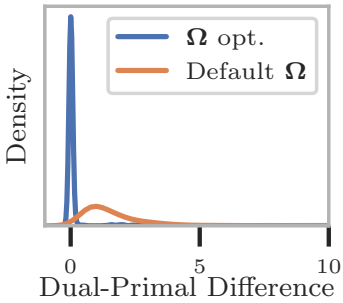


Figure 6.5: Difference of primal and dual bounds.

Tightness of lower bounds Next, we aim to analyze how tight our dual lower bounds are, which we needed to obtain efficient certification. For this, we analyze (i) the value of $g_{q,Q}(\cdot)$ we obtain from our dual solution (either when optimizing over $\mathbf{\Omega}$ are using the default value), compared to (ii) the value of the primal solution we obtain using our construction from Sec. 6.4.4. The smaller the difference, the better. As seen in Figure 6.5, when optimizing over $\mathbf{\Omega}$, for most of the nodes the gap is 0. Thus, indeed we can often find the *exact* minimum of the primal via the dual. As expected, when using the default value for $\mathbf{\Omega}$ the difference between dual and primal is larger. Still, for most nodes the difference is small. Indeed, and more importantly, when considering the actual certificates (where we only need to verify whether the dual is positive; its actual value is not important), the difference between optimizing $\mathbf{\Omega}$ and its default value become negligible: on Cora-ML, the average maximal Q for which we can certify robustness drops by 0.54; Citeseer 0.18; PubMed 2.3. This highlights that we can use the default

Dataset	Training	Avg. Max Q robust	% Robust $Q = 12$	Acc. (labeled)	Acc. (unlabeled)
Citeseer	CE	6.77	0.17	1.00	0.67
	RCE	18.62	0.58	0.99	0.69
	RH	15.51	0.54	0.99	0.68
	RH-U	18.48	0.76	0.99	0.68
Cora-ML	CE	16.36	0.54	1.00	0.83
	RCE	38.58	0.77	1.00	0.83
	RH	32.49	0.74	1.00	0.83
	RH-U	35.58	0.91	1.00	0.83
PubMed	CE	5.82	0.15	0.99	0.86
	RCE	50.68	0.62	0.88	0.84
	RH	48.56	0.62	0.90	0.85
	RH-U	47.56	0.63	0.90	0.86

Table 6.1: Robust training results. Our robust training methods significantly improve the robustness of GNNs while not sacrificing accuracy. Robust training was done for $Q = 12$. Results are averaged over five random data splits.

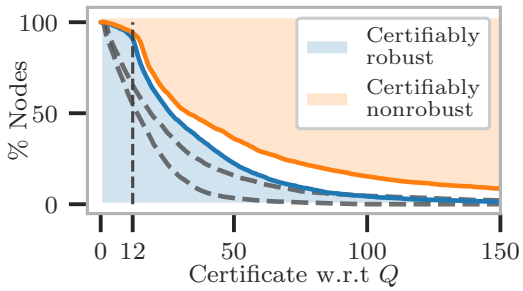


Figure 6.6: Robust training results (Cora-ML). Dashed: no robust training.

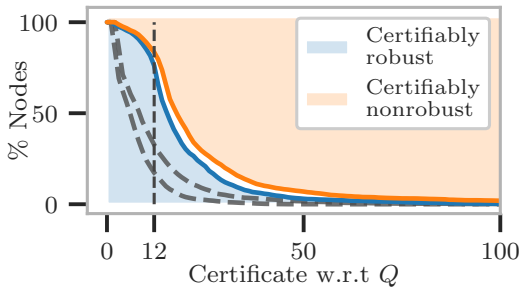


Figure 6.7: Robust training results (Citeseer). Dashed: no rob. training.

values of $\boldsymbol{\Omega}$ to very efficiently certify many or even all nodes in a GNN. In all remaining experiments we, thus, only operate with this default choice.

6.6.2 Robust Training of GNNs

Next, we analyze our robust training procedure. If not mentioned otherwise, we use our robust hinge-loss including the unlabeled nodes *RH-U* and we robustify the models with $Q = 12$ since for this value more than 50% of nodes across our datasets were not certifiably robust (when using standard training).

Figure 6.6 and 6.7 show again the percentage of certified nodes w.r.t. a certain Q – now when using a robustly trained GCN. With dotted lines, we have plotted the curves one obtains for the standard (non-robust) training – e.g. the dotted lines in Fig. 6.6 are the ones already seen in Fig. 6.2. As it becomes clear, with robust training, we can

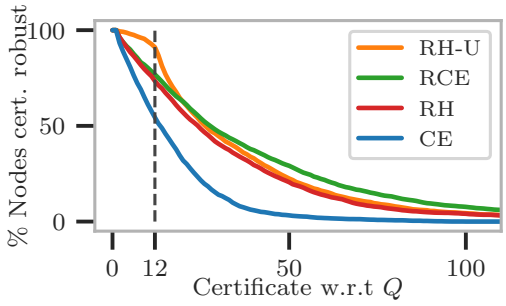


Figure 6.8: RH-U is most successful for robustness at $Q = 12$ (CORA-ML).

dramatically increase the number of nodes which are robust. Almost every node is robust when considering the Q for which the model has been trained for. e.g., for CITESEER, our method is able to quadruple the number of certifiable nodes for $Q = 12$. Put simply: When performing an adversarial attack with $Q \leq 12$ on this model, it cannot do any harm! Moreover the share of nodes that can be certified for any given Q has increased significantly (even though we have not trained the model for $Q > 12$). Most remarkably, nodes for which we certified non-robustness before become now certifiably robust (the blue region above the gray lines).

Accuracy The increased robustness comes at almost *no loss in classification accuracy* as Table 6.1 shows. There we report the results for all datasets and all training principles. The last two columns show the accuracy obtained for node classification (for train and test nodes separately). In some cases, our robust classifiers even outperform the non-robust one on the unlabeled nodes. Interestingly, for PUBMED we see that the accuracy on the labeled nodes drops to the accuracy on the unlabeled nodes. This indicates that our method can even improve generalization.

Training principles Comparing the different robust training procedures (also given in more detail in Figure 6.8), we see that RH-U achieves significantly higher robustness when considering $Q = 12$. This is shown by the third-last column in the table, where the percentage of nodes which are certifiably robust for $Q = 12$ (i.e., the Q the models have been robustified for) is shown. The third column shows the largest Q for which a node is still certifiably robust (averaged over all nodes). As shown, for all training principles the average exceeds the value of 12.

Effect of training with Q If we strongly increase the Q for which the classifier is trained for, we only observe a small drop in the classification accuracy. E.g., training accuracy drops from 99% to 87% when going from $Q = 12$ to 48, while test accuracy stays almost unchanged (68% vs. 66%) on CITESEER. We attribute this to the fact that the GNN still uses the *normal* CE loss in addition to our robust hinge loss during training. Figure 6.10 shows the results for CORA-ML where we trained three models with different Q. To clarify: We have to distinguish between the Q used for training a model (mentioned in

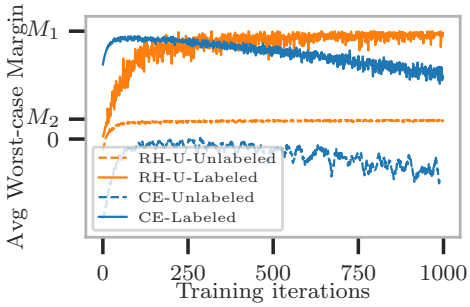


Figure 6.9: Training dynamics.

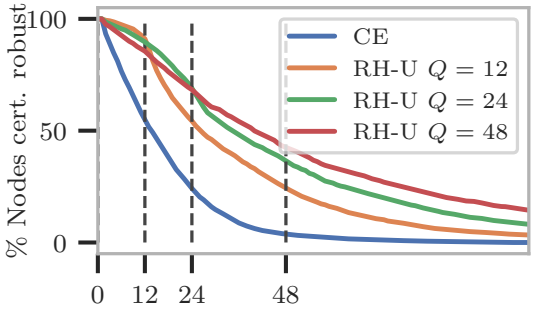


Figure 6.10: Robust training, different Q.

the legend) and the Q we are computing certificates for (the x-axis). We see: (i) Clearly, all trainings lead to significantly more robust models. Though, the larger Q, the harder it gets. (ii) Importantly, each model is the 'winner in robustness' when considering the Q for which the model has been trained for.

Training Dynamics Lastly, we analyze the behavior when training a GCN using either standard training or robust training with *RH-U*. In Figure 6.9 we monitor the worst-case margin (averaged over a minibatch of nodes; separately for the labeled and unlabeled nodes) obtained in each training iteration. As seen, with *RH-U* the worst-case margin *increases* to the specified values M_1/M_2 – i.e., making them robust. In contrast, for standard training the worst-case margin *decreases*. Specifically the unlabeled nodes (which account to 90% of all nodes) are not robust.

Overall, all experiments show that our robust training is highly effective: robustness is increased while the accuracy is still high.

6.7 Conclusion

We proposed the first approach for certifying robustness of GNNs, considering perturbations of the node attributes under a challenging L_0 perturbation budget and tackling the discrete data domain. By relaxing the GNN and considering the dual, we realized an efficient computation of our certificates – simultaneously our experiments have shown that our certificates are tight since for most nodes a certificate can be given. We have shown that traditional training of GNNs leads to non-robust models that can easily be fooled. In contrast, using our novel (semi-supervised) robust training the resulting GNNs are shown to be much more robust. All this is achieved with only a minor effect on the classification accuracy. In the following chapter, we consider perturbations of the graph structure.

Retrospective

In this chapter we present the first robustness certificates for message-passing GNNs, along with a highly effective robust training procedure. While we show that standard GCNs have very limited provable robustness properties, which are improved by our robust training scheme, there are two important limitations of the approach presented in this chapter. The fist one is that our certification and robust training procedure is limited to GNNs with static aggregation weights and thus does not directly apply to models such as GAT [224]. The second limitation is that our certificates are not applicable to graph structure perturbations; we address this limitation in the following chapter. There, we also intriguingly observe increased robustness to *structure* perturbations for models trained with our robust training method for *attribute* robustness, which we discuss in more detail in Chapter 8. Moreover, our certificates are limited to binary node attributes, though an extension to continuous-valued attributes seems possible.

Another aspect to consider in practice is the need to set the local and global budgets q and Q in advance before training. This could be difficult as it might not be clear to a practitioner what magnitude of perturbations to expect.

7 Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations

7.1 Introduction

In Part II we have established that graph neural networks are highly non-robust w.r.t. adversarial attacks. In the previous chapter we presented the first method for robustness certification and robust training for GNNs w.r.t. perturbations of the node attribute. So far, there exist no effective robustness certification methods for the popular graph convolutional network (GCN) against perturbations of the graph structure. In this chapter, for the first time we close this gap and present a technique for certified robustness of GCN's predictions.

In robustness certification for 'traditional' data (i.e. images) it is sufficient to consider the data instances individually and determine whether we can certify a robust prediction. Robustness certification for graph neural networks is significantly more challenging since we have to take into account all nodes that influence the target node's prediction via message passing. The approach presented in the previous chapter tackles such robustness certification of graph convolutional networks – however, it only considers perturbations to the node features.

In this chapter we address for the first time the even more challenging setting where we assume the *graph structure* can be altered by an attacker. Specifically, we consider the realistic case where the attacker is allowed to insert new edges. This scenario reflects real-world conditions where new edges (corresponding to, e.g. links, likes, following, etc.) are cheap and easy to inject. A certificate issued by our method states that a node's prediction *could not have been altered* by edges potentially inserted by an attacker.

From a technical view, we solve the following three challenges:

(1) **Nonconvexity of the neural network**. Neural networks are (in general) nonconvex functions; therefore finding the optimal (i.e. worst-case) perturbation is intractable.

(2) **Discreteness of the data**. Since the adjacency matrix is binary, the number of possible perturbations grows exponentially with the perturbation budget; therfore enumerating and testing all admissible perturbations is intractable.

(3) **Modification of the message passing scheme**. While previous certificates deal with what happens when the *input* to the (static) MPNN is modified, we aim to certify robustness for cases when the graph structure – and therefore the way embeddings are

propagated – changes. From the perspective of an individual node, the *computation graph itself* changes as the output is computed from a different set of nodes.

Indeed, **challenge (1)** is also encountered when certifying robustness in more traditional scenarios. For this aspect, we thus largely follow the convex relaxation of GCN presented by [277], which relies on a relaxation of the ReLU activation function introduced by [234].

Challenge (2), in contrast, requires special care: While [277] handles binary *attribute data* by performing a continuous relaxation, we will show in Section 7.3.2 that a standard continuous relaxation is not well-suited for perturbations of the *graph structure*. Instead, one of our contributions is to show how to bypass the problem altogether by working directly on the (continuous-valued) degree-normalized message passing matrix used by GCN. To this end we carefully construct *induced constraints* on the normalized message passing matrix from L_0 constraints on the (binary) adjacency matrix.

Challenge (3) – how to deal with changes to the message passing matrix – is completely unstudied and therefore our second main contribution. The difficulty lies in the fact that the message passing matrix (which becomes *variable*) is used at *each layer* – as opposed to the input features, which appear only in the first layer of the neural network. Thus, the ReLU relaxations of [277, 234] no longer lead to convex (linear) constraints but become nonconvex (details are discussed in Section 7.3.4). Indeed, we will show that the problem can be expressed as a jointly constrained bilinear program – "one of the most persistently difficult and recurrent nonconvex problems in mathematical programming" [3]. To solve this challenging problem we propose a novel branch-and-bound algorithm that effectively exploits key insights to obtain lower bounds on the worst-case change in a node's prediction. If positive, these lower bounds serve as robustness certificates, stating that a node's prediction does not change under *any* admissible set of perturbations.

Related Work GNNs are a fundamental part of the modern machine learning landscape and have been successfully used for a variety of tasks. However, GNNs are highly sensitive to small adversarial perturbations [273, 56, 276]. While some (heuristic) defenses exist [245, 69, 268], we can never assume attackers will not be able to break them.Robustness certificates, on the other hand, provide provable guarantees that no perturbation regarding a specific perturbation model will change the prediction of a sample. Only a few robustness certificates for graphs have been developed. While [277] can only handle perturbations to the node attributes, [24] is limited to a specific class of graph-models based on PageRank, not covering the highly important principle of graph convolutional networks. Our work closes this gap by providing the first robustness certificate for GCNs under structure perturbations. In concurrent work, [25] provide another certification method for structure perturbations.

7.2 Preliminaries

We consider the task of (semi-supervised) node classification in a single large graph with D-dimensional node features. Let $G = (\boldsymbol{A}, \boldsymbol{X})$ be an attributed, unweighted graph, where $\boldsymbol{A} \in \{0,1\}^{N \times N}$ is the adjacency matrix and $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ are the nodes' features.

W.l.o.g. we assume the node-ids to be $\mathcal{V} = \{1, \ldots, N\}$. Given a subset $\mathcal{V}_L \subseteq \mathcal{V}$ of labeled nodes, with class labels from $\mathcal{C} = \{1, 2, \ldots, K\}$, the goal is to assign each node in G to one class in $\mathcal{C}$. We focus on node classification employing graph neural networks. In particular, we consider graph convolutional networks where the latent representations $\boldsymbol{H}^{(l)}$ at layer l are of the form

$$\boldsymbol{H}^{(l)} = \sigma^{(l)}\left(\hat{\boldsymbol{H}}^{(l)}\right), \text{ where} \tag{7.1}$$

$$\hat{\boldsymbol{H}}^{(l)} = \hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l-1)} + \boldsymbol{b}^{(l-1)} \tag{7.2}$$

for $l = 2, ..., L$, where $\boldsymbol{H}^{(1)} = \boldsymbol{X}$ and activation functions given by

$$\sigma^{(L)}(\cdot) = \text{softmax}(\cdot), \qquad \sigma^{(l)}(\cdot) = \text{ReLU}(\cdot)$$

for $l = 2, ..., L-1$. Here, $\mathcal{T}(\cdot)$ is a transformation that is applied to the adjacency matrix in order to obtain the message passing matrix. This message passing matrix defines how the activations are propagated in the network. In GCN [117], for example,

$$\hat{\boldsymbol{A}} = \boldsymbol{D}^{-\frac{1}{2}}(\boldsymbol{A} + \boldsymbol{I}_{N\times N})\boldsymbol{D}^{-\frac{1}{2}}, \text{ and } \boldsymbol{D}_{ii} = d_i = 1 + \sum\nolimits_j \boldsymbol{A}_{ij} \tag{7.3}$$

The $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are the trainable weights of the graph neural network (summarized as θ), typically learned by minimizing the cross-entropy loss on the given labeled training nodes $\mathcal{V}_L$.

The output $\boldsymbol{H}^{(L)}_{vc}$ is the probability of assigning node v to class c, and we denote the logits (before applying softmax) as $\hat{\boldsymbol{H}}^{(L)}_v =: f^v_\theta(\boldsymbol{X}, \hat{\boldsymbol{A}}$ indicating the dependency on $\boldsymbol{X}$, $\boldsymbol{A}$, and θ.

Notations: We denote with $\mathcal{N}_t$ the 1-hop neighborhood of a node t, i.e. all neighbors of t and the node t itself (regarding the original, unperturbed graph). Given a matrix $\boldsymbol{X}$, we denote its positive part with $[\boldsymbol{X}]_+ = \max(\boldsymbol{X}, 0)$ where the max is applied entry-wise. Similarly, the negative part is $[\boldsymbol{X}]_- = -\min(\boldsymbol{X}, 0)$, which are non-negative numbers. We denote with $h^{(l)}$ the dimensionality of the latent space in layer l, i.e. $\boldsymbol{H}^{(l)} \in \mathbb{R}^{N\times h^{(l)}}$.

7.3 Robustness Certification for Graph Structure Perturbations

7.3.1 Problem Definition

To derive our robustness certificates, we first set up the optimization problem we aim to solve. We assume that we are provided with an already trained GCN with weights θ as well as a (potentially corrupted) graph structure in form of an adjacency matrix $\boldsymbol{A}$. The true (unperturbed) graph structure $\boldsymbol{A}^*$ is not known, but $\boldsymbol{A}$ is assumed to be reachable from $\boldsymbol{A}^*$ via an admissible set of perturbations.

We aim to certify robustness for a single target node t at a time (called the *target* node). Such certificate guarantees that the prediction made for node t *cannot have been altered* by the attacker. In practice, predictions of nodes for which robustness cannot be certified can be ignored or sent to an expert for verification.

Formally we aim to solve:

Problem 4. *Given a graph G with adjacency matrix $\boldsymbol{A}$, a target node t, and a* GCN *with parameters θ. Let y^* denote the class of node t (ground truth or predicted). The worst case margin between classes y^* and y achievable under some set $\mathcal{A}(\boldsymbol{A})$ of admissible perturbations to the graph structure is given by*

$$m^t(y^*, y) := \underset{\tilde{\boldsymbol{A}}}{\text{minimize}} \; f_\theta^t(\boldsymbol{X}, \mathcal{T}(\tilde{\boldsymbol{A}}))_{y^*} - f_\theta^t(\boldsymbol{X}, \mathcal{T}(\tilde{\boldsymbol{A}}))_y \quad \text{subject to } \tilde{\boldsymbol{A}} \in \mathcal{A}(\boldsymbol{A}) \tag{7.4}$$

If $m^t(y^*, y) > 0$ for all classes $y \neq y^*$, the neural network is certifiably robust w.r.t. node t and $\mathcal{A}$.

If the minimum obtained from Eq. (7.4) is positive, it means that there exists *no* adversarial example (within our defined admissible perturbations) that leads to the classifier changing its prediction to the other class y – i.e. the logit of class y^* is always larger than the one of y. Note that since we are interested in certifying an individual target node t's prediction, we omit t in the following when it is clear from the context.

Roadmap Solving the above optimization problem is hard due to the three challenges mentioned in the introduction. Nevertheless, as we will show in the following, we can find *lower bounds* on the minimum of the original problem by (i) optimizing over the continuous-valued message passing matrix $\hat{\boldsymbol{A}}$ instead of the binary adjacency matrix $\boldsymbol{A}$; (ii) performing relaxations of the activation functions in the neural network; (iii) expressing the problem as a jointly-constrained bilinear program and proposing a novel branch-and-bound algorithm. Finally, if the lower bound is positive, the classifier is robust w.r.t. admissible perturbations.

7.3.2 Optimization over the graph structure

In this section we address **Challenge (2)** – how to efficiently optimize over the discrete graph structure – from the introduction.

First, it is important to set reasonable constraints to the perturbations the attacker can perform such that resulting certificates reflect realistic attacks. For discrete data (such as the graph structure $\boldsymbol{A}$), a natural norm to measure 'distance' is the *number* of perturbed elements, as measured by the non-convex L_0 norm. Here we translate the setup of [277], who introduce both local and global L_0 constraints on the node attributes, to graph structure perturbations.

More precisely, we allow the attacker to insert at most $\boldsymbol{q}_i$ edges to any node i, and at most $Q \in \mathbb{N}$ edges across the whole graph. Formally, any admissible perturbed adjacency matrix $\tilde{\boldsymbol{A}}$ must be in:

$$\begin{aligned} \mathcal{A}(\boldsymbol{A}) = \Big\{ & \tilde{\boldsymbol{A}} \in \{0,1\}^{N \times N} \mid \tilde{\boldsymbol{A}}_{ij} \leq \boldsymbol{A}_{ij} \wedge \tilde{\boldsymbol{A}} = \tilde{\boldsymbol{A}}^T \\ & \wedge \|\tilde{\boldsymbol{A}} - \boldsymbol{A}\|_0 \leq 2Q \\ & \wedge \|\tilde{\boldsymbol{A}}_i - \boldsymbol{A}_i\|_0 \leq \boldsymbol{q}_i \; \forall 1 \leq i \leq N \Big\}. \end{aligned} \tag{7.5}$$

Recall that we assume an attacker has potentially *inserted* edges to the (unknown) original graph structure $\boldsymbol{A}^*$ to produce $\boldsymbol{A}$. This means that $\boldsymbol{A}^*$ must be reachable from $\boldsymbol{A}$ by *removing* edges. Hence in Eq. (7.5) we have the constraints $\tilde{\boldsymbol{A}}_{ij} \leq \boldsymbol{A}_{ij}$ in the definition of the admissible perturbations on $\boldsymbol{A}$. Note also that we consider only undirected graphs, hence the constraint $\tilde{\boldsymbol{A}} = \tilde{\boldsymbol{A}}^T$ and $2Q$ in the constraint, even though in principle our certification procedure also applies to directed graphs. We further assume $\boldsymbol{q}_i$ to be smaller then node i's degree, in order to prevent singleton nodes.

One traditional approach (e.g. pursued by [277]) to optimize over a discrete variable is to perform a continuous relaxation, i.e. constrain the entries to be in $\tilde{\boldsymbol{A}}_{ij} \in [0, 1]$. However, recall from Eq. (7.3) that GCN uses a degree-normalized message passing matrix

$$\hat{\boldsymbol{A}}_{ij} = \begin{cases} \frac{1}{\sqrt{d_i d_j}} & \text{if } \boldsymbol{A}_{ij} = 1 \vee i = j \\ 0 & \text{else} \end{cases}. \tag{7.6}$$

The terms $\frac{1}{\sqrt{d_i d_j}} = \frac{1}{\sqrt{(1+\sum_k \boldsymbol{A}_{ik}) \cdot (1+\sum_k \boldsymbol{A}_{jk})}}$ contain nonconvex quadratic terms of the form $\boldsymbol{A}_{ij}\boldsymbol{A}_{kl}$, and therefore any optimization problem involving $\tilde{\boldsymbol{A}}$ in the objective function is not convex in the variables $\tilde{\boldsymbol{A}}_{ij}$. Thus, a simple continuous relaxation does not lead to a tractable optimization problem.

As an alternative we propose to optimize over the (continuous-valued) degree-normalized message-passing matrix (i.e. the matrix after having applied $\mathcal{T}$) instead of the (binary) adjacency matrix $\tilde{\boldsymbol{A}}$. We denote the variable corresponding to the message-passing matrix by $\hat{\boldsymbol{A}}$. This has the benefits of avoiding to optimize over a discrete variable as well as bypassing the nonconvex degree-normalization procedure of GCN. That is, we replace Eq. 7.4 by

$$\begin{aligned} \hat{m}^t(y^*, y) := \; & \underset{\hat{\boldsymbol{A}}}{\text{minimize}} \; f_\theta^t(\boldsymbol{X}, \hat{\boldsymbol{A}})_{y^*} - f_\theta^t(\boldsymbol{X}, \hat{\boldsymbol{A}})_y \\ & \text{subject to } \hat{\boldsymbol{A}} \in \hat{\mathcal{A}}(\boldsymbol{A}). \end{aligned} \tag{7.7}$$

Note that the neural network f_θ^t now directly takes as input the degree-normalized message-passing matrix instead of the $\hat{\boldsymbol{A}}$, i.e., we have absorbed the degree-normalization procedure into the optimization problem. However, we now have to carefully design the set of admissible degree-normalized message passing matrices $\hat{\mathcal{A}}(\boldsymbol{A})$ based on the perturbation budget and the graph at hand.

It is crucial that $\hat{\mathcal{A}}(\boldsymbol{A}) \supseteq \mathcal{T}(\mathcal{A}(\boldsymbol{A}))$. Here, $\mathcal{T}(\mathcal{A}(\boldsymbol{A}))$ denotes the set of all degree-normalized message-passing matrices that are produced from adjacency matrices in $\mathcal{A}(\boldsymbol{A})$. In other words, *any* degree-normalized matrix that could be produced by first performing discrete perturbations to $\boldsymbol{A}$ and then degree-normalizing the resulting binary matrix $\tilde{\boldsymbol{A}}$ must be included in this new set of admissible matrices. In this case it follows that $\hat{m}^t(y^*, y) \leq m^t(y^*, y)$, i.e., the optimum of our modified optimization problem in Eq. (7.7) is a lower bound on the original problem's optimal value. Thus if Eq. (7.7) leads to a positive value, we have certificate.

Clearly, instantiating $\hat{\mathcal{A}}(\boldsymbol{A})$ with all matrices $\hat{\boldsymbol{A}} \in [0, 1]^{N \times N}$ fulfills this property; however, it leads to a very loose approximation. Thus, we would also like $\hat{\mathcal{A}} \setminus \mathcal{T}(\mathcal{A})$

to be as small as possible, meaning that any relaxations leading to $\hat{\mathcal{A}}$ are as tight as possible. In the following we derive *induced constraints* that give rise to a valid and tight set of admissible message passing matrices $\hat{\mathcal{A}}$ and are convex, thus enabling tractable optimization.

7.3.2.1 Induced constraints on $\hat{\mathcal{A}}$

In this section we construct a set of constraints on the message-passing matrix $\hat{\boldsymbol{A}}$ that are induced by the local and global budgets on the change on the adjacency matrix (see Eq. (7.5)) as well as the degree-normalization procedure of obtaining the message-passing matrix in GCN (Eq. (7.3)). These constraints reflect necessary conditions every matrix from $\mathcal{T}(\mathcal{A})$ fulfills; thus our search space over $\hat{\boldsymbol{A}}$ can be restricted to a smaller domain. To enable efficient optimization, we further require these constraints to be linear (resp., can be reformulated to a linear constraint) as well as are efficient to compute.

In the following we derive the constraints in detail since they are one core contribution of our work. The eager reader might directly jump to Equation (7.11) for the final result.

For convenience we denote with r_i the number of edges which have been removed from the node i for an arbitrary perturbed matrix $\tilde{\boldsymbol{A}} \in \mathcal{A}$. Note that $0 \leq r_i \leq \boldsymbol{q}_i$. We further introduce $\mathcal{D}_i \subset \mathcal{N}_i$ as the set of edges being removed from node i (recall that $\mathcal{N}_i$ are the neighbors in the original graph including i).

Element-wise bounds We can bound every element of $\hat{\boldsymbol{A}}$ individually by $\boldsymbol{L}_{ij} \leq \hat{\boldsymbol{A}}_{ij} \leq \boldsymbol{U}_{ij}$. For the lower bound we have (i) $\boldsymbol{L}_{ii} = \hat{\boldsymbol{A}}_{ii}$, since the self loops in the preprocessing procedure cannot be removed and $\frac{1}{d_i - r_i} \geq \frac{1}{d_i}$ (ii) everywhere else we have $\boldsymbol{L}_{ij} = 0$, representing a potential deletion of the edge.

The upper bound is $\boldsymbol{U}_{ij} = \min\{\boldsymbol{A}_{ij}, ((d_i - \boldsymbol{q}_i)(d_j - \boldsymbol{q}_j))^{-\frac{1}{2}}\}$ for $i \neq j$. The first term within the min ensures that no edges are added in the certification procedure. This technically means that, in the following, we *only* need to consider entries (i, j) for which there exists an edge in $\boldsymbol{A}$, leading to a sparse optimization problem. For the second term, note that the degrees of all nodes can never increase and since they appear in the denominator (see Eq. (7.6)), $((d_i - \boldsymbol{q}_i)(d_j - \boldsymbol{q}_j))^{-\frac{1}{2}}$ is an upper bound on the individual entries $\hat{\boldsymbol{A}}_{ij}$. For $\boldsymbol{U}_{ii}$ we use only the second term of the min.

Please note that from the above discussion we can also conclude that $\hat{\boldsymbol{A}}_{ij} = 0$ if the edge between i and j is deleted and $\hat{\boldsymbol{A}}_{ij} \geq \hat{\boldsymbol{A}}_{ij}$ else. While we cannot use this insight immediately (since it forms a non-convex set), we will show in Section 7.3.4.3 how to use it.

Row-wise bounds (I) The element-wise bounds do not take dependencies between elements into account. Thus, it would, e.g., be possible to set for a single node i all non-diagonal $\hat{\boldsymbol{A}}_{ij}$ to 0; likely violating the local and global budget. To prevent this, we now introduce constraints on specific row sums of $\hat{\boldsymbol{A}}$.

For this we first rephrase the row sum as

$$\sum_j \hat{\boldsymbol{A}}_{ij} = \frac{1}{\sqrt{d_i - r_i}} \sum_{\substack{j \neq i \\ j \in \mathcal{N}_i \setminus \mathcal{D}_i}} \frac{1}{\sqrt{d_j - r_j}} + \frac{1}{d_i - r_i} \tag{7.8}$$

When removing an edge from node i (i.e., increasing r_i) we can observe two opposing effects: the term in front of the sum increases; while the sum itself contains fewer summands and, thus, decreases (assuming the other r_j fixed). Thus, our general idea is to upper/lower bound the value the sum can take for a fixed r_i. Plugging this bound on the sum in Eq. (7.8) leads to an upper/lower bound for a fixed r_i, denoted by $U_i^{\text{row}}(r_i)$ and $L_i^{\text{row}}(r_i)$, respectively. Now, by simply evaluating $U_i^{\text{row}}(r_i)$ and $L_i^{\text{row}}(r_i)$ for $0 \leq r_i \leq \boldsymbol{q}_i$ we can find the overall row-bound in an efficient way (e.g., for the lower bound $L_i^{\text{row}} = \min_{r_i} L_i^{\text{row}}(r_i)$).

(1) We start with the lower bound $L_i^{\text{row}} \leq \sum_j \hat{\boldsymbol{A}}_{ij}$ since the solution is easy to see. For a fixed r_i, the smallest possible sum is achieved by keeping those nodes which have the largest degree. Clearly, we set $r_j = 0$ since this leads to the smallest value. In short: To obtain the lower bound on the sum, we simply sort the nodes based on their degree in descending order and sum up the first $d_i - 1 - r_i$ terms $1/\sqrt{d_j}$ (-1 since the self-loops is counted in d_i).

(2) For the upper bound $\sum_j \hat{\boldsymbol{A}}_{ij} \leq U_i^{\text{row}}$, we need to find an upper bound for the sum. Clearly, in such a scenario $r_j > 0$ since then the fractions get larger. However, we cannot simply set every $r_j = q_j$ since this would violate the overall budget constraint. The question is now twofold: how to select the terms/nodes to remove from the sum, and how to spend the remaining budget to increase the remaining terms? Clearly, we select the smallest r_i summands (corresponding to the neighbors with largest degree) to drop from the sum. For the remaining nodes $\mathcal{M}$, we have to solve the following optimization problem:

$$\max_{r_j, j \in \mathcal{M}} \sum_{j \in \mathcal{M}} \frac{1}{\sqrt{d_j - r_j}} \qquad \text{s.t.} \sum_{j \in \mathcal{M}} r_j \leq B \tag{7.9}$$

where B is the remaining budget one can spend. Note that we cannot assume that the remaining budget is simply $Q - r_i$. The reason is that a single edge removal (i.e. one 'unit' from Q) can lead to *two* neighbors' degrees reducing by one, if these neighbors are connected by this edge. Thus, in the worst case, if all neighbors are connected with each other, we can effectively spend twice the remaining removal budget to increase the r_j. Therefore we denote the number of neighbors of i that are connected to each other as $\mathcal{N}_i^{\text{conn}}$, from which $r^{\text{nbs}} = \min\{Q - r_i, |\mathcal{N}_i^{\text{conn}}|\}$ can become disconnected using the remaining budget, leading to a total remaining budget to increase r_j of $B = 2 \cdot r^{\text{nbs}} + (Q - r_i - r^{\text{nbs}})$.

As it turns out, we can solve the optimization problem (7.9) exactly using a greedy approach: We maximize the term by sorting the neighbors $\mathcal{N}_i$ by their degree in ascending order, and, starting with the lowest degree, use all available budget $\boldsymbol{q}_j$ to reduce the neighbors' degrees, until the budget B is depleted. This is a valid upper bound on the

sum because $h(x) = \frac{1}{\sqrt{x}}, x > 0,\ x \in \mathbb{N}$ is a monotonically decreasing function whose second derivative is positive everywhere, i.e., $\frac{1}{\sqrt{u}} - \frac{1}{\sqrt{u-1}} \geq \frac{1}{\sqrt{u+1}} - \frac{1}{\sqrt{u}}$ for $u \in \mathbb{N} > 1$. Put simply: the larger the term $\frac{1}{\sqrt{d_j - r_j}}$ already is, the more effect has increasing r_j on increasing it even further. Thus, the strategy outlined above is guaranteed to maximize (7.9).

Complexity analysis Both bounds require sorting the neighbors based on their degree, which can be done once. We have to compute $\boldsymbol{q}_i$ many terms $U_i^{\text{row}}(r_i)$ and $L_i^{\text{row}}(r_i)$. Each term, however, can be computed incrementally based on the previous r_i (since one only iterates over more elements in the sorted list). Thus, to compute all terms, we have to iterate through all neighbors at most once. Thus the overall complexity is $\mathcal{O}(|\mathcal{N}_i| \log |\mathcal{N}_i| + |\mathcal{N}_i|) = \mathcal{O}(d_i \log d_i)$

Row-wise bounds (II) (1) So far we have defined lower and upper bounds on the row sum $\sum_j \hat{\boldsymbol{A}}_{ij}$ of node i. We additionally define constraints on the L_1 norm of the difference between the perturbed message-passing matrix $\hat{\boldsymbol{A}}$ and the original message-passing matrix $\hat{\boldsymbol{A}}$: $\sum_j |\hat{\boldsymbol{A}}_{ij} - \hat{\boldsymbol{A}}_{ij}| \leq \bar{U}_i^{\text{row}}$. This covers the case where large positive and large negative changes mostly cancel each other out, such that the lower and upper bounds on the row sum are not violated even though the perturbed message-passing matrix is substantially different.

Again, we denote the set of nodes whose edges to i are removed by $\mathcal{D}_i$. For any term $j \in \mathcal{D}_i$ the absolute change is $|\hat{\boldsymbol{A}}_{ij} - \hat{\boldsymbol{A}}_{ij}| = \hat{\boldsymbol{A}}_{ij}$ since it corresponds to a removed edge. For any node $j \in \mathcal{N}_i \backslash \mathcal{D}_i$ (which also includes the node i itself) the absolute change is at most $\delta_{ij} = \boldsymbol{U}_{ij} - \hat{\boldsymbol{A}}_{ij}$. Assuming that all terms not removed take their maximum value, we aim to solve the following optimization problem:

$$\begin{aligned} \bar{U}_i^{\text{row}} &= \max_{\mathcal{D}_i} \sum_{j \in \mathcal{D}_i} \hat{\boldsymbol{A}}_{ij} + \sum_{j \in \mathcal{N}_i \backslash \mathcal{D}_i} \delta_{ij} \\ &= \sum_{j \in \mathcal{N}_i} \delta_{ij} + \max_{\mathcal{D}_i} \sum_{j \in \mathcal{D}_i} \hat{\boldsymbol{A}}_{ij} - \sum_{j \in \mathcal{D}_i} \delta_{ij} \\ &= \sum_{j \in \mathcal{N}_i} \delta_{ij} + \max_{\mathcal{D}_i} \sum_{j \in \mathcal{D}_i} 2 \cdot \hat{\boldsymbol{A}}_{ij} - \boldsymbol{U}_{ij} \\ &\quad s.t.\ |\mathcal{D}_i| \leq \boldsymbol{q}_i \end{aligned} \tag{7.10}$$

Hence we sort all neighbors of node i by their value $2 \cdot \hat{\boldsymbol{A}}_{ij} - \boldsymbol{U}_{ij}$ and choose the largest $\boldsymbol{q}_i$ values to obtain $\bar{U}^{\text{row}}$.

(2) Another measure to avoid cancelling positive and negative changes to $\mathcal{T}(\boldsymbol{A})$ is to constrain the total change in the negative direction. That is, we consider only the change regarding edge removal – effectively discarding from Eq. (7.10) the change in positive direction (i.e. the δ_{ij} terms):

$$\sum_{j \neq i} \left[\hat{\boldsymbol{A}}_{ij} - \mathcal{T}(\boldsymbol{A})_{ij} \right]_- \leq \Delta_i^- = \max_{\mathcal{D}_i} \sum_{j \in \mathcal{D}_i} \hat{\boldsymbol{A}}_{ij} \quad s.t.\ |\mathcal{D}_i| \leq \boldsymbol{q}_i$$

The solution $\mathcal{D}_i$ is simply the set of the $\boldsymbol{q}_i$ neighbors with lowest degree, which are allowed to be disconnected from i (i.e., doing so does not lead to singleton nodes).

Complexity analysis: Like before, the dominating complexity is sorting the neighbors, leading to $\mathcal{O}(d_i \log d_i)$.

Global bounds Besides entry-wise and row-wise constraints we can also make use of global constraints on $\hat{\boldsymbol{A}}$.

(1) An upper bound $\bar{U}_{\text{glob}}$ on the L_1 difference of $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{A}}$ can be obtained in a similar fashion as the row-wise bound (Eq. 7.10). Formally we want to solve

$$\begin{aligned}\bar{U}_{\text{glob}} &= \max_{\mathcal{D}} \sum_{\substack{i<j \\ (i,j)\in\mathcal{D}}} \mathcal{T}(\boldsymbol{A})_{ij} + \sum_{\substack{i\leq j \\ (i,j)\notin\mathcal{D}}} \delta_{ij} \\ &= \sum_{i\leq j} \delta_{ij} + \max_{\mathcal{D}} \sum_{\substack{i<j \\ (i,j)\in\mathcal{D}}} 2\cdot\mathcal{T}(\boldsymbol{A})_{ij} - \boldsymbol{U}_{ij} \\ &\text{s.t. } |\mathcal{D}| \leq Q\end{aligned}$$

where $\mathcal{D}$ represents the set of removed edges (this set does not include the self-loops since they cannot be removed). As before, the optimal solution is obtained by selecting the Q entries (i,j) with largest values $2\cdot\mathcal{T}(\boldsymbol{A})_{ij} - \boldsymbol{U}_{ij}$ (excl. diagonal terms).

(2) We can also compute an upper bound Δ^-_{glob} on the negative change analogously to Δ^-_i. That is,

$$\sum_{i<j} \left[\hat{\boldsymbol{A}}_{ij} - \hat{\boldsymbol{A}}_{ij}\right]_- \leq \Delta^-_{\text{glob}} = \max_{\mathcal{D}} \sum_{\substack{i<j \\ (i,j)\in\mathcal{D}}} \mathcal{T}(\boldsymbol{A})_{ij}\,, \quad s.t.\ |\mathcal{D}| \leq Q$$

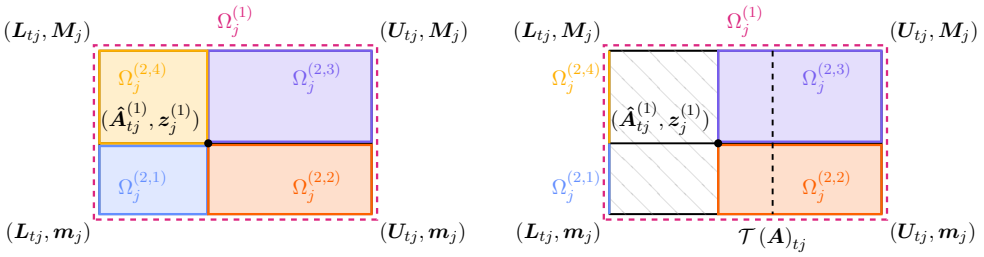
which is maximized by setting $\mathcal{D}$ to the Q largest values $\mathcal{T}(\boldsymbol{A})_{ij}\,, i<j$.

Complexity analysis: The bounds can again be computed based on sorting; now based on the full edge set. This leads to $\mathcal{O}(E \log E)$, where E is the number of edges in $\boldsymbol{A}$.

Summary The intersection of the (linear) *induced constraints* defined above describes a convex set $\hat{\mathcal{A}}(\boldsymbol{A})$ of admissible perturbed message-passing matrices $\hat{\boldsymbol{A}}$.

$$\begin{aligned}\hat{\mathcal{A}}(\boldsymbol{A}) = \big\{ \hat{\boldsymbol{A}} \in [0,1]^{N\times N} \mid \hat{\boldsymbol{A}} = \hat{\boldsymbol{A}}^T \wedge\ & \forall i,j : \boldsymbol{L}_{ij} \leq \hat{\boldsymbol{A}}_{ij} \leq \boldsymbol{U}_{ij} \\ \wedge\ & \forall i : L^{\text{row}}_i \leq \sum_j \hat{\boldsymbol{A}}_{ij} \leq U^{\text{row}}_i \ \wedge \forall i : \sum_j |\hat{\boldsymbol{A}}_{ij} - \hat{\boldsymbol{A}}_{ij}| \leq \bar{U}^{\text{row}}_i \\ \wedge\ & \forall i : \sum_{j\neq i} \left[\hat{\boldsymbol{A}}_{ij} - \hat{\boldsymbol{A}}_{ij}\right]_- \leq \Delta^-_i \\ \wedge\ & \sum_{i\leq j} |\hat{\boldsymbol{A}}_{ij} - \hat{\boldsymbol{A}}_{ij}| \leq \bar{U}_{\text{glob}} \wedge \sum_{i<j} \left[\hat{\boldsymbol{A}}_{ij} - \hat{\boldsymbol{A}}_{ij}\right]_- \leq \Delta^-_{\text{glob}} \big\}\end{aligned} \tag{7.11}$$

By construction of the set $\hat{\mathcal{A}}(\boldsymbol{A})$ it holds:



(a) Standard branching as in [3]. **(b)** Improved branching for graph certificates.

Figure 7.1: Overview of the branching step in the first iteration on dimension j

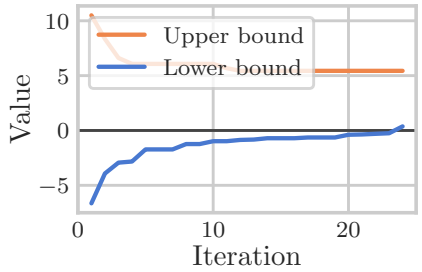


Figure 7.2: Converging lower and upper bounds; node selected from the CORA-ML dataset.

Theorem 6. *$\mathcal{T}(\mathcal{A}(\boldsymbol{A})) \subseteq \hat{\mathcal{A}}(\boldsymbol{A})$, i.e. for $\tilde{\boldsymbol{A}} \in \mathcal{A}(\boldsymbol{A})$ we have $\mathcal{T}(\tilde{\boldsymbol{A}}) \in \hat{\mathcal{A}}(\boldsymbol{A})$.*

From Theorem 6 it follows that the optimal solution of Eq. (7.7) leads to a valid lower bound on the original problem in Eq. (7.4).

7.3.3 Relaxation of the neural network

The above constraints can be computed efficiently and form a convex set, thus the constraint in Equation (7.7) can be efficiently handled. Still, however, the objective function, which is based on the original neural network f_θ, is challenging due to the nonlinearities (see **Challenge (1)** mentioned in the introduction). Here we follow the relaxation approach described in [234, 277]. Under this relaxation, the output $\boldsymbol{H}$ of the ReLU activation function in Eq. (7.1) is no longer deterministic but instead treated as a variable (like $\hat{\boldsymbol{A}}$) with the following constraints:

$$\begin{aligned} \boldsymbol{H}_{nj}^{(l)} \geq 0, \qquad \boldsymbol{H}_{nj}^{(l)} \geq \hat{\boldsymbol{H}}_{nj}^{(l)} \\ \boldsymbol{H}_{nj}^{(l)} \left(\boldsymbol{S}_{nj}^{(l)} - \boldsymbol{R}_{nj}^{(l)} \right) \leq \boldsymbol{S}_{nj}^{(l)} \left(\hat{\boldsymbol{H}}_{nj}^{(l)} - \boldsymbol{R}_{nj}^{(l)} \right) \quad \text{if } (n,j) \in \mathcal{I}^{(l)} \end{aligned} \tag{7.12}$$

Here, $\boldsymbol{S}_{nj}^{(l)}$ and $\boldsymbol{R}_{nj}^{(l)}$ are upper and lower bounds on the pre-ReLU activation $\hat{\boldsymbol{H}}_{nj}^{(l)}$ (which we describe shortly). $\mathcal{I}^{(l)}$ is the set of tuples (n, j) for which $\boldsymbol{S}_{nj}^{(l)}$ and $\boldsymbol{R}_{nj}^{(l)}$ have different signs. Analogously, $\mathcal{I}_+^{(l)}$ and $\mathcal{I}_-^{(l)}$ respectively consist of tuples (n, j) for which the lower and upper bounds are both positive / negative. For $\mathcal{I}_+^{(l)}$ and $\mathcal{I}_-^{(l)}$ we have the following

constraints on $\boldsymbol{H}_{nj}^{(l)}$:

$$\boldsymbol{H}_{nj}^{(l)} = \hat{\boldsymbol{H}}_{nj}^{(l)} \text{ if } (n,j) \in \mathcal{I}_+^{(l)} \qquad \boldsymbol{H}_{nj}^{(l)} = 0 \text{ if } (n,j) \in \mathcal{I}_-^{(l)} \tag{7.13}$$

We denote the set of hidden activations compliant with the above constraints as $\mathcal{Z}(\boldsymbol{A})$.

Computation of $\boldsymbol{S}_{nj}^{(l)}$ and $\boldsymbol{R}_{nj}^{(l)}$ For the convex relaxation described above we need valid lower and upper bounds on the pre-ReLU activations in the graph neural network. In contrast to [277], who make heavy use of the given (static) graph structure, in our case the graph structure itself changes. Instead we use the following upper and lower bounds (inspired by [189] for standard neural networks):

$$\begin{aligned}
\boldsymbol{S}^{(l)} &= \boldsymbol{U}\left[\boldsymbol{S}^{(l-1)}\boldsymbol{W}^{(l-1)}\right]_+ - \boldsymbol{L}\left[\boldsymbol{R}^{(l-1)}\boldsymbol{W}^{(l-1)}\right]_- + \boldsymbol{b}^{(l-1)} \\
\boldsymbol{R}^{(l)} &= \boldsymbol{L}\left[\boldsymbol{R}^{(l-1)}\boldsymbol{W}^{(l-1)}\right]_+ - \boldsymbol{U}\left[\boldsymbol{S}^{(l-1)}\boldsymbol{W}^{(l-1)}\right]_- + \boldsymbol{b}^{(l-1)} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{for } 2 \leq l \leq L
\end{aligned}$$

where $\boldsymbol{S}^{(1)} = \boldsymbol{R}^{(1)} = \boldsymbol{X}$ and $\boldsymbol{U}$, $\boldsymbol{L}$ are the element-wise lower and upper bounds on the message passing matrix (see Sec. 7.3.2.1).

7.3.4 Jointly Constrained Bilinear Program

Using the relaxation from Section 7.3.3 and the constraints on the message passing matrix from Section 7.3.2 we can rephrase the objective function in Eq. 7.7 as

$$\hat{\boldsymbol{H}}_{ty^*}^{(L)} - \hat{\boldsymbol{H}}_{ty}^{(L)} = \hat{\boldsymbol{H}}_{t}^{(L)}\boldsymbol{c} = \ \hat{\boldsymbol{A}}_t\boldsymbol{H}^{(L-1)}\boldsymbol{W}^{(L-1)}\boldsymbol{c} + \boldsymbol{b}^{(L-1)^\top}\boldsymbol{c}$$

where $\boldsymbol{c} = \boldsymbol{e}_{y^*} - \boldsymbol{e}_y$ is a K-dimensional constant vector with value 1 at y^*, -1 at y, and 0 else. Since $\boldsymbol{b}^{(L-1)^\top}\boldsymbol{c}$ is constant this leads to the overall problem:

$$\begin{aligned}
\tilde{m}^t(y^*, y) := \ &\underset{\hat{\boldsymbol{A}}, \boldsymbol{H}^{(\cdot)}}{\text{minimize}} \ \hat{\boldsymbol{A}}_t\boldsymbol{H}^{(L-1)}\boldsymbol{W}^{(L-1)}\boldsymbol{c} \\
&\text{subject to } \boldsymbol{H}^{(\cdot)} \in \mathcal{Z}(\boldsymbol{A}),\ \hat{\boldsymbol{A}} \in \hat{\mathcal{A}}(\boldsymbol{A})
\end{aligned}$$

Note that in the objective function we have the term $\hat{\boldsymbol{A}}_t\boldsymbol{H}^{(L-1)}$ and thus we have a bilinear objective function.

7.3.4.1 Canonical form

We will bring the above problem in a canonical form to simplify further analysis. From now on we consider the special case of a GCN with two message passing steps (i.e., $L = 3$), and subsequently show how to generalize to $L > 3$. First, as a shorthand, we define the vector $\boldsymbol{w} = \boldsymbol{W}^{(2)}\boldsymbol{c} \in \mathbb{R}^{h^{(2)}}$ and the matrix

$$\boldsymbol{P} \in \mathbb{R}^{|\mathcal{N}_t| \times |\mathcal{N}_t| \cdot h^{(2)}} = \begin{bmatrix} \boldsymbol{w}^T & \boldsymbol{0}^T & \cdots & \boldsymbol{0}^T \\ \boldsymbol{0}^T & \boldsymbol{w}^T & \cdots & \boldsymbol{0}^T \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0}^T & \cdots & \cdots & \boldsymbol{w}^T \end{bmatrix},$$

where $\mathbf{0}$ is a $h^{(2)}$-dimensional vector of zeros. Next, we denote with $\boldsymbol{H}_{\text{vec}}^{(2)} \in \mathbb{R}^{|\mathcal{N}_t| \cdot h^{(2)}}$ the result of flattening the matrix $\boldsymbol{H}^{(2)}$ into a vector.[1] Finally, we introduce another variable $z \in \mathbb{R}^{|\mathcal{N}_t|} = \boldsymbol{P}\boldsymbol{H}_{\text{vec}}^{(2)}$. With this, the above problem can equivalently be written as

$$\begin{aligned} &\underset{\hat{\boldsymbol{A}}, \boldsymbol{z}, \boldsymbol{H}^{(2)}}{\text{minimize}} \ \hat{\boldsymbol{A}}_t \boldsymbol{z} \\ &\text{subject to } \boldsymbol{H}^{(2)} \in \mathcal{Z}(\boldsymbol{A}),\ \hat{\boldsymbol{A}} \in \hat{\mathcal{A}}(\boldsymbol{A}),\ z = \boldsymbol{P}\boldsymbol{H}_{\text{vec}}^{(2)} \end{aligned} \tag{7.14}$$

Note that all constraints are linear since the constraints on $\boldsymbol{H}^{(2)}$ (see Eqs. (7.12, 7.13)) relate to the value of $\hat{\boldsymbol{H}}^{(2)} = \hat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}$ (see Eq. (7.2), recall that $\boldsymbol{H}^{(1)} = \boldsymbol{X}$), which is linear in $\hat{\boldsymbol{A}}$. Furthermore, we can easily add the following constraint to the problem without changing its solution: $(\hat{\boldsymbol{A}}_t, \boldsymbol{z}) \in \Omega$ where

$$\Omega = \{(\hat{\boldsymbol{A}}_t, \boldsymbol{z}) : \boldsymbol{L}_{tj} \leq \hat{\boldsymbol{A}}_{tj} \leq \boldsymbol{U}_{tj}, \boldsymbol{m}_j \leq \boldsymbol{z}_j \leq \boldsymbol{M}_j; 1 \leq j \leq |\mathcal{N}_t|\} \tag{7.15}$$

forms a simple rectangular constraint on the variables (see Figure 7.1a). Here, $\boldsymbol{L}$ and $\boldsymbol{U}$ are the familiar entry-wise box constraints on $\hat{\boldsymbol{A}}$, and $\boldsymbol{m}$ and $\boldsymbol{M}$ are bounds on $\boldsymbol{z}$ which, e.g., can be obtained via standard interval arithmetic from the bounds on $\boldsymbol{H}^{(2)}$:

$$\boldsymbol{m} = \left[\boldsymbol{R}^{(2)}\right]_+ [\boldsymbol{w}]_+ - \left[\boldsymbol{S}^{(2)}\right]_+ [\boldsymbol{w}]_-, \quad \boldsymbol{M} = \left[\boldsymbol{S}^{(2)}\right]_+ [\boldsymbol{w}]_+ - \left[\boldsymbol{R}^{(2)}\right]_+ [\boldsymbol{w}]_-$$

So far, it seems that this new constraint adds no benefit – but it indeed now opens the door for solving the problem: Using the above reformulation, we can apply the principle proposed in [3] for solving so-called jointly constrained bilinear optimization problems.

7.3.4.2 Branch-and-bound algorithm

In the following we give an overview of the branch-and-bound algorithm we employ to solve the bilinear program in Eq. (7.14). We refer the interested reader to [3] for more details on the procedure and convergence proofs.

The idea is twofold: (1) We use the convex envelope[2] of the objective function to compute *lower bounds* on a rectangular domain over $\hat{\boldsymbol{A}}_i$ and $\boldsymbol{z}$. That is, instead of minimizing the objective in Eq. (7.14) we are minimizing

$$\sum_{j=1}^{|\mathcal{N}_t|} \max\{\boldsymbol{m}_j \hat{\boldsymbol{A}}_{tj} + \boldsymbol{L}_{tj}\boldsymbol{z}_j - \boldsymbol{L}_{tj}\boldsymbol{m}_j, \boldsymbol{M}_j \hat{\boldsymbol{A}}_{tj} + \boldsymbol{U}_{tj}\boldsymbol{z}_j - \boldsymbol{U}_{tj}\boldsymbol{M}_j\},$$

[1] We slightly abuse the notation here. $\boldsymbol{H}$ actually has a shape of $\mathbb{R}^{N \times h^{(2)}}$. For the objective function, however, we only need the elements from N_t, i.e. we can slice $\boldsymbol{H}$ to a smaller matrix. Equivalently we can slice $\hat{\boldsymbol{A}}_t$ to have shape $\mathbb{R}^{|\mathcal{N}_t|}$. This step also leads to more efficient computation. Indeed, we can start our overall procedure by slicing $\boldsymbol{A}$ into shape $\mathcal{N}_t^{(2)} \times \mathcal{N}_t^{(2)}$, where $\mathcal{N}_t^{(2)}$ are the two-hop neighbors of t, since t's prediction does not depend on any other nodes for $L = 3$.

[2] The convex envelope of a function f on a domain Ω is the pointwise supremum of all convex functions that underestimate f over Ω [3]. It can be efficiently computed for the scalar product of two d-dimensional vectors as in our objective function.

Dataset	Q	q	% Cert. Robust	% Cert. Nonrobust	% Certifiable
Citeseer	1	1	88.4	7.2	95.6
	5	3	78.8	11.8	90.6
	10	5	73.0	12.8	85.8
Cora-ML	1	1	80.8	11.6	92.4
	5	3	56.2	21.2	77.4
	10	5	43.8	29.0	72.8
PubMed	1	1	78.8	9.2	88.0
	5	3	58.2	14.8	73.0
	10	5	49.0	18.4	67.4

(a) Certification results for standard GCN.

Dataset	Q	q	% Cert. Robust	% Cert. Nonrobust	% Certifiable
Citeseer	1	1	96.2	2.6	98.8
	5	3	83.4	7.4	90.8
	10	5	76.4	10.4	86.8
Cora-ML	1	1	87.8	5.6	93.4
	5	3	57.2	15.2	72.4
	10	5	47.0	22.6	69.6
PubMed	1	1	89.4	4.8	94.2
	5	3	72.8	10.6	83.4
	10	5	63.2	14.4	77.6

(b) Results for GCN with robust training proposed in [277].

Table 7.1: Robustness certification results. Our method can certify a large percentage of the nodes.

When using this objective function in Eq. (7.14), we (i) obtain a convex problem, more precisely even an LP, which is extremely efficient to solve; and (ii) we obtain a *lower bound* v on the global optimum v^* of the original problem on the hyperrectangle Ω.

(2) This brings us to the second idea: We use a branch-and-bound procedure to recursively subdivide the rectangular domain. (a) We initialize the branch-and-bound procedure (i.e., $k = 1$) with the initial hyperrectangle $\Omega = \Omega^{(1)}$ and solve the problem using an off-the-shelf LP solver to obtain a lower bound $v^{(1)}$ (along with the optimal solution $(\hat{\boldsymbol{A}}_t^{(1)}, \boldsymbol{z}^{(1)})$). The tuple $(\Omega^{(1)}, v^{(1)})$ is added to the list $\mathcal{L}$ of (yet to be branched) rectangles.

(b) In the branching step, i.e., in iteration k, we select the tuple from $\mathcal{L}$ with the smallest lower bound for branching (recall that our goal is to solve a minimization problem). The branching is illustrated in Figure 7.1. We choose a dimension j to split (see Appendix E.1 on how we choose j) and divide 'around' the optimal solution that was obtained on the overall rectangle. Figure 7.1 illustrates how we divide $\Omega_j^{(1)}$ into the subregions $\Omega_j^{(2,1)}, \Omega_j^{(2,2)}, \Omega_j^{(2,3)}$, and $\Omega_j^{(2,4)}$ where the optimal solution $(\hat{\boldsymbol{A}}_{tj}^{(1)}, \boldsymbol{z}_j^{(1)})$ is indicated by the black dot. We solve these 4 problems, obtain 4 new tuples, and add them to $\mathcal{L}$. Since dividing the domain Ω into smaller sub-rectangles leads to tighter convex envelopes, we have $v^{(k+1)} \geq v^{(k)}$ (see Fig. 7.2), so that for $k \to \infty$ we recover the global solution. While (theoretically) this might require infinitely many iterations, in practice this is not a concern as we will discuss next along with further improvement.

Moreover note that with the above procedure we also easily get upper bounds as a by-product. Every time we solve an LP we use its optimal solution $(\hat{\boldsymbol{A}}_t^{(k)}, \boldsymbol{z}^{(k)})$ and plug it into the original Eq. (7.14). This leads to an upper bound. The upper bound $V^{(k)}$ in iteration k is then the minimum of all upper bounds obtained so far.

7.3.4.3 Improvements for Graph Certificates

It is crucial to observe that we are actually not interested in finding the global solution of Eq. (7.14), but we are only interested in its sign (positive or negative). Since in each iteration of the above approach we obtain increasingly accurate upper and lower bounds on the global solution, we can perform an early stopping. We can stop (i) if the lower bound is positive, $v^{(k)} > 0$. In this case we have successfully certified robustness. We can also stop (ii) if the upper bound is negative, $V^{(k)} < 0$. In this case we are certain that we cannot certify robustness for this instance. Note that (ii) *does not* imply that the classifier is not robust; it only means that no robustness guarantee can be made. In Fig. 7.2 we show a real-world example of this early stopping.

We further improve the procedure by exploiting knowledge about the graph domain, which we visualize in Fig. 7.1b. More precisely, when we branch on $\hat{\boldsymbol{A}}_{tj}$ and we find that the parent problem's optimal value for $\hat{\boldsymbol{A}}_{tj}$ is *smaller* than the original value $\mathcal{T}(\boldsymbol{A})_{tj}$, we set both the upper and lower bounds on $\hat{\boldsymbol{A}}_{tj}$ to zero for the sub-rectangles $\Omega_j^{(k+1,1)}$ and $\Omega_j^{(k+1,4)}$. This reflects the fact that we can only decrease values in $\mathcal{T}(\boldsymbol{A})$ by *removing* the corresponding edge (see element-wise bounds in Sec. 7.3.2.1). While we cannot encode this efficiently into the original bilinear problem, we can exploit this fact during the branch step. Note that the two sub-rectangles collapse into *lines* on the border of $\Omega_j^{(k)}$. Since the convex envelope is exact on the border of the rectangle, this not only leads to faster optimization but also tighter lower bounds on the original problem's global optimum.

7.3.4.4 Generalizing to $L > 3$

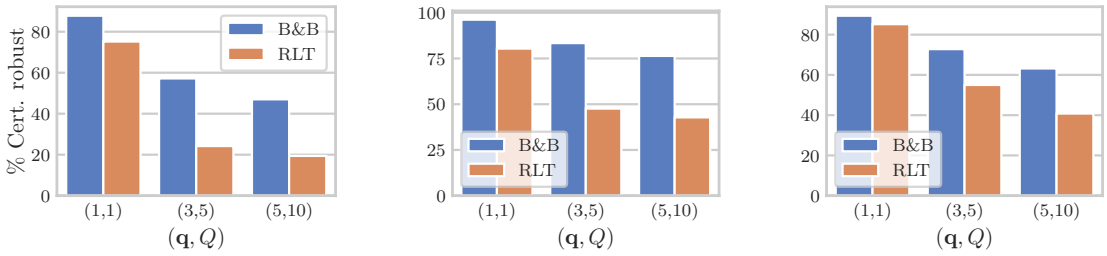
Our robustness certificates can be generalized to arbitrary number of message-passing steps. To this end we first observe that for $2 < l < L$ the constraints in the ReLU convex relaxation are no longer linear. For instance in the constraint

$$\boldsymbol{H}^{(l)} \geq \hat{\boldsymbol{H}}^{(l)} = \hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l-1)} + \boldsymbol{b}^{(l-1)}$$

we have the bilinear term $\hat{\boldsymbol{A}}\boldsymbol{H}^{(l-1)}$, which is not convex. One solution is to use the reformulation-linearization technique Qualizza et al. [187] to define a new variable $\tilde{\boldsymbol{H}}^{(l)}$ and use the entry-wise lower and upper bounds on $\hat{\boldsymbol{A}}$ and $\boldsymbol{H}^{(l-1)}$ to derive upper and lower bounds on $\tilde{\boldsymbol{H}}^{(l)}$ using interval arithmetic. This leads to the constraints being linear again and we can apply our improved branch and bound procedure from above.

7.3.4.5 Alternative solution approaches

A bilinear program (BLP) is a special case of a quadratic program (QP). However, expressing a BLP as a QP leads to a matrix $\boldsymbol{Q}$ with zero diagonal, hence cannot be positive semidefinite and therefore the QP is not convex. Semidefinite relaxations such as in [189], where $\boldsymbol{Q}$ is replaced by a psd matrix, do not apply either due to the zero diagonal on $\boldsymbol{Q}$.



(a) Cora-ML **(b)** Citeseer **(c)** PubMed

Figure 7.3: Comparison of our branch-and-bound (B&B) algorithm with a reformulation-linearization (RLT) baseline.

Dataset:	Cora-ML					Citeseer					PubMed				
Rob. Training	None	Robust GCN	ED	Atk.	B&B	None	Robust GCN	ED	Atk.	B&B	None	Robust GCN	ED	Atk.	B&B
% Cert. robust	56.2	**57.2**	55.4	55.8	56.6	78.8	**83.4**	78.0	79.2	79.8	58.2	**72.8**	60.4	61.2	60.4

Table 7.2: Results for employing various training schemes: standard training, robust training as in [277], ED (edge dropout with $p = 0.2$), ED with edges found by adversarial attacks, ED with edges found by B&B. $(\boldsymbol{q}, Q) = (3, 5)$.

Another relaxation approach for nonconvex QPs, resp. BLPs, is the reformulation-linearization technique (RLT) [187]. Since RLT performs a relaxation on the original BLP, the result serves as a lower bound on the BLP's optimal value. We derived the RLT for our problem and compare with this alternative in our experiments.

Summary We have rephrased our problem of certifying robustness of GCN under perturbation of the graph structure as a jointly constrained bilinear program (see Eq. (7.14)). While the BLP is not convex, we can use the branch-and-bound framework to get increasingly accurate lower and upper bounds on the global optimum. Our improved branch-and-bound algorithm, combined with early stopping when either the upper or lower bound crosses zero, leads to an efficient procedure for robustness certification.

7.4 Experiments

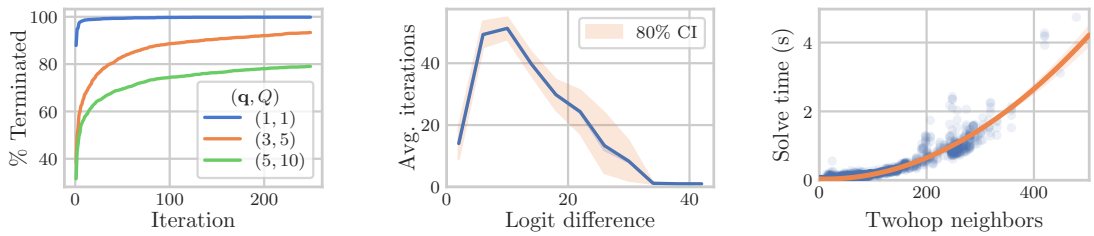
We evaluate our robustness certification method on the publicly available and widely used datasets Citeseer[201], Cora-ML[152], and PubMed[201]. See Table A.1 for more information on the datasets. On each dataset we train a GCN with two message-passing steps (i.e. $L = 3$) with hidden dimension 32, using 10% of the labels during training. We provide further details on our experimental setup and hyperparameters in App. E.2.

Robustness certificates We first present our results on our core contribution: the first method for certified robustness of GCN under perturbations of the graph structure. To evaluate our method on different severities of perturbations we use three different local ($\boldsymbol{q}$) and global (Q) perturbation budgets ($\boldsymbol{q}, Q$): $(1, 1)$, $(3, 5)$, and $(5, 10)$. Recall that our method can certify robustness, but not *non-robustness*. Obtaining the true number of non-robust nodes requires a brute-force search over the set of all possible perturbations and is therefore intractable for all above cases except $(1, 1)$. However, we obtain an estimate (lower bound) on the number of non-robust nodes by also performing gradient-based adversarial attacks with these budgets. We therefore report the share of nodes certified robust by our method, the share of nodes certified *non-robust* by the adversarial attack, and the total share of certified nodes, which is the sum of the former two. The gap to 100% are nodes for which no certificate can be given.

In Table 7.1a we present our results on standard GCN. We see the general trend that increasing the perturbation budget leads to the fraction of certifiably robust nodes decreasing, while on the other hand the share of non-robust nodes increases. Somewhat surprisingly, on all datasets, more than 5% of the nodes can change their predicted class label even when only a single edge removed, indicating that standard GCN is highly nonrobust. For the perturbation budget of $(3, 5)$, we can certify as robust more than half of the nodes on each of the datasets, while the share of nonrobust nodes increases significantly.

In Table 7.1b we see the results of certifying GCN trained for robustness against *attribute* perturbations proposed in [277]. A striking difference is that on every dataset and every budget, we have more certifiably robust nodes and less nonrobust nodes compared to standard GCN. For PubMed and a budget $(5, 10)$ we can even certify more than 25% (rel.) additional nodes as robust while having more than 20% (rel.) fewer nonrobust nodes. This result is remarkable since it suggests that robust training on a different objective (robustness to attribute perturbations) also has a beneficial effect on the robustness w.r.t. graph structure perturbations. Our method is the first that enables us to draw such conclusions based on robustness certification. Observe the high share of overall certifiable nodes; thus, for the vast majority of nodes we have a clear decision (robust/not robust). The cases in which no decision can be made can either be due to the relaxation in the certification method *or* the fact that we only have a heuristic adversarial attack (except for the budget $(1, 1)$, where we compute the exact numbers by exhaustive search). On Cora-ML we also analyzed a budget of $(10, 20)$: in such a setting still around 41% of the nodes can be certified as robust.

Comparison to linear relaxation We further compare our branch-and-bound (B&B) algorithm to a relaxation via reformulation-linearization (RLT) of the bilinear program (see [187] for details on the relaxation). While the latter only requires solving a single linear program, its optimal solution is necessarily a lower bound on our solution obtained via B&B. In Fig. 7.3 we compare the share of successful robustness certifications for our B&B algorithm and the RLT relaxation. As expected, on every dataset and for every budget our B&B approach outperforms the baseline. Even more, the gap widens



(a) Termination rate of B&B.

(b) Required iterations vs. logit difference.

(c) Solve time vs. number of twohop neighbors.

Figure 7.4: Analysis of the branch-and-bound procedure on the CORA-ML dataset. For (b) and (c) we have $(\boldsymbol{q}, Q) = (3, 5)$.

for increasing budgets, e.g. for CORA-ML and budgets $(3, 5)$ and $(5, 10)$ our method certifies more than twice as many nodes as the baseline. In conclusion, we find that our proposed B&B algorithm has substantial advantages over the linear relaxation of the bilinear program.

Other types of robust training Adversarial training, i.e. including adversarially perturbed examples into the training, is a standard defense against attacks [88]. In our setting, this corresponds to randomly dropping edges during training. To evaluate whether this leads to improved robustness we test a range of adversarial training techniques in Table 7.2. ED corresponds to edge dropout, i.e. randomly dropping edges (with $p = 0.2$) at each training iterations. Atk. is similar to ED but we drop out edges proportional to how often they were removed by adversarial attacks. In B&B we drop out edges proportional to how often they were set to zero in the optimal solutions obtained by B&B. We see that none of the robust training procedures consistently improve upon standard training, except the robust training for robustness w.r.t. feature perturbations proposed in [277]. Hence, we conclude that standard adversarial training does not lead to higher robustness. Note that this finding is in line with [56], who also observe only a minimal positive effect on robustness via edge dropout. Future work could explore training methods making direct use of the robustness goal (i.e. similar to [277]); however due to the branch-and-bound procedure for obtaining certificates, this approach is much more challenging.

Analysis of B&B In Fig. 7.4 we present insights into the optimization procedure of our branch-and-bound algorithm. In Fig. 7.4a we can see that most instances terminate within the first few iterations of our branch-and-bound procedure (note that the y-axis starts at 40%; we force termination after 250 iterations). Recall that our algorithm terminates at iteration k if either the lower bound $v^{(k)}$ is positive (recall Fig. 7.2 for an example), leading to a robustness certificate, or that the upper bound $V^{(k)}$ is negative, meaning that we are certain that no certificate can be given. Furthermore, for increasing budgets, our procedure tends to require more iterations to converge. This can be ex-

plained by the fact that the domain Ω becomes larger for increasing budgets. Therefore we need more subdivision steps to achieve the accuracy required to make a decision.

Fig. 7.4b shows the relationship between the logit difference before the attack and the average number of iterations until convergence. Recall that the logit difference is the difference of the largest log-probability assigned by the classifier minus the second-largest log-probability. Therefore it indicates how confidently a node is classified in its current class. In the figure we can see that for nodes with a very large logit difference converge after very few iterations. This is because they are so confidently classified that our method terminates very quickly. Similarly, nodes that have a small logit difference (i.e., less confidently classified) require, on average, relatively few steps until termination. An explanation is that it is easy to change these nodes' classification. Nodes with a medium logit difference require the most iterations (on average), since for thes instances neither the lower nor upper bounds change their sign early in the procedure. Still, even for these cases, the average number of iterations to terminate is only around 50, meaning that our overall algorithm is very efficient.

In Fig. 7.4c we see how the time required to solve one problem instance (i.e., one iteration in our B&B procedure) relates to the number of twohop neighbors (i.e., nodes reachable within two or less hops) of the target node. Note that the median time required for solving is about 75 ms, and less than 5% of instances require more than one second to solve. The quadratic trend comes from the fact that the off-the-shelf LP solver we use does not fully support *sparse* variable matrices as we have in our problems – with full sparse support the solve time scales linearly in the number of edges in the twohop neighborhood. Even with this setup, the median total time until convergence is well below one second (ca. 820 ms), hence our procedure can be used to efficiently certify large numbers of nodes.

7.5 Conclusion

In this work we present the first method for certifiable robustness on GCN under structure perturbations. We show how to frame this problem as a jointly-constrained bilinear program, and propose a branch-and-bound procedure that makes use of knowledge about the graph domain. Our procedure decomposes the original problem into sub-problems, which are in turn linear programs and can be solved using highly optimized off-the-shelf solvers. Our certification method outperforms a reformulation-linearization baseline by a large margin and is able to certify a large fraction of the nodes.

Retrospective

The approach in this chapter is a natural extension of the attribute robustness certificates presented in Chapter 6. However, because of the products of the decision variables (the entries in the preprocessed adjacency matrix) the optimization problem is non-convex, making this setting much more challenging to handle. This leads to some limitations of the approach. First, we can only handle edges that were inserted by a potential adversary.

Extending the work to include edges removed by an adversary is very challenging because we cannot limit the optimization problem to the L-hop neighborhood of the node in question. This is similar to the limitation of Bojchevski et al. [27], where they assume that there are only edges which were *added* by some data corruptions and do not handle potentially dropped edges, and is further in line with the limitation of the later work by Schuchardt et al. [198]. Another limitation of our work is that because of the series of relaxations and carefully chosen constraints on the optimization problem, the approach is specific to the GCN model and cannot easily be extended or modified to other GNN architectures. Finally, since we need to solve multiple linear programs in the customized branch-and-bound scheme, there is no obvious way to integrate the certificates into the training procedure in an end-to-end fashion similar to the approach in Chapter 6.

8 Improving GNN Robustness: Retrospective

To reduce the potential harm of adversarial attacks on GNNs, researchers have proposed a wide range of methods aimed at improving models' robustness in recent years. Here, we group them broadly following the categorization from Section 2.4.2.

8.1 Heuristic methods

Heuristic methods attempt to improve the empirical robustness of ML models but do not come with mathematical or statistical guarantees of robustness. Still, we can use empirical robustness against adversarial attacks as well as model-agnostic provable robustness methods such as randomized smoothing to evaluate upper and lower bounds on the model robustness, respectively.

We can further split heuristic methods for GNN robustness into two main groups: **structure-cleaning approaches** and **robust GNN** models. We further summarize findings of applying **robust training methods** such as adversarial training to GNNs below.

Structure-cleaning approaches process the (potentially perturbed) graph with the goal of removing or mitigating the effect of adversarially inserted or removed edges. Entezari et al. [69] observe that Nettack's perturbations mostly affect the high-rank spectrum of a graph. As a defense, they preprocess the graph via the singular value decomposition (SVD) and only retain some small number of low-rank components. This leads to drastically increased robustness against Nettack's local attacks. On global attacks, however, Geisler et al. [83] report much smaller empirical robustness gains and even reduced provable robustness with the SVD preprocessing. This highlights a major drawback of empirical robustness methods: they may perform well against existing strong attacks, but can fail against attacks that were not considered in the design or that were later developed. Moreover, a limitation of the SVD preprocessing is that the resulting graph is not sparse, i.e., needs $O(N^2)$ memory. This leads to the SVD defense not scaling to graphs much larger than $N = 20,000$ nodes on commodity GPUs.

Wu et al. [237] propose a method that removes edges where the nodes' feature vectors are very dissimilar as measured by the Jaccard index (also known as intersection over union (IoU)). They argue that these edges are likely to stem from an attacker, as in homophilic datasets connected nodes tend to be similar. Their defense increases empirical robustness against local gradient-based attacks and Nettack. In the study of Geisler

et al. [83], robustness gains for global attacks and provable robustness are relatively minor.

Xu et al. [249] propose a structure-cleaning approach that uses bi-level optimization. Their method is effectively a reverse version of the meta-learning-based attacks we present in Chapter 4. A key difference is that they average the (approximate) meta-gradients over k different training/validation splits of the labeled data. Further, they propose a more efficient variant where the graph structure modification has low-rank structure. The authors report substantial robustness gains against global poisoning attacks by our meta-gradient attack. However, the authors did not study *adaptive* attacks which also take into account their specific defense. Further, the authors show that their method can be used to improve node classification results.

Elinas et al. [67] propose structure learning on GNNs via variational inference. They learn one parameter for each entry in the adjacency matrix, that is, each pair of nodes. The original graph serves as the prior over the adjacency matrix. The likelihood of the model is the expected cross-entropy classification loss. The authors report improved robustness to the poisoning attacks by Bojchevski and Günnemann [23]. Because they explicitly parameterize each adjacency matrix entry, their method scales in $O(N^2)$ and can therefore not scale to large graphs.

Zhang and Zitnik [260]'s GNNGuard method prunes edges between nodes with dissimilar features or representations and increases the weight between similar nodes. In contrast to the Jaccard-based defense above, the method is applied to each layer of the GNN, so effectively we have a different graph structure *per layer* of the GNN. Thus, this method not purely preprocessing-based as it can update the graph structure also at test time. They report strong robustness improvements against global poisoning attacks by our proposed meta-gradient attack.

Jin et al. [111] propose Pro-GNN, a method that learns a 'cleaned' adjacency matrix along with the GNN parameters during training. Their loss function encourages sparsity and low rank of the learned adjacency matrix as well as feature smoothness, i.e., similar features of connected nodes. Thus, this method combines the findings by Entezari et al. [69]'s SVD low-rank preprocessing as well as Wu et al. [237]'s Jaccard feature-smoothness preprocessing. A major drawback is scalability, as the method learns a dense $N \times N$ adjacency matrix. The method further has some resemblance of works refining or learning from scratch an adjacency matrix with the goal of improving some downstream prediction task such as node classification [78, 47, 255], time series forecasting [115, 4, 241, 202], or anomaly detection [60].

Robust GNNs. Another stream of work aims to improve GNN robustness by modifying the GNN models themselves, e.g., by modifying the neighborhood aggregation function of a GNN. Zhu et al. [268] propose a robust GCN variant which parameterize nodes' representations as a Gaussian distribution with a mean vector and diagonal covariance matrix. In their aggregation function, they reduce the weight of neighbors with high variance, arguing that high variance corresponds to high uncertainty assigned by the model. They report improved robustness against local attacks by Nettack or RL-S2V

[56]. Later studies [260, 83] report results on global attacks, where this robust GCN variant did not lead to substantial gains in empirical or provable robustness. Wu et al. [239] propose the Graph Information Bottleneck (GIB), an instantiation of the Information Bottleneck (IB) [217] principle for graphs. The authors report that a GNN based on the GIB principle is more robust to local evasion and poisoning attacks by Nettack. Zheng et al. [266] report improved robustness on some dataset-model combinations by introducing layer normalization [10].

Geisler et al. [83] propose a robust aggregation function called Soft Medoid which attempts to filter outliers. The argument is that the most severe attacks insert edges to a graph, leading to additional neighbors in the nearby nodes' message passing aggregation. They observe that these adversarially inserted neighbors tend to have very different features or latent representations than the 'clean' neighbors (corroborating findings reported by earlier works mentioned above). Hence, they propose an aggregation function inspired by robust statistics, which aims to reduce the effect of outliers. The main drawback of their aggregation function is that it scales quadratically with the number of neighbors of a node. They address this shortcoming in a follow-up work [81], where they propose the Soft Median aggregation function which scales linearly in the number of neighbors of a node. Interestingly, for both the Soft Medoid and the Soft Median, the authors observe no substantial robustness improvement based on the robust aggregation functions alone. Only after adding the GDC preprocessing proposed by Klicpera et al. [122] they report strong empirical and provable robustness gains on global and local evasion attacks. They argue that since many nodes have a very low degree in real-world power law graphs, a robust aggregation function is unable to detect and filter outliers for these nodes. The GDC preprocessing acts as a low-pass filter densifying connections to low-degree nodes, enabling robustness gains from the proposed aggregation functions. Further, the authors report that the GDC preprocessing alone does not yield meaningful robustness improvements.

Robust training is a heuristic approach that aims to find more robust models by modifying the training procedure, e.g., by optimizing some alternative loss function that encourages robustness. The most well-known example is adversarial training (see Section 2.4.2), where at each training step we first construct an adversarial example via some attack method and then optimize the loss on this perturbed example instead. This procedure has shown consistent and meaningful robustness improvements for image classification, so a natural idea is to generalize it to GNNs.

Xu et al. [245, 247] investigate adversarial training and report increased robustness. However, as Pfeifle [183] studied in his thesis, the improved robustness does overwhelmingly not stem from the robust training procedure. Instead, the self-training component, which is part of their robust training procedure, is responsible for virtually all robustness improvement. In self-training, we treat the predictions of the trained classifier as ground-truth labels and continue training on these.

In our work [278] presented in Chapter 7 we investigate several robust training schemes: randomly dropping edges during training; removing edges based on adversarial attacks;

removing edges based on the certification procedure; and training with the robust training procedure for improved feature robustness [277] introduced in Chapter 6. We found no improved robustness for all except the last robust training procedure and conclude that, somewhat surprisingly, the robust training procedure for *feature robustness* improves resilience to *structure* attacks. However, similar to Xu et al. [245]'s method, this training procedure also includes a self-training phase. In light of the findings by Pfeifle [183] that self-training itself can substantially improve robustness, an interesting follow-up study would be to find out how the robustness gains we report in Chapters 6 and 7 stem from self-training and the robust training itself, respectively.

In their work on randomized smoothing for discrete data (such as graphs), Bojchevski et al. [25] also experiment with robust training. Here, they add random perturbations to the graph structure during training with the goal of obtaining a model which is more robust to the perturbations of randomized smoothing. This would lead to higher certified robustness of the trained model. However, the authors report that this robust training procedure did not improve certified robustness in a meaningful way.

A counterexample to this general struggle to find effective structure-based robust training techniques for GNNs is the work of Tang et al. [213]. They study the task of graph classification and report improved robustness stemming from adversarial training of the model. An important difference between [213] and the other ones mentioned above is that for graph classification the individual samples are *independent*, i.e., do not influence each other. Thus, from this perspective, graph classification is more similar to tasks such as image classification, and this might explain why adversarial training can improve robustness here.

In their GNN robustness benchmark Zheng et al. [266], the authors report improved robustness on some dataset and model combinations for adversarial training. Here, the adversarial training consists of injecting adversarial nodes into the graph at each epoch. Studying these instances and their commonalities could lead to a better understanding of robust training for GNNs.

In summary, robust training for GNNs remains largely unsolved. This is intriguing since for images adversarial training is a simple and effective heuristic method leading to substantial gains in certified and empirical robustness. Developing methods or GNN models for which robust training is effective is therefore a very important and potentially fruitful direction for future research.

8.2 Provable robustness

While most of the proposed approaches improving GNN robustness are of heuristic nature, there has also been progress in provable robustness for GNNs besides the robustness certificates based on convex relaxation for attribute and structure robustness presented in Chapters 6 and 7.

Wang et al. [231] propose a robust aggregation function, similar in spirit to the ones presented in [83, 81, 260], which reduces the edge strength at each layer between nodes with dissimilar representations. The authors exploit the Lipschitz smoothness [232]

of the resulting GNN to derive upper bounds on the change in the output given a feature or structure perturbation. An interesting question for future studies could be to evaluate whether the models presented in [83, 81, 260] also admit tractable computation of lower bounds on the Lipschitz constants. This would allow us to compute robustness guarantees for the otherwise heuristic approaches presented in the mentioned studies.

Another stream of research considers randomized smoothing [53] and how to efficiently and effectively map it to the discrete graph domain. Jia et al. [108] propose randomized smoothing for community detection in graphs. However, their method does not exploit the sparsity present in real-world graphs and therefore lacks scalability. Bojchevski et al. [25] address this shortcoming by proposing sparsity-aware randomized smoothing for discrete data, which they also apply to graphs and GNNs. Their approach has been used by several follow-up studies [e.g., 83, 81] to evaluate the provable robustness of different models and defenses. Wang et al. [227] prove tightness of their randomized smoothing certificates for GNNs. This means that without further assumptions on the model, for any perturbation radius larger than the certified radius there exists at least one adversarial example.

Schuchardt et al. [198] propose *collective* certificates for adversarial attacks on graph. Their method assumes we have some *base* certificates *per node*, e.g., obtained from methods such as [277, 278, 25]. The important observation is that these base certificates hold individually per node. In practice, an adversary has to choose how to spend their budget to perturb individual nodes. The authors explicitly model this decision problem of the adversary to derive certificates saying that *on the whole graph*, at most K nodes' predictions can in the worst case be changed by the adversary with some perturbation limit. Thus, this method leverages *local* robustness certificates to prove robustness against *global* attacks. One limitation is that since the model exploits locality in GNN computations, their collective certificates do not work against adversarial attacks inserting edges to the graph.

Part IV

Conclusion

9 Conclusion

In this thesis we presented four studies of GNN robustness – two studies in Part II concerned with establishing that GNNs are nonrobust, and two further studies in Part III whose goal is to improve GNN robustness. In addition, we provide retrospective insight at the end of the respective parts and embed the studies into the broader research context.

In this chapter we conclude the thesis by highlighting properties beyond model robustness which are important for safe practical application of GNNs. We explore broader impact aspects of ML in general and GNNs in particular, and close this thesis by posing open research questions for future work.

9.1 Beyond robustness of GNNs

Robustness to noise and adversaries is ony one key property for ML models in practical applications. Two additional properties are **calibrated uncertainty** and **privacy**.

Calibrated uncertainty estimates enable us to ignore a model's prediction if it indicates high uncertainty. In other words: we know when *not* to trust a model's prediction, e.g., because it is presented a sample that does not match the training distribution. Uncertainty estimation for models dealing with i.i.d. data (e.g., images) is a very active field of research with many recent and traditional studies [e.g., 43, 42, 128, 20, 79, 142, 6, 148, 149].

There are also a number of recent works on uncertainty estimation for graph ML methods. We can group them into Bayesian neural network approaches [e.g., 261, 174, 175], graph Gaussian processes [e.g., 169, 30, 267, 144] and, most recently, models predicting the parameters of the conjugate prior of the target distribution (e.g., a Dirichlet distribution instead of a categorical distribution in the case of classification) [206, 265]. To date, these studies are orthogonal to the work on studying GNN robustness. Thus, an interesting opportunity for follow-up research is to study the robustness of uncertainty-based GNNs or whether uncertainty can be used to detect adversarial attacks. Such a study has recently been proposed for traditional i.i.d. data [124] with the finding that Dirichlet-based uncertainty estimation models are not robust.

Privacy is another key requirement for real-world use of machine learning. Privacy violations include that an attacker can infer facts about members of the population, about members of the training set, or about the model parameters [58]. A widely used method to prevent an attacker from gaining knowledge about (potentially sensitive)

attributes of members of the training set is differential privacy (DP) [64]. In DP, the goal is that the model produces very similar output when any individual sample is included in the training set or not. Thus, DP addresses the paradox of learning nothing about the individual training samples while learning useful information about the training dataset as a whole [65]. Hence, privacy of ML models has an inherent connection to adversarial robustness.

Privacy in ML for graphs is particularly relevant because of the interdependence of nodes. Similar to the influence attacks from Chapter 3, it is possible to infer (potentially sensitive) attributes about a node by knowing its neighbors' features. Ellers et al. [68] show that it is possible to recover a deleted nodes' neighbors, raising concerns that simply deleting nodes is not enough to erase their information. Other recent studies consider differentially private GNNs [172, 235, 137]. Studying the intersection of privacy and robustness of GNNs is a promising direction for future research.

9.2 Broader impact

As machine learning expands into more and more areas of professional and private life, it is important to consider potential unwanted side effects of the models we use. Most of these are shared between 'standard' ML and graph ML, while there are some potential impacts specific to the latter.

Regarding fairness and bias aspects of ML in general, Mehrabi et al. [154] present a comprehensive survey. For instance, ML models can disadvantage certain societal groups, e.g., when used to decide whether to grant a person a loan. In addition, models can pick up biases that exist in the data used to train the model, as Vestby and Vestby [225] point out: an example they provide is predictive policing, i.e., deciding to send police patrols to areas where we expect certain crimes to be committed. A model could be trained on historical crime data, so we would tend to send more police to areas where many crimes happen. The key issue is that the historical crime data is based on crimes that were actually reported, and more police presence tends to lead to a higher share of crimes being discovered. Hence, the historical data is biased to contain more crimes in regions where police presence is high, and learning an ML model on this data can further increase this tendency even though the crime rate could be similarly high in other regions where they simply are not detected because of low police presence [225].

Another broader impact concern is mass surveillance of public spaces. Deep learning enables accurate facial recognition and thus makes it possible to track when people are where and with whom. While there are benefits of such systems, e.g., detecting or prosecuting crime, the potential adverse effects of mass surveillance on people's freedom are substantial.

More specific to graph ML and very closely related to GNNs is the issue of algorithmic content recommendation. Since the goal of content recommendation on the Web is typically to increase the engagement with the site and the time spent on it (leading to increased ad revenue), presenting people with content (e.g., websites or posts) they agree with and/or connect with on an emotional level can lead to the desired outcome.

However, this may create or facilitate echo chambers, filter bubbles [181], polarization [208], and radicalization [221, 166]. It is worth noting that there is an ongoing debate on how pervasive these problems are in the real world [e.g., 129, 34, 159] Nevertheless, this shows that is important to ensure that the goal the ML model aims to achieve is well-aligned with our overall goals in society.

An aspect closely related to this thesis is that adversaries could exploit vulnerabilities of ML systems to their benefit. One example could be to have a fraudulent website (i.e., a node in the Web graph) be classified as trustworthy by a search engine's classification model by exploiting its vulnerabilities, e.g., by inserting carefully chosen links or modifying the website text accordingly. This could lead to thousands of people to be defrauded of money or to their computers to be infected with malware. The methods presented in Part III are contributions to mitigate such unwanted side effects in graph ML.

9.3 Open questions

We can identify a number of interesting and relevant open questions arising from this thesis. The first one is whether and how it is possible to identify the root cause of adversarial non-robustness. In the case of image classification, Ilyas et al. [107] present evidence that neural networks tend to focus on patterns in the data which are highly predictive yet brittle and incomprehensible to humans. Still, we do not have a full understanding why these non-robust features exist and how to ensure that models do not take them into account. In addition, there is no comparable study of adversarial examples in the graph domain. One potential reason for this is that it is difficult to visualize and characterize what kind of features and patterns a classifier focuses on.

Another open question which is more specific to GNNs is how to enable robust training for robustness under structure perturbations. As highlighted in Chapter 8, previous attempts were not successful or inconclusive, and thus we lack one of the most successful techniques to improve robustness from the i.i.d. setting.

Finally, as briefly mentioned in Section 9.1, properties such as privacy and calibration are also very important for safe practical application of GNNs in the real world. The space of how to combine the study of robust GNNs with privacy and calibration is underexplored, which makes it a fruitful area for future research.

Bibliography

[1] Adamic, L. A. and Glance, N. (2005). The political blogosphere and the 2004 U.S. election: Divided they blog. In *Proceedings of the 3rd International Workshop on Link Discovery*, pages 36–43. ACM Press.

[2] Agostinelli, F., Hoffman, M. D., Sadowski, P. J., and Baldi, P. (2015). Learning Activation Functions to Improve Deep Neural Networks. In *International Conference on Learning Representations (ICLR; Workshop Track)*.

[3] Al-Khayyal, F. A. and Falk, J. E. (1983). Jointly Constrained Biconvex Programming. *Mathematics of Operations Research*, 8(2):273–286.

[4] Alet, F., Weng, E., Lozano-Pérez, T., and Kaelbling, L. P. (2019). Neural Relational Inference with Fast Modular Meta-learning. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[5] Allamanis, M., Brockschmidt, M., and Khademi, M. (2018). Learning to Represent Programs with Graphs. In *International Conference on Learning Representations (ICLR)*.

[6] Amini, A., Schwarting, W., Soleimany, A., and Rus, D. (2020). Deep Evidential Regression. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[7] Ammanabrolu, P. and O., R. M. (2019). Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning. In *Proceedings of NAACL-HLT 2019*.

[8] Angriman, E., van der Grinten, A., Bojchevski, A., Zügner, D., Günnemann, S., and Meyerhenke, H. (2020). Group Centrality Maximization for Large-scale Graphs. In *2020 Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 56–69. SIAM.

[9] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein Generative Adversarial Networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR.

[10] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[11] Barabási, A.-L. and Pósfai, M. (2016). *Network Science*. Cambridge University Press, Cambridge, United Kingdom.

[12] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261 [cs, stat]*.

[13] Bengio, S., Bengio, Y., Cloutier, J., and Gescei, J. (2013). On the optimization of a synaptic learning rule. In *Optimality in Biological and Artificial Networks?*, pages 281–303. Routledge.

[14] Bengio, Y. (2000). Gradient-Based Optimization of Hyperparameters. *Neural Computation*, 12(8):1889–1900.

[15] Bessi, A. (2015). Two samples test for discrete power-law distributions. *arXiv:1503.00643 [physics, stat]*.

[16] Biggio, B., Corona, I., Fumera, G., Giacinto, G., and Roli, F. (2011). Bagging Classifiers for Fighting Poisoning Attacks in Adversarial Classification Tasks. In Sansone, C., Kittler, J., and Roli, F., editors, *Multiple Classifier Systems*, volume 6713, pages 350–359. Springer Berlin Heidelberg, Berlin, Heidelberg.

[17] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. (2013). Evasion attacks against machine learning at test time. In Blockeel, H., Kersting, K., Nijssen, S., and Železný, F., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 387–402, Berlin, Heidelberg. Springer Berlin Heidelberg.

[18] Biggio, B., Fumera, G., and Roli, F. (2014). Security Evaluation of Pattern Classifiers under Attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996.

[19] Biggio, B. and Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331.

[20] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight Uncertainty in Neural Network. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1613–1622. PMLR.

[21] Bojchevski, A. and Günnemann, S. (2018a). Bayesian Robust Attributed Graph Clustering: Joint Learning of Partial Anomalies and Group Structure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[22] Bojchevski, A. and Günnemann, S. (2018b). Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *International Conference on Learning Representations (ICLR)*.

[23] Bojchevski, A. and Günnemann, S. (2019a). Adversarial Attacks on Node Embeddings via Graph Poisoning. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 695–704. PMLR.

[24] Bojchevski, A. and Günnemann, S. (2019b). Certifiable Robustness to Graph Perturbations. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[25] Bojchevski, A., Klicpera, J., and Günnemann, S. (2020a). Efficient Robustness Certificates for Discrete Data: Sparsity-Aware Randomized Smoothing for Graphs, Images and More. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1003–1013. PMLR.

[26] Bojchevski, A., Klicpera, J., Perozzi, B., Kapoor, A., Blais, M., Rózemberczki, B., Lukasik, M., and Günnemann, S. (2020b). Scaling Graph Neural Networks with Approximate PageRank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 2464–2473, Virtual Event, CA, USA. ACM.

[27] Bojchevski, A., Matkovic, Y., and Günnemann, S. (2017). Robust Spectral Clustering for Noisy Data: Modeling Sparse Corruptions Improves Latent Embeddings. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '17, pages 737–746, Halifax, NS, Canada. ACM.

[28] Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. (2018). NetGAN: Generating Graphs via Random Walks. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 610–619. PMLR.

[29] Bongini, P., Bianchini, M., and Scarselli, F. (2021). Molecular generative Graph Neural Networks for Drug Discovery. *Neurocomputing*, 450:242–252.

[30] Borovitskiy, V., Azangulov, I., Terenin, A., Mostowsky, P., Deisenroth, M., and Durrande, N. (2021). Matérn Gaussian Processes on Graphs. In Banerjee, A. and Fukumizu, K., editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 2593–2601. PMLR.

[31] Bouritsas, G., Frasca, F., Zafeiriou, S., and Bronstein, M. M. (2021). Improving Graph Neural Network Expressivity via Subgraph Isomorphism Counting.

[32] Brock, A., Donahue, J., and Simonyan, K. (2019). Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *International Conference on Learning Representations (ICLR)*.

[33] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language Models are Few-Shot Learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[34] Bruns, A. (2019). Filter bubble. *Internet Policy Review*, 8(4).

[35] Cai, D., Shao, Z., He, X., Yan, X., and Han, J. (2005). Mining Hidden Community in Heterogeneous Social Networks. In *Proceedings of the 3rd International Workshop on Link Discovery*, LinkKDD '05, pages 58–65. ACM.

[36] Cai, H., Zheng, V. W., and Chang, K. C.-C. (2018). A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637.

[37] Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Veličković, P. (2021). Combinatorial Optimization and Reasoning with Graph Neural Networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 4348–4355, Montreal, Canada. International Joint Conferences on Artificial Intelligence Organization.

[38] Carlini, N. and Wagner, D. (2017). Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, San Jose, CA, USA. IEEE.

[39] Chakrabarti, D. and Faloutsos, C. (2006). Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1):2.

[40] Chang, H., Rong, Y., Xu, T., Huang, W., Zhang, H., Cui, P., Zhu, W., and Huang, J. (2020). A restricted black-box adversarial framework towards attacking graph embedding models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3389–3396.

[41] Chapelle, O., Schölkopf, B., and Zien, A., editors (2006). *Semi-Supervised Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass.

[42] Charpentier, B., Borchert, O., Zügner, D., Geisler, S., and Günnemann, S. (2022). Natural Posterior Network: Deep Bayesian Predictive Uncertainty for Exponential Family Distributions. In *International Conference on Learning Representations (ICLR)*.

[43] Charpentier, B., Zügner, D., and Günnemann, S. (2020). Posterior Network: Uncertainty Estimation without OOD Samples via Density-Based Pseudo-Counts. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[44] Chen, J., Lin, X., Shi, Z., and Liu, Y. (2020a). Link Prediction Adversarial Attack Via Iterative Gradient Attack. *IEEE Transactions on Computational Social Systems*, 7(4):1081–1094.

[45] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

[46] Chen, Y., Nadji, Y., Kountouras, A., Monrose, F., Perdisci, R., Antonakakis, M., and Vasiloglou, N. (2017). Practical Attacks Against Graph-based Clustering. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1125–1142, New York, NY, USA. Association for Computing Machinery.

[47] Chen, Y., Wu, L., and Zaki, M. (2020b). Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[48] Chen, Z.-M., Wei, X.-S., Wang, P., and Guo, Y. (2019). Multi-label image recognition with graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5177–5186.

[49] Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. (2019). Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 257–266, Anchorage, AK, USA. ACM.

[50] Chien, E., Peng, J., Li, P., and Milenkovic, O. (2021). Adaptive Universal Generalized PageRank Graph Neural Network. In *International Conference on Learning Representations (ICLR)*.

[51] Cisse, M., Bojanowski, P., Grave, E., Dauphin, Y., and Usunier, N. (2017). Parseval Networks: Improving Robustness to Adversarial Examples. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 854–863. PMLR.

[52] Clauset, A., Shalizi, C. R., and Newman, M. E. J. (2009). Power-Law Distributions in Empirical Data. *SIAM Review*, 51(4):661–703.

[53] Cohen, J., Rosenfeld, E., and Kolter, Z. (2019). Certified Adversarial Robustness via Randomized Smoothing. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320. PMLR.

[54] Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., and Bharath, A. A. (2018). Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65.

[55] Croce, F., Andriushchenko, M., and Hein, M. (2019). Provable Robustness of ReLU networks via Maximization of Linear Regions. In Chaudhuri, K. and Sugiyama, M., editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 2057–2066. PMLR.

[56] Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., and Song, L. (2018). Adversarial Attack on Graph Structured Data. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1115–1124. PMLR.

[57] Dalvi, N., Domingos, P., Mausam, Sanghai, S., and Verma, D. (2004). Adversarial Classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '04, pages 99–108, Seattle, WA, USA. ACM.

[58] De Cristofaro, E. (2021). A critical overview of privacy in machine learning. *IEEE Security and Privacy*, 19(4):19–27.

[59] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

[60] Deng, A. and Hooi, B. (2021). Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4027–4035.

[61] Derrow-Pinion, A., She, J., Wong, D., Lange, O., Hester, T., Perez, L., Nunkesser, M., Lee, S., Guo, X., Wiltshire, B., Battaglia, P. W., Gupta, V., Li, A., Xu, Z., Sanchez-Gonzalez, A., Li, Y., and Velickovic, P. (2021). ETA Prediction with Graph Neural Networks in Google Maps. In Demartini, G., Zuccon, G., Culpepper, J. S., Huang, Z., and Tong, H., editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 3767–3776. ACM.

[62] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North*

American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, pages 4171–4186. Association for Computational Linguistics.

[63] Diehl, F., Brunner, T., Le, M. T., and Knoll, A. (2019). Graph Neural Networks for Modelling Traffic Participant Interaction. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 695–701, Paris, France. IEEE.

[64] Dwork, C. (2008). Differential Privacy: A Survey of Results. In Agrawal, M., Du, D., Duan, Z., and Li, A., editors, *Theory and Applications of Models of Computation*, pages 1–19, Berlin, Heidelberg. Springer Berlin Heidelberg.

[65] Dwork, C. and Roth, A. (2013). The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3-4):211–407.

[66] Elflein, S., Charpentier, B., Zügner, D., and Günnemann, S. (2021). On Out-of-distribution Detection with Energy-based Models. In *ICML Workshop on Uncertainty & Robustness in Deep Learning*.

[67] Elinas, P., Bonilla, E. V., and Tiao, L. (2020). Variational Inference for Graph Convolutional Networks in the Absence of Graph Data and Adversarial Settings. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[68] Ellers, M., Cochez, M., Schumacher, T., Strohmaier, M., and Lemmerich, F. (2019). Privacy attacks on network embeddings. *arXiv preprint arXiv:1912.10979*.

[69] Entezari, N., Al-Sayouri, S. A., Darvishzadeh, A., and Papalexakis, E. E. (2020). All You Need Is Low (Rank): Defending Against Adversarial Attacks on Graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, pages 169–177, New York, NY, USA. Association for Computing Machinery.

[70] Eswaran, D., Günnemann, S., Faloutsos, C., Makhija, D., and Kumar, M. (2017). ZooBP: Belief propagation for heterogeneous networks. *Proc. VLDB Endow.*, 10(5):625–636.

[71] Executive Agency for Small and Medium sized Enterprises. (2020). *Artificial Intelligence: Critical Industrial Applications : Report on Market Analysis of Prioritised Value Chains, the Most Critical AI Applications and the Conditions for AI Rollout.* Publications Office, LU.

[72] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018). Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634.

[73] Faber, L., Martinkus, K., Papp, P. A., and Wattenhofer, R. (2021). DropGNN: Random Dropouts Increase the Expressiveness of Graph Neural Networks. In Ranzato, M., Beygelzimer, A., Nguyen, K., Liang, P., Vaughan, J., and Dauphin, Y., editors, *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc.

[74] Fan, A., Bhosale, S., Schwenk, H., Ma, Z., El-Kishky, A., Goyal, S., Baines, M., Celebi, O., Wenzek, G., Chaudhary, V., et al. (2021). Beyond english-centric multilingual machine translation. *Journal of Machine Learning Research*, 22(107):1–48.

[75] Feng, B., Wang, Y., Li, X., and Ding, Y. (2020). Scalable Adversarial Attack on Graph Neural Networks with Alternating Direction Method of Multipliers. *arXiv:2009.10233 [cs, stat]*.

[76] Fernandes, P., Allamanis, M., and Brockschmidt, M. (2019). Structured Neural Summarization. In *International Conference on Learning Representations (ICLR)*.

[77] Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR.

[78] Franceschi, L., Niepert, M., Pontil, M., and He, X. (2019). Learning Discrete Structures for Graph Neural Networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1972–1982. PMLR.

[79] Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059. PMLR.

[80] Gao, J., Zhang, T., and Xu, C. (2019). Graph convolutional tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4649–4659.

[81] Geisler, S., Schmidt, T., Şirin, H., Zügner, D., and Günnemann, S. (2021a). Robustness of Graph Neural Networks at Scale. In Ranzato, M., Beygelzimer, A., Nguyen, K., Liang, P., Vaughan, J., and Dauphin, Y., editors, *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc.

[82] Geisler, S., Sommer, J., Schuchardt, J., Bojchevski, A., and Günnemann, S. (2021b). Generalization of Neural Combinatorial Solvers Through the Lens of Adversarial Robustness. *arXiv:2110.10942 [cs]*.

[83] Geisler, S., Zügner, D., and Günnemann, S. (2020). Reliable Graph Neural Networks via Robust Aggregation. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F.,

and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[84] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural Message Passing for Quantum Chemistry. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR.

[85] Globerson, A. and Roweis, S. (2006). Nightmare at Test Time: Robust Learning by Feature Deletion. In Cohen, W. and Moore, A., editors, *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 353–360. Association for Computing Machinery.

[86] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[87] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

[88] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2015). Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations (ICLR)*.

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

[90] Grigorescu, S., Trasnea, B., Cocias, T., and Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386.

[91] Grosse, K., Papernot, N., Manoharan, P., Backes, M., and McDaniel, P. (2016). Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *arXiv:1606.04435 [cs]*.

[92] Grosse, K., Papernot, N., Manoharan, P., Backes, M., and McDaniel, P. (2017). Adversarial Examples for Malware Detection. In Foley, S. N., Gollmann, D., and Snekkenes, E., editors, *Computer Security – ESORICS 2017*, pages 62–79, Cham. Springer International Publishing.

[93] Grover, A. and Leskovec, J. (2016). Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '16, pages 855–864, San Francisco, California, USA. ACM.

[94] Günnemann, S. (2022). Graph Neural Networks: Adversarial Robustness. In Wu, L., Cui, P., Pei, J., and Zhao, L., editors, *Graph Neural Networks: Foundations, Frontiers, and Applications*, chapter 8, pages 149–176. Springer, Singapore.

[95] Guo, S., Lin, Y., Feng, N., Song, C., and Wan, H. (2019). Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 922–929.

[96] Guo, Y., Wei, X., Wang, G., and Zhang, B. (2021). Meaningful Adversarial Stickers for Face Recognition in Physical World. *arXiv:2104.06728 [cs]*.

[97] Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[98] Hamilton, W. L. (2020). Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159.

[99] Hamrick, J. B., Allen, K. R., Bapst, V., Zhu, T., McKee, K. R., Tenenbaum, J., and Battaglia, P. W. (2018). Relational inductive bias for physical construction in humans and machines. In Kalish, C., Rau, M. A., Zhu, X. J., and Rogers, T. T., editors, *Proceedings of the 40th Annual Meeting of the Cognitive Science Society, CogSci 2018, Madison, WI, USA, July 25-28, 2018*. cognitivesciencesociety.org.

[100] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

[101] Hein, M. and Andriushchenko, M. (2017). Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[102] Hellendoorn, V. J., Sutton, C., Singh, R., Maniatis, P., and Bieber, D. (2020). Global Relational Models of Source Code. In *International Conference on Learning Representations*.

[103] Hendrycks, D. and Dietterich, T. (2019). Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. In *International Conference on Learning Representations*.

[104] Hooi, B., Shah, N., Beutel, A., Günnemann, S., Akoglu, L., Kumar, M., Makhija, D., and Faloutsos, C. (2016). BIRDNEST: Bayesian Inference for Ratings-Fraud Detection. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 495–503. Society for Industrial and Applied Mathematics.

[105] Hsieh, I.-C. and Li, C.-T. (2021). NetFense: Adversarial Defenses against Privacy Attacks on Neural Networks for Graph Data. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1.

[106] Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open Graph Benchmark: Datasets for Machine Learning on Graphs. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[107] Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial Examples Are Not Bugs, They Are Features. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[108] Jia, J., Wang, B., Cao, X., and Gong, N. Z. (2020). Certified robustness of community detection against adversarial structural perturbation via randomized smoothing. In *Proceedings of The Web Conference 2020*, pages 2718–2724.

[109] Jiang, D., Wu, Z., Hsieh, C.-Y., Chen, G., Liao, B., Wang, Z., Shen, C., Cao, D., Wu, J., and Hou, T. (2021). Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models. *Journal of Cheminformatics*, 13(1):12.

[110] Jin, W., Li, Y., Xu, H., Wang, Y., Ji, S., Aggarwal, C., and Tang, J. (2021). Adversarial Attacks and Defenses on Graphs. *SIGKDD Explor. Newsl.*, 22(2):19–34.

[111] Jin, W., Ma, Y., Liu, X., Tang, X., Wang, S., and Tang, J. (2020). Graph Structure Learning for Robust Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 66–74, Virtual Event, CA, USA. ACM.

[112] Jo, E., Sunwoo, M., and Lee, M. (2021). Vehicle Trajectory Prediction Using Hierarchical Graph Neural Network for Considering Interaction among Multimodal Maneuvers. *Sensors*, 21(16):5354.

[113] Junying Li, Deng Cai, X. H. (2017). Learning Graph-Level Representation for Drug Discoveryk. *arXiv preprint arXiv:1709.03741*.

[114] Ker, J., Wang, L., Rao, J., and Lim, T. (2017). Deep learning applications in medical image analysis. *IEEE access : practical innovations, open solutions*, 6:9375–9389.

[115] Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. (2018). Neural Relational Inference for Interacting Systems. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2688–2697. PMLR.

[116] Kipf, T. N. and Welling, M. (2016). Variational Graph Auto-Encoders. *NIPS Workshop on Bayesian Deep Learning*.

[117] Kipf, T. N. and Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[118] Klicpera, J., Becker, F., and Günnemann, S. (2021). GemNet: Universal Directional Graph Neural Networks for Molecules. In Ranzato, M., Beygelzimer, A., Nguyen, K., Liang, P., Vaughan, J., and Dauphin, Y., editors, *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc.

[119] Klicpera, J., Bojchevski, A., and Günnemann, S. (2019a). Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations (ICLR)*.

[120] Klicpera, J., Giri, S., Margraf, J. T., and Günnemann, S. (2020a). Fast and Uncertainty-Aware Directional Message Passing for Non-Equilibrium Molecules. In *NeurIPS-W*.

[121] Klicpera, J., Groß, J., and Günnemann, S. (2020b). Directional Message Passing for Molecular Graphs. In *International Conference on Learning Representations (ICLR)*.

[122] Klicpera, J., Weißenberger, S., and Günnemann, S. (2019b). Diffusion Improves Graph Learning. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[123] Komkov, S. and Petiushko, A. (2019). AdvHat: Real-world adversarial attack on ArcFace Face ID system. *arXiv preprint arXiv:1908.08705*.

[124] Kopetzki, A.-K., Charpentier, B., Zügner, D., Giri, S., and Günnemann, S. (2021). Evaluating Robustness of Predictive Uncertainty Estimation: Are Dirichlet-based Models Reliable? In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5707–5718. PMLR.

[125] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[126] Kumar, A. and Goldstein, T. (2021). Center Smoothing: Certified Robustness for Networks with Structured Outputs. In Ranzato, M., Beygelzimer, A., Nguyen, K., Liang, P., Vaughan, J., and Dauphin, Y., editors, *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc.

[127] Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017a). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[128] Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017b). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[129] Ledwich, M. and Zaitsev, A. (2020). Algorithmic extremism: Examining YouTube's rabbit hole of radicalization. *First Monday*.

[130] Lee, G.-H., Yuan, Y., Chang, S., and Jaakkola, T. (2019). Tight Certificates of Adversarial Robustness for Randomly Smoothed Classifiers. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[131] Leman, A. and Weisfeiler, B. (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16.

[132] Li, B., Wang, Y., Singh, A., and Vorobeychik, Y. (2016). Data Poisoning Attacks on Factorization-Based Collaborative Filtering. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

[133] Li, J. (2019a). Lecture notes of lecture 10: "Empirical defenses for adversarial examples" of the course"Robustness in Machine Learning" (Stanford CSE 599-M).

[134] Li, J. (2019b). Lecture notes of lecture 9: "Introduction to Adversarial Examples" of the course"Robustness in Machine Learning" (Stanford CSE 599-M).

[135] Li, J. (2019c). Lecture notes of the course "Robustness in Machine Learning" (Stanford CSE 599-M).

[136] Li, J., Xie, T., Liang, C., Xie, F., He, X., and Zheng, Z. (2021a). Adversarial Attack on Large Scale Graph. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1.

[137] Li, K., Luo, G., Ye, Y., Li, W., Ji, S., and Cai, Z. (2021b). Adversarial Privacy-Preserving Graph Embedding Against Inference Attack. *IEEE Internet of Things Journal*, 8(8):6904–6915.

[138] Li, Y., Yu, R., Shahabi, C., and Liu, Y. (2018a). Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations (ICLR)*.

[139] Li, Z., Chen, Q., and Koltun, V. (2018b). Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[140] Liang, B., Li, H., Su, M., Bian, P., Li, X., and Shi, W. (2018). Deep text classification can be fooled. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pages 4208–4215, Stockholm, Sweden. AAAI Press.

[141] Lin, W., Ji, S., and Li, B. (2020). Adversarial Attacks on Link Prediction Algorithms Based on Graph Neural Networks. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 370–380, Taipei Taiwan. ACM.

[142] Liu, J., Lin, Z., Padhy, S., Tran, D., Bedrax Weiss, T., and Lakshminarayanan, B. (2020a). Simple and Principled Uncertainty Estimation with Deterministic Deep Learning via Distance Awareness. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[143] Liu, T.-Y. (2011). *Learning to Rank for Information Retrieval*. Springer, Berlin Heidelberg New York.

[144] Liu, Z.-Y., Li, S.-Y., Chen, S., Hu, Y., and Huang, S.-J. (2020b). Uncertainty Aware Graph Gaussian Process for Semi-Supervised Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4957–4964.

[145] London, B. and Getoor, L. (2015). Collective Classification of Network Data. In Aggarwal, C. C., editor, *Data Classification: Algorithms and Applications*, pages 399–416. CRC Press, London, England.

[146] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2018). Towards Deep Learning Models Resistant to Adversarial Attacks. In *International Conference on Learning Representations (ICLR)*.

[147] Madry, A. and Schmidt, L. (2018). A Brief Introduction to Adversarial Examples.

[148] Malinin, A. and Gales, M. (2018). Predictive Uncertainty Estimation via Prior Networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[149] Malinin, A. and Gales, M. (2019). Reverse KL-Divergence Training of Prior Networks: Improved Uncertainty and Adversarial Robustness. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[150] Mann, H. B. and Whitney, D. R. (1947). On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50–60.

[151] Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. (2019). Provably Powerful Graph Networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[152] McCallum, A. K., Nigam, K., Rennie, J., and Seymore, K. (2000). Automating the Construction of Internet Portals with Machine Learning. *Information Retrieval*, 3(2):127–163.

[153] McPherson, M., Smith-Lovin, L., and Cook, J. M. (2001). Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology*, 27(1):415–444.

[154] Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., and Galstyan, A. (2021). A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6):1–35.

[155] Mei, S. and Zhu, X. (2015). Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2871–2877, Austin, Texas. AAAI Press.

[156] Metzen, J. H., Genewein, T., Fischer, V., and Bischoff, B. (2017). On Detecting Adversarial Perturbations. In *International Conference on Learning Representations (ICLR)*. OpenReview.net.

[157] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Series in Computer Science. McGraw-Hill, New York.

[158] Miyato, T., Maeda, S.-I., Koyama, M., and Ishii, S. (2019). Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1979–1993.

[159] Moeller, J., Helberger, N., et al. (2018). Beyond the filter bubble: Concepts, myths, evidence and issues for future debates.

[160] Mohamed Mohamed El-Sayed, A. (2016). Modeling Multivariate Correlated Binary Data. *American Journal of Theoretical and Applied Statistics*, 5(4):225.

[161] Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, Honolulu, HI. IEEE.

[162] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, Las Vegas, NV, USA. IEEE.

[163] Morris, C., Fey, M., and Kriege, N. (2021). The Power of the Weisfeiler-Leman Algorithm for Machine Learning with Graphs. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 4543–4550, Montreal, Canada. International Joint Conferences on Artificial Intelligence Organization.

[164] Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. (2019). Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609.

[165] Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E. C., and Roli, F. (2017). Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38.

[166] Munn, L. (2019). Alt-right pipeline: Individual journeys to extremism online. *First Monday*.

[167] Naik, D. and Mammone, R. (1992). Meta-Neural Networks that Learn by Learning. In *IJCNN International Joint Conference on Neural Networks*, volume 1, pages 437–442, Baltimore, MD, USA. IEEE.

[168] Newman, M. E. J. (2003). Mixing patterns in networks. *Physical Review E*, 67(2):026126.

[169] Ng, Y. C., Colombo, N., and Silva, R. (2018). Bayesian Semi-supervised Learning with Graph Gaussian Processes. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[170] Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436.

[171] Nichol, A., Achiam, J., and Schulman, J. (2018). On First-Order Meta-Learning Algorithms. *arXiv:1803.02999 [cs]*.

[172] Olatunji, I. E., Funke, T., and Khosla, M. (2021). Releasing Graph Neural Networks with Differential Privacy Guarantees. *arXiv:2109.08907 [cs]*.

[173] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab / Stanford InfoLab.

[174] Pal, S., Regol, F., and Coates, M. (2019a). Bayesian graph convolutional neural networks using node copying. In *Proc. Workshop on Learning and Reasoning with Graph-Structured Data at Int. Conf. Machine Learning*.

[175] Pal, S., Regol, F., and Coates, M. (2019b). Bayesian graph convolutional neural networks using non-parametric graph learning. *arXiv preprint arXiv:1910.12132*.

[176] Papernot, N. and McDaniel, P. (2017). Extending Defensive Distillation.

[177] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. (2017). Practical Black-Box Attacks against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519, Abu Dhabi United Arab Emirates. ACM.

[178] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016a). The Limitations of Deep Learning in Adversarial Settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387, Saarbrucken. IEEE.

[179] Papernot, N., McDaniel, P., Swami, A., and Harang, R. (2016b). Crafting adversarial input sequences for recurrent neural networks. In *MILCOM 2016 - 2016 IEEE Military Communications Conference*, pages 49–54, Baltimore, MD, USA. IEEE Press.

[180] Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. (2016c). Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, San Jose, CA. IEEE.

[181] Pariser, E. (2011). *The Filter Bubble: What the Internet Is Hiding from You*. Viking, London.

[182] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '14, pages 701–710, New York, New York, USA. ACM.

[183] Pfeifle, T. (2021). Adversarial Training for Graph Neural Networks. Master thesis, Technical University of Munich.

[184] Pham, T., Tran, T., Phung, D., and Venkatesh, S. (2017). Column Networks for Collective Classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

[185] Prates, M., Avelar, P. H., Lemos, H., Lamb, L. C., and Vardi, M. Y. (2019). Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4731–4738.

[186] Qi, M., Li, W., Yang, Z., Wang, Y., and Luo, J. (2019). Attentive relational networks for mapping images to scene graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3957–3966.

[187] Qualizza, A., Belotti, P., and Margot, F. (2012). Linear Programming Relaxations of Quadratically Constrained Quadratic Programs. In Lee, J. and Leyffer, S., editors, *Mixed Integer Nonlinear Programming*, volume 154, pages 407–426. Springer New York, New York, NY.

[188] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[189] Raghunathan, A., Steinhardt, J., and Liang, P. S. (2018). Semidefinite relaxations for certifying robustness to adversarial examples. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[190] Reyes, A. K., Caicedo, J. C., and Camargo, J. E. (2015). Fine-tuning Deep Convolutional Networks for Plant Recognition. *CLEF (Working Notes)*, 1391:467–475.

[191] Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., and Bronstein, M. (2020). Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML Workshop on Graph Representation Learning*.

[192] Sabour, S., Cao, Y., Faghri, F., and Fleet, D. J. (2016). Adversarial Manipulation of Deep Representations. In *International Conference on Learning Representations (ICLR)*.

[193] Salman, H., Li, J., Razenshteyn, I., Zhang, P., Zhang, H., Bubeck, S., and Yang, G. (2019). Provably Robust Deep Learning via Adversarially Trained Smoothed Classifiers. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[194] Sankaranarayanan, S., Jain, A., Chellappa, R., and Lim, S. N. (2018). Regularizing Deep Networks Using Efficient Layerwise Adversarial Training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[195] Scarselli, F., Gori, M., Ah Chung Tsoi, Hagenbuchner, M., and Monfardini, G. (2009). The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

[196] Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., and Welling, M. (2018). Modeling Relational Data with Graph Convolutional Networks. In Gangemi, A., Navigli, R., Vidal, M.-E., Hitzler, P., Troncy, R., Hollink, L., Tordai, A., and Alam, M., editors, *The Semantic Web*, volume 10843, pages 593–607. Springer International Publishing, Cham.

[197] Schmidhuber, J. (1992). Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation*, 4(1):131–139.

[198] Schuchardt, J., Bojchevski, A., Klicpera, J., and Günnemann, S. (2021). Collective robustness certificates. In *International Conference on Learning Representations (ICLR)*.

[199] Schütt, K., Kindermans, P.-J., Sauceda Felix, H. E., Chmiela, S., Tkatchenko, A., and Müller, K.-R. (2017). SchNet: A Continuous-Filter Convolutional Neural Network for Modeling Quantum Interactions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[200] Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. (2019). Learning a SAT Solver from Single-Bit Supervision. In *International Conference on Learning Representations (ICLR)*.

[201] Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008). Collective Classification in Network Data. *AI Magazine*, 29(3):93.

[202] Shang, C., Chen, J., and Bi, J. (2021). Discrete Graph Structure Learning for Forecasting Multiple Time Series. In *International Conference on Learning Representations (ICLR)*.

[203] Sharif, M., Bhagavatula, S., Bauer, L., and Reiter, M. K. (2016). Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 Acm Sigsac Conference on Computer and Communications Security*, pages 1528–1540.

[204] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*.

[205] Skarding, J., Gabrys, B., and Musial, K. (2021). Foundations and Modeling of Dynamic Networks Using Dynamic Graph Neural Networks: A Survey. *IEEE Access*, 9:79143–79168.

[206] Stadler, M., Charpentier, B., Geisler, S., Zügner, D., and Günnemann, S. (2021). Graph Posterior Network: Bayesian Predictive Uncertainty for Node Classification. In Ranzato, M., Beygelzimer, A., Nguyen, K., Liang, P., Vaughan, J., and Dauphin, Y., editors, *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc.

[207] Statista (2021). Global AI funding by quarter 2011-2021. https://www.statista.com/statistics/943151/ai-funding-worldwide-by-quarter/.

[208] Stroud, N. J. (2008). Media use and political predispositions: Revisiting the concept of selective exposure. *Political Behavior*, 30(3):341–366.

[209] Sun, L., Dou, Y., Yang, C., Wang, J., Yu, P. S., He, L., and Li, B. (2018a). Adversarial Attack and Defense on Graph Data: A Survey. *arXiv preprint arXiv:1812.10528*.

[210] Sun, M., Tang, J., Li, H., Li, B., Xiao, C., Chen, Y., and Song, D. (2018b). Data Poisoning Attack against Unsupervised Node Embedding Methods. *arXiv:1810.12881 [cs, stat]*.

[211] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

[212] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2014). Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*.

[213] Tang, H., Ma, G., Chen, Y., Guo, L., Wang, W., Zeng, B., and Zhan, L. (2020). Adversarial Attack on Hierarchical Graph Pooling Neural Networks. *arXiv:2005.11560 [cs, stat]*.

[214] Tewari, A., Fried, O., Thies, J., Sitzmann, V., Lombardi, S., Sunkavalli, K., Martin-Brualla, R., Simon, T., Saragih, J., Nießner, M., et al. (2020). State of the art on neural rendering. In *Computer Graphics Forum*, volume 39, pages 701–727. Wiley Online Library.

[215] Thedchanamoorthy, G., Piraveenan, M., Kasthuriratna, D., and Senanayake, U. (2014). Node Assortativity in Complex Networks: An Alternative Approach. *Procedia Computer Science*, 29:2449–2461.

[216] Thrun, S. and Pratt, L. (1998). Learning to Learn: Introduction and Overview. In Thrun, S. and Pratt, L., editors, *Learning to Learn*, pages 3–17. Springer US, Boston, MA.

[217] Tishby, N., Pereira, F. C., and Bialek, W. (1999). The information bottleneck method. In *Proc. of the 37-Th Annual Allerton Conference on Communication, Control and Computing*, pages 368–377.

[218] Tjeng, V., Xiao, K. Y., and Tedrake, R. (2019). Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations (ICLR)*.

[219] Torkamani, M. and Lowd, D. (2013). Convex Adversarial Collective Classification. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 642–650. PMLR.

[220] Tramèr, F., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. (2017). The space of transferable adversarial examples.

[221] Tufekci, Z. (2018). Opinion — YouTube, the Great Radicalizer. *The New York Times*.

[222] Ummenhofer, B., Prantl, L., Thuerey, N., and Koltun, V. (2019). Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations (ICLR)*.

[223] Unke, O. T. and Meuwly, M. (2019). PhysNet: A Neural Network for Predicting Energies, Forces, Dipole Moments, and Partial Charges. *Journal of Chemical Theory and Computation*, 15(6):3678–3693.

[224] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*.

[225] Vestby, A. and Vestby, J. (2021). Machine learning and the police: Asking the right questions. *Policing: A Journal of Policy and Practice*, 15(1):44–58.

[226] Wan, X., Kenlay, H., Ru, B., Blaas, A., Osborne, M., and Dong, X. (2021). Adversarial Attacks on Graph Classifiers via Bayesian Optimisation. In Ranzato, M., Beygelzimer, A., Nguyen, K., Liang, P., Vaughan, J., and Dauphin, Y., editors, *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc.

[227] Wang, B., Jia, J., Cao, X., and Gong, N. Z. (2021a). Certified Robustness of Graph Neural Networks against Adversarial Structural Perturbation. In *Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 1645–1653, Virtual Event, Singapore. ACM.

[228] Wang, H., He, M., Wei, Z., Wang, S., Yuan, Y., Du, X., and Wen, J.-R. (2021b). Approximate Graph Propagation. In *Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '21, pages 1686–1696, Virtual Event, Singapore. ACM.

[229] Wang, H., Ren, H., and Leskovec, J. (2021c). Relational Message Passing for Knowledge Graph Completion. In *Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1697–1707. ACM.

[230] Wang, J., Luo, M., Suya, F., Li, J., Yang, Z., and Zheng, Q. (2020a). Scalable attack on graph data by injecting vicious nodes. *Data Mining and Knowledge Discovery*, 34(5):1363–1389.

[231] Wang, Y., Liu, S., Yoon, M., Lamba, H., Wang, W., Faloutsos, C., and Hooi, B. (2020b). Provably Robust Node Classification via Low-Pass Message Passing. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 621–630. IEEE.

[232] Weaver, N. (1999). *Lipschitz Algebras*. WORLD SCIENTIFIC.

[233] Wei, X.-S., Xie, C.-W., Wu, J., and Shen, C. (2018). Mask-CNN: Localizing parts and selecting descriptors for fine-grained bird species categorization. *Pattern Recognition*, 76:704–714.

[234] Wong, E. and Kolter, Z. (2018). Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5286–5295. PMLR.

[235] Wu, C., Wu, F., Cao, Y., Huang, Y., and Xie, X. (2021a). FedGNN: Federated Graph Neural Network for Privacy-Preserving Recommendation. *arXiv:2102.04925 [cs]*.

[236] Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019a). Simplifying Graph Convolutional Networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6861–6871. PMLR.

[237] Wu, H., Wang, C., Tyshetskiy, Y., Docherty, A., Lu, K., and Zhu, L. (2019b). Adversarial Examples for Graph Data: Deep Insights into Attack and Defense. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 4816–4823, Macao, China. International Joint Conferences on Artificial Intelligence Organization.

[238] Wu, L., Cui, P., Jian, P., and Lian, Z., editors (2022). *Graph Neural Networks: Foundations, Frontiers, and Applications.* Springer, Singapore.

[239] Wu, T., Ren, H., Li, P., and Leskovec, J. (2020a). Graph Information Bottleneck. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[240] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2021b). A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24.

[241] Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X., and Zhang, C. (2020b). Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 753–763, Virtual Event, CA, USA. ACM.

[242] Xia, F., Liu, T.-Y., Wang, J., Zhang, W., and Li, H. (2008). Listwise Approach to Learning to Rank: Theory and Algorithm. In McCallum, A. and Roweis, S., editors, *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1192–1199. Association for Computing Machinery.

[243] Xu, d., Rruan, C., Korpeoglu, E., Kumar, S., and Achan, K. (2020a). Inductive representation learning on temporal graphs. In *International Conference on Learning Representations (ICLR)*.

[244] Xu, J., Sun, Y., Jiang, X., Wang, Y., Yang, Y., Wang, C., and Lu, J. (2021a). Blindfolded Attackers Still Threatening: Strict Black-Box Adversarial Attacks on Graphs. *arXiv:2012.06757 [cs]*.

[245] Xu, K., Chen, H., Liu, S., Chen, P.-Y., Weng, T.-W., Hong, M., and Lin, X. (2019a). Topology Attack and Defense for Graph Neural Networks: An Optimization Perspective. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, pages 3961–3967, Macao, China. AAAI Press.

[246] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019b). How Powerful are Graph Neural Networks? In *International Conference on Learning Representations (ICLR)*.

[247] Xu, K., Liu, S., Chen, P.-Y., Sun, M., Ding, C., Kailkhura, B., and Lin, X. (2020b). Towards an Efficient and General Framework of Robust Training for Graph Neural Networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8479–8483. IEEE.

[248] Xu, W., Evans, D., and Qi, Y. (2018). Feature squeezing: Detecting adversarial examples in deep neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society.

[249] Xu, Z., Du, B., and Tong, H. (2021b). Graph Sanitation with Application to Node Classification. *arXiv:2105.09384 [cs]*.

[250] Yasunaga, M., Ren, H., Bosselut, A., Liang, P., and Leskovec, J. (2021). QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering. In *North American Chapter of the Association for Computational Linguistics*, NAACL 2021.

[251] Yeo, I.-K. and Johnson, R. A. (2000). A New Family of Power Transformations to Improve Normality or Symmetry. *Biometrika*, 87(4):954–959.

[252] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018a). Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 974–983, London, United Kingdom. ACM.

[253] Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. (2018b). Hierarchical Graph Representation Learning with Differentiable Pooling. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[254] You, J., Ying, R., and Leskovec, J. (2019). Position-aware Graph Neural Networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7134–7143. PMLR.

[255] Yu, D., Zhang, R., Jiang, Z., Wu, Y., and Yang, Y. (2020). Graph-revised convolutional network. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 378–393. Springer.

[256] Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., Shanahan, M., Langston, V., Pascanu, R., Botvinick, M., Vinyals, O., and Battaglia, P. (2018). Relational Deep Reinforcement Learning. *arXiv:1806.01830 [cs, stat]*.

[257] Zečević, M., Dhami, D. S., Veličković, P., and Kersting, K. (2021). Relating Graph Neural Networks to Structural Causal Models. *arXiv:2109.04173 [cs, stat]*.

[258] Zhang, H., Zheng, T., Gao, J., Miao, C., Su, L., Li, Y., and Ren, K. (2019a). Data Poisoning Attack against Knowledge Graph Embedding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 4853–4859, Macao, China. International Joint Conferences on Artificial Intelligence Organization.

[259] Zhang, M. and Chen, Y. (2018). Link Prediction Based on Graph Neural Networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[260] Zhang, X. and Zitnik, M. (2020). GNNGuard: Defending Graph Neural Networks against Adversarial Attacks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[261] Zhang, Y., Pal, S., Coates, M., and Ustebay, D. (2019b). Bayesian graph convolutional neural networks for semi-supervised classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5829–5836.

[262] Zhang, Z., Jia, J., Wang, B., and Gong, N. Z. (2021). Backdoor Attacks to Graph Neural Networks. In *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, pages 15–26, Virtual Event Spain. ACM.

[263] Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., and Li, H. (2020a). T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858.

[264] Zhao, M., An, B., Yu, Y., Liu, S., and Pan, S. (2018). Data Poisoning Attacks on Multi-Task Relationship Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[265] Zhao, X., Chen, F., Hu, S., and Cho, J.-H. (2020b). Uncertainty Aware Semi-Supervised Learning on Graph Data. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

[266] Zheng, Q., Zou, X., Dong, Y., Cen, Y., Yin, D., Xu, J., Yang, Y., and Tang, J. (2021). Graph Robustness Benchmark: Benchmarking the Adversarial Robustness of Graph Machine Learning. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 34. Curran Associates, Inc.

[267] Zhi, Y.-C., Ng, Y. C., and Dong, X. (2020). Gaussian Processes on Graphs via Spectral Kernel Learning. *arXiv:2006.07361 [cs, eess, stat]*.

[268] Zhu, D., Zhang, Z., Cui, P., and Zhu, W. (2019a). Robust Graph Convolutional Networks Against Adversarial Attacks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 1399–1407, Anchorage, AK, USA. ACM.

[269] Zhu, J., Rossi, R. A., Rao, A., Mai, T., Lipka, N., Ahmed, N. K., and Koutra, D. (2021). Graph Neural Networks with Heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11168–11176.

[270] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *Computer Vision (ICCV), 2017 IEEE International Conference On*.

[271] Zhu, S., Zhou, C., Pan, S., Zhu, X., and Wang, B. (2019b). Relation Structure-Aware Heterogeneous Graph Neural Network. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 1534–1539, Beijing, China. IEEE.

[272] Zitnik, M., Agrawal, M., and Leskovec, J. (2018). Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics (Oxford, England)*, 34(13):i457–i466.

[273] Zügner, D., Akbarnejad, A., and Günnemann, S. (2018). Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 2847–2856. ACM.

[274] Zügner, D., Aubet, F.-X., Satorras, V. G., Januschowski, T., Günnemann, S., and Gasthaus, J. (2021a). A Study of Joint Graph Inference and Forecasting. In *ICML Time Series Workshop*.

[275] Zügner, D., Borchert, O., Akbarnejad, A., and Günnemann, S. (2020). Adversarial Attacks on Graph Neural Networks: Perturbations and Their Patterns. *ACM Transactions on Knowledge Discovery from Data*, 14(5).

[276] Zügner, D. and Günnemann, S. (2019a). Adversarial Attacks on Graph Neural Networks via Meta Learning. In *International Conference on Learning Representations (ICLR)*.

[277] Zügner, D. and Günnemann, S. (2019b). Certifiable Robustness and Robust Training for Graph Convolutional Networks. In *Proceedings of the 25th ACM SIGKDD*

International Conference on Knowledge Discovery & Data Mining, KDD '19, pages 246–256. ACM.

[278] Zügner, D. and Günnemann, S. (2020). Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 1656–1665. ACM.

[279] Zügner, D., Kirschstein, T., Catasta, M., Leskovec, J., and Günnemann, S. (2021b). Language-Agnostic Representation Learning of Source Code from Structure and Context. In *International Conference on Learning Representations (ICLR)*.

A Data and Code

A.1 Datasets

In Table A.1 we show the details of the different datasets used throughout this thesis. We use the largest connected component (LCC) of each graph.

Dataset	N_{LCC}	E_{LCC}	D	K	Ref.
Cora-ML	2,810	7,981	2,879	7	[152, 22]
Citeseer	2,110	3,757	3,703	6	[201]
PolBlogs	1,222	16,714	-	2	[1]
Pubmed	19,717	44,324	500	3	[201]

Table A.1: Dataset statistics.

A.2 Code

The code implementations of the methods described in this thesis are publicly available. The respective URLs to the repositories of the papers are displayed in Table A.2.

Ch.	Ref.	Title	Project page	Repository
3	[273], [275]	Adversarial Attacks on Graph Neural Networks: Perturbations and their Patterns	/nettack/	/nettack/
4	[276]	Adversarial Attacks on Graph Neural Networks via Meta Learning	/gnn_meta_attack/	/gnn-meta-attack/
6	[277]	Certifiable Robustness and Robust Training for Graph Convolutional Networks	/robust_gcn/	/robust-gcn/
7	[278]	Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations	/robust_gcn/	/robust-gcn-structure/

Table A.2: List of own publications which form the basis of this thesis with links to the respective code repositories. The project pages are available at `https://www.daml.in.tum.de/[project]`. The repositories are available at `https://github.com/danielzuegner/[repository]`.

B Adversarial Attacks on Graph Neural Networks: Perturbations and their Patterns

Metric	Cora-ML	Citeseer	Polblogs	PubMed
Node Degree	+0.0081	+0.0011	−0.7901	−0.0004
2-Hop Node Degree	−0.2976	−0.1589	−0.1455	−0.1505
Local Degree Assortativity	+0.0040	+0.2401	+0.0016	−0.0050
Node Betweenness Centrality	−0.0509	−0.0003	+0.0002	+0.1042
PageRank Score	+0.0067	+0.0077	−0.0007	+0.0054
Closeness Centrality	−0.0069	+0.0023	−0.0052	−0.0332
Relative Node Degree	+0.0060	−0.0009	−0.0003	−0.0002
Relative 2-Hop Node Degree	−0.0025	−0.0001	−0.0002	−0.0169
Relative Node Betweenness Centrality	+0.0002	−0.0002	+0.0001	−0.0016
Relative PageRank Score	+0.0036	+0.0036	+0.0001	+0.0284
Relative Closeness Centrality	−0.0053	+0.0006	−0.0037	−0.0011
Relative Local Degree Assortativity	−0.0003	−0.0002	+0.0037	−0.0020
Attribute Similarity	−0.0052	−0.0014	0	+0.0056
Mean TFIDF of Shared Features	+0.0128	−0.0006	0	+0.0007
Number of Shared Features	−0.0057	+0.0185	0	+0.0060
Number of Additional Features	+0.1470	+0.2826	0	+0.2236
Average Frequency of Additional Features in Target Class	−0.2052	−0.1683	0	−0.2763
Label Equality	−0.2319	−0.1126	−0.0487	−0.1390

Table B.1: Parameters of linear ranking models when trained on different graphs with strong L1-regularization.

We define all metrics we analyzed for Fasttack given a graph $G = (\boldsymbol{A}, \boldsymbol{X})$ with set of nodes $\mathcal{V}$, node labels $\boldsymbol{y}$. Further, let $v, w \in \mathcal{V}$ be the target node of an attack, and any other node, respectively. We use $P(v, w)$ as the set of all shortest paths between nodes v and w, and $P_c(u \,|\, v, w)$ as the set of all shortest paths between v and w containing u. Additionally, $\text{path}_{\min}(v, w)$ yields the length of the shortest path between v and w. Lastly, r is given as the degree assortativity of G as in [168]. The absolute metrics are given as follows where the local degree assortativity is taken from [215] and the

PageRank score from [173]:

$$\deg(w) = \sum_j a_{wj} \qquad \text{btw}(w) = \sum_{u \in V \setminus \{w\}} \sum_{t \in V \setminus \{w,u\}} \frac{|P_c(w \,|\, u,t)|}{|P(u,t)|}$$

$$\deg_2(w) = \deg(w) + \sum_{u \in \mathcal{N}(w)} \deg(u) \qquad \text{pgr}(w) = \frac{1}{|\mathcal{N}(w)|} \sum_{u \in \mathcal{N}(w)} \text{pgr}(u)$$

$$\text{asr}(w) = \frac{r+1}{|V|} - \frac{\sum_{u \in \mathcal{N}(w)} |\deg(w) - \deg(u)|}{|\mathcal{N}(w)|} \qquad \text{clc}(w) = \frac{|V| - 1}{\sum_{u \in V \setminus \{w\}} \text{path}_{\min}(w,u)}$$

Further, the relative metrics and feature metrics depend on $\boldsymbol{t}$ as the TFIDF scores for all features and $\boldsymbol{f}$ as the frequencies of all features among nodes sharing the target node's label. They are then defined as follows:

$$\begin{aligned}
\deg \text{ rel}_v(w) &= \deg(w) - \deg(v) \\
\deg_2 \text{ rel}_v(w) &= \deg_2(w) - \deg_2(v) \\
\text{btw rel}_v(w) &= \text{btw}(w) - \text{btw}(v) \\
\text{pgr rel}_v(w) &= \text{pgr}(w) - \text{pgr}(v) \\
\text{clc rel}_v(w) &= \text{clc}(w) - \text{clc}(v) \\
\text{asr rel}_v(w) &= \text{asr}(w) - \text{asr}(v)
\end{aligned}$$

$$\begin{aligned}
\text{sim}_v(w) &= \frac{\boldsymbol{x}_v^T \boldsymbol{x}_w}{||\boldsymbol{x}_v + \boldsymbol{x}_w||_1 - \boldsymbol{x}_v^T \boldsymbol{x}_w} \\
\text{tfi}_v(w) &= (\boldsymbol{x}_v \odot \boldsymbol{x}_w)^T \boldsymbol{t} \\
\text{shr}_v(w) &= \boldsymbol{x}_v^T \boldsymbol{x}_w \\
\text{add}_v(w) &= ||\max(\boldsymbol{x}_w - \boldsymbol{x}_v, 0)||_0 \\
\text{frq}_v(w) &= \frac{\max\{\boldsymbol{x}_w - \boldsymbol{x}_v, 0\}^T \boldsymbol{f}_{y_v}}{||\max\{\boldsymbol{x}_w - \boldsymbol{x}_v, 0\}||_0} \\
\text{leq}_v(w) &= \mathbb{I}[y_v = y_w]
\end{aligned}$$

Algorithm 4: FASTTACK: Scalable adversarial attack on graphs (summary)

Input: Graph $G = (\boldsymbol{A}, \boldsymbol{X})$, target node v_0, modification budget Δ, class predictions $\boldsymbol{Y}$, model $f(\boldsymbol{m} \mid \boldsymbol{\theta})$

Output: Modified Graph $G' = (\boldsymbol{A}', \boldsymbol{X})$

$E_I, S_I \leftarrow$ `valid_edge_insertions_and_scores` $(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{Y}, v_0, f)$;
$E_I, S_I \leftarrow$ `sort_descending` (E_I, S_I);
$E_R, S_R \leftarrow$ `valid_edge_removals_and_scores` $(\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{Y}, v_0, f)$;
$E_R, S_R \leftarrow$ `sort_ascending` (E_R, S_R);
// E_I and E_R are ordered sets of edges to insert and remove; sorted by the scores S_I and S_R
$p_I, p_R \leftarrow 0$; // Pointers to the next best adversarial insertion/removal
$a_n \leftarrow \sum_{s \in S_R} s$; // Numerator of adversarial ratio
$a_d \leftarrow \sum_{s \in S_R} 1 - s$; // Denominator of adversarial ratio
$\boldsymbol{A}' \leftarrow \boldsymbol{A}$;
while $\frac{1}{2}|\boldsymbol{A}' - \boldsymbol{A}| < \Delta$ **do**
  if $\xi'(a_n + S_I[p_I], a_d + 1 - S_I[p_I]) > \xi'(a_n - S_R[p_R], a_d - 1 + S_R[p_R])$ **then**
    // Next perturbation is insertion
    $e \leftarrow E_I[p_I] \quad a'_n \leftarrow a_n + S_I[p_I] \quad a'_d \leftarrow a_d + 1 - S_I[p_I] \quad p_I \leftarrow p_I + 1$
  else
    // Next perturbation is removal
    $e \leftarrow E_R[p_R] \quad a'_n \leftarrow a_n - S_R[p_R] \quad a'_d \leftarrow a_d - 1 + S_R[p_R] \quad p_R \leftarrow p_R + 1$
  if `degree_test_allows_perturbation` $(\boldsymbol{A}', e)$ **then**
    $a_n \leftarrow a'_n \quad a_d \leftarrow a'_d \quad E$;
    $\boldsymbol{A}' \leftarrow$ `flip_edge` $(\boldsymbol{A}', e)$
  else
    // Degree constraint prevents perturbation, try next one
return : $(\boldsymbol{A}', \boldsymbol{X})$

C Adversarial Attacks on Graph Neural Networks via Meta Learning

In this section we present additional results of our experiments. In Table C.1 we see that our heuristic is successful at attacking PUBMED. Tables C.2 and C.3 show misclassification rates with 1% and 10% perturbed edges, respectively. Figures C.1 through C.8 show how the models' classification accuracies change for different attacks and datasets.

	Pubmed		
	GCN	**CLN**	**DeepWalk**
Clean	13.8 ± 0.3	15.9 ± 0.5	21.8 ± 0.1
DICE	15.3 ± 0.1	16.6 ± 0.4	25.1 ± 0.1
A-Meta-Self	16.4 ± 0.2	16.4 ± 0.4	27.4 ± 0.2

Table C.1: Misclassification rate (in %) with 5% perturbed edges on PUBMED when training the surrogate model for $T = 30$ iterations to compute the approximate meta gradients.

	Cora			Citeseer			PolBlogs			Avg.
Attack	**GCN**	**CLN**	**DeepWalk**	**GCN**	**CLN**	**DeepWalk**	**GCN**	**CLN**	**DeepWalk**	**rank**
Clean	16.6 ± 0.3	17.3 ± 0.3	20.3 ± 0.9	28.5 ± 0.8	28.3 ± 0.8	34.8 ± 1.3	6.4 ± 0.5	7.6 ± 0.5	5.3 ± 0.5	6.3
DICE	16.6 ± 0.3	17.6 ± 0.2	20.1 ± 0.2	28.4 ± 0.3	28.4 ± 0.3	35.9 ± 0.3	7.7 ± 0.9	8.5 ± 0.6	7.4 ± 1.0	4.8
First-order	16.6 ± 0.3	17.3 ± 0.1	20.2 ± 0.2	28.2 ± 0.3	28.3 ± 0.4	34.9 ± 0.4	7.0 ± 0.8	7.7 ± 0.5	8.5 ± 1.6	5.7
Nettack*	-	-	-	29.0 ± 0.4	28.6 ± 0.4	36.4 ± 0.4	-	-	-	-
A-Meta-Train	16.3 ± 0.4	18.2 ± 0.2	20.6 ± 0.3	29.1 ± 0.5	28.6 ± 0.5	34.7 ± 0.5	8.9 ± 2.9	10.2 ± 1.9	5.0 ± 0.2	4.7
A-Meta-Both	17.4 ± 0.4	17.6 ± 0.2	21.6 ± 0.3	28.5 ± 0.4	28.3 ± 0.5	34.6 ± 0.3	13.7 ± 1.6	10.4 ± 1.2	7.5 ± 0.6	3.6
Meta-Train	16.2 ± 0.3	18.0 ± 0.3	20.6 ± 0.4	28.3 ± 0.5	28.1 ± 0.6	35.3 ± 0.3	9.4 ± 1.1	10.3 ± 1.7	7.3 ± 2.5	4.8
Meta-Self	17.0 ± 0.4	17.9 ± 0.2	21.1 ± 0.3	29.2 ± 0.5	29.0 ± 0.4	35.2 ± 0.3	11.4 ± 0.4	10.7 ± 1.7	6.6 ± 1.4	2.7
Meta with Oracle	16.2 ± 0.3	18.2 ± 0.2	20.5 ± 0.3	30.1 ± 0.5	29.5 ± 0.5	34.8 ± 0.3	13.6 ± 1.1	10.5 ± 1.2	6.0 ± 0.7	3.5

* Did not finish within three days on CORA-ML and POLBLOGS

Table C.2: Misclassification rate (in %) with 1% perturbed edges.

	Cora			Citeseer			PolBlogs			Avg.
Attack	**GCN**	**CLN**	**DeepWalk**	**GCN**	**CLN**	**DeepWalk**	**GCN**	**CLN**	**DeepWalk**	**rank**
Clean	16.6 ± 0.3	17.3 ± 0.3	20.3 ± 1.0	28.5 ± 0.8	28.3 ± 0.8	34.8 ± 1.3	6.4 ± 0.5	7.6 ± 0.5	5.3 ± 0.5	7.5
DICE	19.5 ± 0.5	19.1 ± 0.2	26.2 ± 0.3	29.7 ± 0.3	29.9 ± 0.3	41.2 ± 0.3	14.4 ± 0.8	14.3 ± 0.6	12.9 ± 0.3	4.9
First-order	17.6 ± 0.5	17.9 ± 0.2	21.5 ± 0.2	28.2 ± 0.3	28.7 ± 0.4	32.4 ± 0.4	7.7 ± 0.6	7.6 ± 0.3	8.2 ± 0.6	7.1
Nettack*	-	-	-	-	-	-	-	-	-	-
A-Meta-Train	28.1 ± 1.1	23.6 ± 0.4	33.6 ± 0.7	34.3 ± 1.1	31.3 ± 0.6	32.1 ± 0.5	12.8 ± 1.6	18.2 ± 2.6	6.9 ± 0.2	4.9
A-Meta-Both	24.6 ± 1.0	20.0 ± 0.3	34.8 ± 0.6	29.1 ± 0.5	29.2 ± 0.4	33.6 ± 0.4	22.7 ± 0.7	22.3 ± 0.9	26.3 ± 1.0	4.8
Meta-Train	37.3 ± 1.4	24.9 ± 0.5	34.4 ± 1.6	31.8 ± 1.0	29.9 ± 0.7	36.0 ± 0.2	28.7 ± 3.6	32.9 ± 1.6	73.7 ± 3.9	2.6
Meta-Self	34.5 ± 0.9	22.9 ± 0.6	37.0 ± 1.0	38.6 ± 1.0	35.3 ± 0.7	36.0 ± 1.2	26.1 ± 0.6	23.5 ± 0.9	60.7 ± 2.7	2.8
Meta with Oracle	34.8 ± 1.5	25.2 ± 0.4	44.0 ± 0.9	40.1 ± 1.2	37.2 ± 0.9	37.2 ± 0.6	28.9 ± 0.4	25.8 ± 0.9	67.1 ± 2.4	1.4

* Did not finish within three days for any dataset.

Table C.3: Misclassification rate (in %) with 10% perturbed edges.

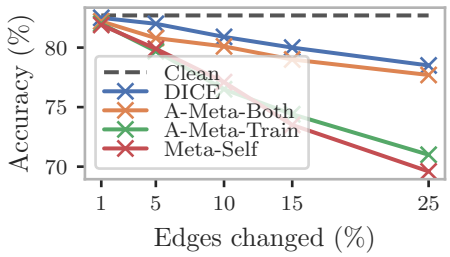


Figure C.1: Change in accuracy of CLN on Cora-ML.

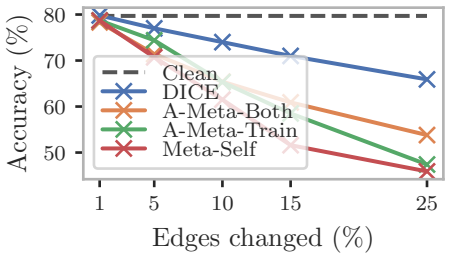


Figure C.2: Change in accuracy of Deepwalk on Cora-ML.

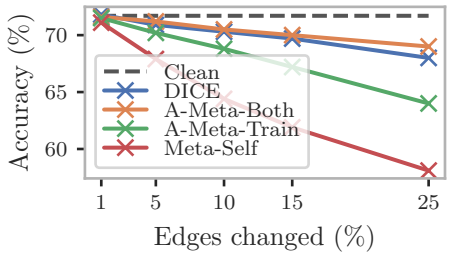


Figure C.3: Change in accuracy of CLN on Citeseer.

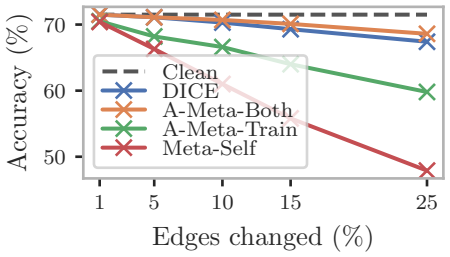


Figure C.4: Change in accuracy of GCN on Citeseer.

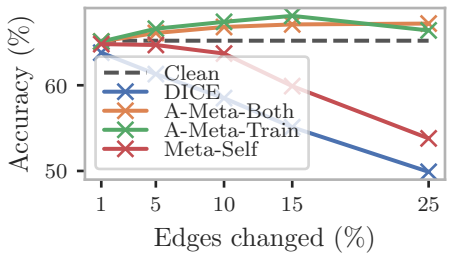


Figure C.5: Change in accuracy of Deepwalk on Citeseer.

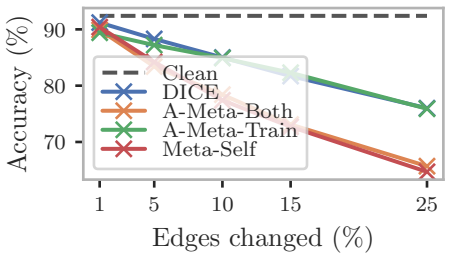


Figure C.6: Change in accuracy of CLN on PolBlogs.

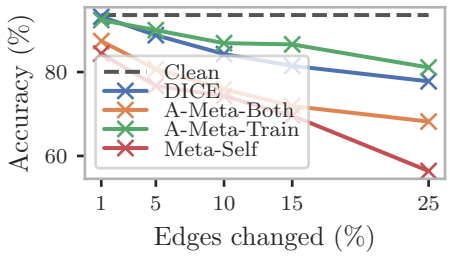


Figure C.7: Change in accuracy of GCN on PolBlogs.

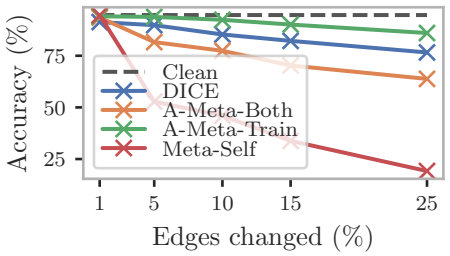


Figure C.8: Change in accuracy of Deepwalk on PolBlogs.

D Certifiable Robustness and Robust Training for Graph Convolutional Networks

Implementation Details We perform the robust training using stochastic gradient descent with mini-batches and Adam Optimizer. For this we randomly sample in each iteration 20 nodes from the labeled nodes (for RH-U from all nodes) and compute the nodes' twohop neighbors. We then slice the adjacency and attribute matrices appropriately and compute the lower/upper activation bounds for all nodes in the batch. We use dropout of 0.5, L_2 regularization with strength $1e-5$, learning rate of 0.001. We use Tensorflow 1.12 and train on NVIDIA GTX 1080 Ti.

D.1 Proofs

We reformulate the problem in Eq. (6.9) as the linear program below.

$$\underset{\tilde{\boldsymbol{X}},\boldsymbol{H}^{(\cdot)},\hat{\boldsymbol{H}}^{(\cdot)},\hat{\varepsilon}}{\text{minimize}} \; \boldsymbol{c}^\top \hat{\boldsymbol{H}}^{(L)} \text{subject to} \tag{D.1}$$

$$\begin{aligned}
&\hat{\boldsymbol{H}}^{(l+1)} = \dot{\boldsymbol{A}}^{(l)} \boldsymbol{H}^{(l)} \boldsymbol{W}^{(l)} + \boldsymbol{b}^{(l)}, l = 2, \dots, L && \Rightarrow \boldsymbol{\Phi}^{(l+1)} \in \mathbb{R}^{\mathcal{N}_{L-l} \times h^{(l)}} \\
&\boldsymbol{H}^{(1)}_{nj} \leq 1 && \Rightarrow \varepsilon^+ \in \mathbb{R}^{\mathcal{N}_{L-1} \times h^{(1)}} \\
&\boldsymbol{H}^{(1)}_{nj} \geq 0 && \Rightarrow \varepsilon^- \in \mathbb{R}^{\mathcal{N}_{L-1} \times h^{(1)}} \\
&\boldsymbol{H}^{(1)}_{nj} \leq \dot{\boldsymbol{X}}_{nj} + \hat{\varepsilon}_{nj} && \Rightarrow \gamma^+ \in \mathbb{R}^{\mathcal{N}_{L-1} \times h^{(1)}} \\
&\boldsymbol{H}^{(1)}_{nj} \geq \dot{\boldsymbol{X}}_{nj} - \hat{\varepsilon}_{nj} && \Rightarrow \gamma^- \in \mathbb{R}^{\mathcal{N}_{L-1} \times h^{(1)}} \\
&\sum_j \hat{\varepsilon}_{nj} \leq q \quad \forall n \in \mathcal{N}_{L-1} && \Rightarrow \boldsymbol{\eta} \in \mathbb{R}^{\mathcal{N}_{L-1}} \\
&\sum_{n,j} \hat{\varepsilon}_{nj} \leq Q && \Rightarrow \rho \in \mathbb{R} \\
&\boldsymbol{H}^{(l)}_{nj} = 0, l = 2, \dots, L-1, (n,j) \in \mathcal{I}^{(l)}_- \\
&\boldsymbol{H}^{(l)}_{nj} = \hat{\boldsymbol{H}}^{(l)}_{nj}, l = 2, \dots, L-1, (n,j) \in \mathcal{I}^{(l)}_+
\end{aligned}$$

$$
\begin{aligned}
&\boldsymbol{H}^{(l)}_{nj} \geq 0, l = 2, \dots, L-1, (n,j) \in \mathcal{I}^{(l)} && \Rightarrow \tau^{(l)} \in \mathbb{R}^{\mathcal{N}_{L-l} \times h^{(l)}} \\
&\boldsymbol{H}^{(l)}_{nj} \geq \hat{\boldsymbol{H}}^{(l)}, l = 2, \dots, L-1, (n,j) \in \mathcal{I}^{(l)} && \Rightarrow \mu^{(l)} \in \mathbb{R}^{\mathcal{N}_{L-l} \times h^{(l)}} \\
&\boldsymbol{H}^{(l)}_{nj} \left(\boldsymbol{S}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj} \right) \leq \boldsymbol{S}^{(l)}_{nj} \left(\hat{\boldsymbol{H}}^{(l)}_{nj} - \boldsymbol{R}^{(l)}_{nj} \right), && \Rightarrow \lambda^{(l)} \in \mathbb{R}^{\mathcal{N}_{L-l} \times h^{(l)}} \\
&\qquad l = 2, \dots, L-1, (n,j) \in \mathcal{I}^{(l)}
\end{aligned}
$$

Note that $\tilde{\boldsymbol{X}} = \boldsymbol{H}^{(1)}$; moreover the $\boldsymbol{H}^{(l)}_{nj} = 0$ and $\boldsymbol{H}^{(l)}_{nj} = \hat{\boldsymbol{H}}^{(l)}$ can be simply eliminated from the optimization. λ, μ, and τ are only defined for $(n,j) \in \mathcal{I}^{(l)}$; we keep the matrix notation for simplicity.

Proof of Theorem 4. Applying standard duality construction, the (non-simplified!) dual problem of the above linear program is

$$
\begin{aligned}
\max & \sum_{l=2}^{L-1} \sum_{(n,j) \in \mathcal{I}^{(l)}} \lambda^{(l)}_{nj} \boldsymbol{S}^{(l)}_{nj} \boldsymbol{R}^{(l)}_{nj} - \sum_{l=1}^{L-1} \mathbf{1}^\top \boldsymbol{\Phi}^{(l+1)} \boldsymbol{b}^{(l)} \\
& + \sum_{n,j} \dot{\boldsymbol{X}}_{nj} \left[\gamma^-_{nj} - \gamma^+_{nj} \right] - \varepsilon^+ - q \sum_n \boldsymbol{\eta}_n - Q\rho \quad \text{subject to}
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{\Phi}^{(L)} = -\boldsymbol{c} \qquad \boldsymbol{\Phi}^{(l)}_{nj} &= 0 && \text{for } l = 2, \dots L, (n,j) \in \mathcal{I}^{(l)}_- \\
\boldsymbol{\Phi}^{(l)} &= \dot{\boldsymbol{A}}^{(l)\top} \boldsymbol{\Phi}^{(l+1)} \boldsymbol{W}^{(l)\top}, && \text{for } l = 2, \dots L, (n,j) \in \mathcal{I}^{(l)}_+ \\
\boldsymbol{\Phi}^{(l)}_{nj} &= \lambda_{nj} \boldsymbol{S}^{(l)}_{nj} - \mu^{(l)}_{nj} && \text{for } l = 2, \dots L, (n,j) \in \mathcal{I}^{(l)} \\
\dot{\boldsymbol{A}}^{(l)\top} \boldsymbol{\Phi}^{(l+1)} \boldsymbol{W}^{(l)\top} &= \lambda^{(l)} \odot \left[\boldsymbol{S}^{(l)} - \boldsymbol{R}^{(l)} \right] \\
&\quad - \tau^{(l)} - \mu^{(l)} && \text{for } l = 2, \dots L, (n,j) \in \mathcal{I}^{(l)} \\
\dot{\boldsymbol{A}}^{(1)\top} \boldsymbol{\Phi}^{(2)} \boldsymbol{W}^{(1)\top} &= \varepsilon^+ - \varepsilon^- + \gamma^+ - \gamma^- && \text{(D.2)} \\
\rho + \boldsymbol{\eta}_n &\geq \gamma^+_{nj} + \gamma^-_{nj} \quad \forall n, j && \text{(D.3)} \\
\lambda, \tau, \mu, &\varepsilon^+, \varepsilon^-, \gamma^+, \gamma^-, \boldsymbol{\eta}, \rho \geq 0
\end{aligned}
$$

As done in [234] we can exploit complementarity of the ReLU constraints corresponding to $\boldsymbol{H}^{(l)} \geq 0$ and $\boldsymbol{H}^{(l)} \geq \hat{\boldsymbol{H}}^{(l)}$ to eliminate τ, μ, and λ from the problem. For this we write

$$
\begin{aligned}
\left[\boldsymbol{S}^{(l)} - \boldsymbol{R}^{(l)} \right] \odot \lambda^{(l)} = \left[\dot{\boldsymbol{A}}^{(l)\top} \boldsymbol{\Phi}^{(l+1)} \boldsymbol{W}^{(l)\top} \right]_+ &= \left[\hat{\boldsymbol{\Phi}}^{(l)} \right]_+ \\
\tau^{(l)} + \mu^{(l)} = \left[\dot{\boldsymbol{A}}^{(l)\top} \boldsymbol{\Phi}^{(l+1)} \boldsymbol{W}^{(l)\top} \right]_- &= \left[\hat{\boldsymbol{\Phi}}^{(l)} \right]_-
\end{aligned}
$$

where we have defined $\hat{\boldsymbol{\Phi}}^{(l)} := \dot{\boldsymbol{A}}^{(l)\top} \boldsymbol{\Phi}^{(l+1)} \boldsymbol{W}^{(l)\top}$. Given the non-negativity of the dual-variables, it becomes apparent that $\tau^{(l)}$ and $\mu^{(l)}$ "share" the negative part of $\hat{\boldsymbol{\Phi}}^{(l)}$. Thus,

we define new variables $\boldsymbol{\Omega}_{nj}^{(l)} \in [0,1]$ such that $\mu_{nj}^{(l)} = \boldsymbol{\Omega}_{nj}^{(l)} \left[\hat{\boldsymbol{\Phi}}_{nj}^{(l)}\right]_-$. Combining this with the constraint $\boldsymbol{\Phi}_{nj}^{(l)} = \lambda_{nj}\boldsymbol{S}_{nj}^{(l)} - \mu_{nj}^{(l)}$ we can rephrase to get

$$\boldsymbol{\Phi}_{nj}^{(l)} = \frac{\boldsymbol{S}_{nj}^{(l)}}{\boldsymbol{S}_{nj}^{(l)} - \boldsymbol{R}_{nj}^{(l)}} \left[\hat{\boldsymbol{\Phi}}_{nj}^{(l)}\right]_+ - \boldsymbol{\Omega}_{nj}^{(l)} \left[\hat{\boldsymbol{\Phi}}_{nj}^{(l)}\right]_-$$
$$\lambda_{nd}^{(l)} = \left[\hat{\boldsymbol{\Phi}}_{nd}^{(l)}\right]_+ \cdot \left(\boldsymbol{S}_{nj}^{(l)} - \boldsymbol{R}_{nj}^{(l)}\right)^{-1}$$

Similarly, by complementarity of the constraints, we know that only one of ε_{nj}^+ and ε_{nj}^- and only one of γ_{nj}^+ and γ_{nj}^- can be positive. From Eq. (D.2) we can therefore see that ε_{nj}^+ and γ_{nj}^+ need to "share" the positive part of the left hand side in Eq. (D.2) (since all variables are ≥ 0); similarly ε_{nj}^- and γ_{nj}^- share the negative part. We denote this (unknown) share by a new variable $\beta_{nj} \in [0,1]$ and get

$$\varepsilon_{nj}^+ = \beta_{nj}\left[\hat{\boldsymbol{\Phi}}_{nj}^{(1)}\right]_+, \qquad \gamma_{nj}^+ = (1-\beta_{nj})\left[\hat{\boldsymbol{\Phi}}_{nj}^{(1)}\right]_+$$
$$\varepsilon_{nj}^- = \beta_{nj}\left[\hat{\boldsymbol{\Phi}}_{nj}^{(1)}\right]_-, \qquad \gamma_{nj}^- = (1-\beta_{nj})\left[\hat{\boldsymbol{\Phi}}_{nj}^{(1)}\right]_-$$

Putting this into Eq. (D.3) we can now see that

$$\boldsymbol{\eta}_n + \rho \geq (1-\beta_{nj})|\hat{\boldsymbol{\Phi}}_{nj}^{(1)}| \quad \forall 1 \leq n \leq \mathcal{N}_{L-1}, 1 \leq j \leq h^{(1)},$$

from which we can get

$$\beta_{nj} \geq 1 - \frac{\rho + \boldsymbol{\eta}_n}{|\hat{\boldsymbol{\Phi}}_{nj}^{(1)}|}, \beta_{nj} \geq 0 \quad \forall n, j \tag{D.4}$$

to replace the constraint in Eq. (D.3). Now we can simplify the following term from the dual objective

$$\begin{aligned}\dot{\boldsymbol{X}}_{nj}\left[\gamma_{nj}^- - \gamma_{nj}^+\right] - \varepsilon_{nj}^+ &= -\hat{\boldsymbol{\Phi}}_{nj}^{(1)}\dot{\boldsymbol{X}}_{nj} + \beta_{nj}\hat{\boldsymbol{\Phi}}_{nj}^{(1)}\dot{\boldsymbol{X}}_{nj} - \beta_{nj}\left[\hat{\boldsymbol{\Phi}}_{nj}^{(1)}\right]_+ \\ &= -\hat{\boldsymbol{\Phi}}_{nj}^{(l)}\dot{\boldsymbol{X}}_{nj} - \boldsymbol{\Delta}_{nj}\beta_{nj} \text{ where} \\ \boldsymbol{\Delta}_{nj} &:= \left[\hat{\boldsymbol{\Phi}}_{nj}^{(1)}\right]_+ \cdot (1 - \dot{\boldsymbol{X}}_{nd}) + \left[\hat{\boldsymbol{\Phi}}_{nj}^{(1)}\right]_- \cdot \dot{\boldsymbol{X}}_{nd}.\end{aligned}$$

In the definition of $\boldsymbol{\Delta}_{nj}$ we essentially have a case distinction: if $\hat{\boldsymbol{\Phi}}_{nj}^{(1)}$ is positive, we know that increasing the value of the corresponding primal variable $\tilde{\boldsymbol{X}}_{nd}$ will improve the primal objective. If $\dot{\boldsymbol{X}}_{nd}$ is already 1, however, we set the value of $\boldsymbol{\Delta}_{nj}$ to zero by multiplying by $1 - \dot{\boldsymbol{X}}_{nd}$ (similarly for the case when $\hat{\boldsymbol{\Phi}}_{nj}^{(1)}$ is negative). This effectively enforces the $0 \leq \tilde{\boldsymbol{X}} \leq 1$ constraint on the perturbations.

Plugging all terms defined above into our dual objective we get

$$\begin{aligned}\max \sum_{l=2}^{L-1} \sum_{(n,j)\in\mathcal{I}^{(l)}} \frac{\boldsymbol{S}_{nj}^{(l)}\boldsymbol{R}_{nj}^{(l)}}{\boldsymbol{S}_{nj}^{(l)} - \boldsymbol{R}_{nj}^{(l)}} \left[\hat{\boldsymbol{\Phi}}_{nj}^{(l+1)}\right]_+ - \sum_{l=1}^{L-1} \mathbf{1}^\top \boldsymbol{\Phi}^{(l+1)}\boldsymbol{b}^{(l)} \\ - \operatorname{Tr}\left[\dot{\boldsymbol{X}}^\top \hat{\boldsymbol{\Phi}}^{(1)}\right] - \|\boldsymbol{\Delta} \odot \beta\|_1 - q \cdot \sum_n \boldsymbol{\eta}_n - Q \cdot \rho\end{aligned}$$

Notice that $\mathrm{Tr}\left[\dot{\boldsymbol{X}}^\top \hat{\boldsymbol{\Phi}}^{(1)}\right] = \sum_n \sum_j \hat{\boldsymbol{\Phi}}_{nj} \dot{\boldsymbol{X}}_{nj}$. Since $\boldsymbol{\Delta} \geq 0$ and $\beta \geq 0$ we could safely write $\|\boldsymbol{\Delta} \odot \beta\|_1$. By observing that $\boldsymbol{\Delta} \geq 0$ for all entries we see that to maximize the objective, we will set β to a value as small as is admissible. This means we can replace Eq. (D.4) with $\beta = \max\left\{1 - \frac{\rho+\boldsymbol{\eta}_n}{|\hat{\boldsymbol{\Phi}}^{(1)}_{nj}|}, 0\right\}$. Thus, we have now eliminated all dual variables except $\boldsymbol{\Omega}$, ρ, and $\boldsymbol{\eta}_n$. Finally, we define $\boldsymbol{\Psi}_{nj} := \boldsymbol{\Delta}_{nj}\beta_{nj} = \max\{\boldsymbol{\Delta}_{nj} - (\boldsymbol{\eta}_n + \rho), 0\}$, which finalizes the proof. □

Proof of Theorem 5. Given a fixed $\boldsymbol{\Omega}$, the dual function $g^t_{q,Q}$ reduces to $-\|\boldsymbol{\Psi}\|_1 - q \cdot \sum_n \boldsymbol{\eta}_n - Q \cdot \rho + const$ with the term $\boldsymbol{\Psi}_{nd} = \max\{\boldsymbol{\Delta}_{nd} - (\boldsymbol{\eta}_n + \rho), 0\}$ and $\boldsymbol{\Delta}_{nd}$ constant. Noticing that $\boldsymbol{\Psi}_{nd} \geq 0$, we see that maximizing the dual is equivalent to minimizing

$$\min_{\rho, \boldsymbol{\eta}_n \geq 0} h(\rho, \boldsymbol{\eta}_n) = \sum_{n,d} \max\{\boldsymbol{\Delta}_{nd} - (\boldsymbol{\eta}_n + \rho), 0\} + q \cdot \sum_n \boldsymbol{\eta}_n + Q \cdot \rho$$

Observe that we can equivalently rephrase this as

$$\min_{\rho, \boldsymbol{\eta}_n, \boldsymbol{U}_{nd} \geq 0} h'(\rho, \boldsymbol{\eta}_n, \boldsymbol{U}) = \sum_{n,d} \boldsymbol{U}_{nd} + q \cdot \sum_n \boldsymbol{\eta}_n + Q \cdot \rho$$
$$s.t.\ \boldsymbol{U}_{nd} \geq \boldsymbol{\Delta}_{nd} - \boldsymbol{\eta}_n - \rho$$

Here we have replaced $\max\{\boldsymbol{\Delta}_{nd} - (\boldsymbol{\eta}_n + \rho), 0\}$ in $h(\rho, \boldsymbol{\eta}_n)$ with a new variable $\boldsymbol{U}_{nd}$ with the constraints $\boldsymbol{U}_{nd} \geq 0$ and $\boldsymbol{U}_{nd} \geq \boldsymbol{\Delta}_{nd} - \boldsymbol{\eta}_n - \rho$ (for each $1 \leq n \leq \mathcal{N}_{L-1}, 1 \leq d \leq h^{(1)}$). Since we are minimizing, the optimal values w.r.t. $h'(\rho, \boldsymbol{\eta}_n, \boldsymbol{U})$ and $h(\rho, \boldsymbol{\eta}_n)$ will be the same.

Finding the minimum of $h'(\cdot)$ is a linear program. Thus, we can again form its dual (denoting the dual variables as α_{nd}):

$$\max_{\alpha_{nd} \geq 0} g'(\alpha_{nd}) = \sum_{n,d} \boldsymbol{\Delta}_{nd} \alpha_{nd}$$
$$s.t.\ \alpha_{nd} \leq 1, \qquad \sum_{n,d} \alpha_{nd} \leq Q, \qquad \sum_d \alpha_{nd} \leq q\ \forall n$$

An optimal solution of this dual can be seen and computed easily. Since all $\boldsymbol{\Delta}_{nd}$ are nonnegative, we simply set those α_{nd} to 1 corresponding to the largest values of $\boldsymbol{\Delta}_{nd}$ – additionally taking the two other constraints into account: The third constraint tells us that the row sums of the α matrix can be at most q, hence we can only set the α_{nd} corresponding to the q largest $\boldsymbol{\Delta}_{nd}$ to 1 for each row to maximize the objective. The second constraint means that we can set at most Q entries α_{nd} to 1. So among the set of all q largest $\boldsymbol{\Delta}_{nd}$ of the rows we select again the Q largest $\boldsymbol{\Delta}_{nd}$ and set their corresponding α_{nd} to 1[1]. Observe that this is precisely the selection process described in the main text for Thm. 5. That is, an optimal solution of the dual can be found by setting $\alpha_{nd} = 1 \Leftrightarrow (n, d) \in S_Q$.

[1] W.l.o.g. we assume $Q \leq |\mathcal{N}_{L-1}| \cdot q$ here; otherwise we simply select all of the q largest $\boldsymbol{\Delta}_{nd}$ per row (which is equivalent to choosing $Q = |\mathcal{N}_{L-1}| \cdot q$)

We now prove that the variables ρ and $\boldsymbol{\eta}_n$ as described in the main text (along with $\boldsymbol{U}_{nd} = \max\{\boldsymbol{\Delta}_{nd} - \boldsymbol{\eta}_n - \rho, 0\}$) correspond to an *optimal* solution of their respective problem. For this we show that the Karush-Kuhn-Tucker (KKT) conditions hold – using the above constructed solution for α_{nd}. (1) Dual and primal feasibility hold by construction. (2) Next, we check complementary slackness

$$\alpha_{nd}(\boldsymbol{\Delta}_{nd} - \boldsymbol{\eta}_n - \rho - \boldsymbol{U}_{nd}) = 0 \tag{D.5}$$

If $\alpha_{nd} > 0$ it must hold that the second term is 0. In this case we know that $\boldsymbol{\Delta}_{nd} \geq \boldsymbol{\eta}_n + \rho$ and thus $\boldsymbol{U}_{nd} = \Delta_{nd} - \boldsymbol{\eta}_n - \rho$, which means the term is always 0. When the second term in Eq. (D.5) is nonzero, α_{nd} must be 0. This is given since when $\boldsymbol{\Delta}_{nd} < \boldsymbol{\eta}_n + \rho + \boldsymbol{U}_{nd}$ it is smaller than the smallest $\boldsymbol{\Delta}_{nd}$ for which α_{nd} is set to 1 and therefore $\alpha_{nd} = 0$. (3) Finally we show that $\nabla_\theta L(\theta, \alpha) = \nabla_\theta \sum_{n,d} \boldsymbol{U}_{nd} + q \cdot \sum_n \boldsymbol{\eta}_n + Q \cdot \rho + \sum_{n,d} \alpha_{nd}(\boldsymbol{\Delta}_{nd} - \boldsymbol{\eta}_n - \rho - \boldsymbol{U}_{nd}) = 0$ for $\theta = \{\boldsymbol{U}_{nd}, \boldsymbol{\eta}_n, \rho\}$. Consider first ρ: $\nabla_\rho L(\theta, \alpha) = Q - \sum_{n,d} \alpha_{nd} = 0$ since we set exactly Q many α_{nd} to 1 (and the rest to 0); $\boldsymbol{\eta}_n$ follows analogously. $\nabla_{\boldsymbol{U}_{nd}} L(\theta, \alpha) = \mathbb{I}_{\boldsymbol{U}_{nd}>0}(1 - \alpha_{nd}) = 0$ holds since when $\alpha_{nd} = 0$, $\boldsymbol{\Delta}_{nd} - \boldsymbol{\eta}_n - \rho \leq 0$ which means that $\boldsymbol{U}_{nd}$ must be 0 because of its constraints. Thus, all KKT conditions hold. □

Proof of Corollary 2. Assume we are given the optimal values for $\boldsymbol{\Omega}$. We can then compute the optimal values of ρ and $\boldsymbol{\eta}$ as described in Theorem 5. Recall from the proof of Thm. 4 that the $\boldsymbol{\Delta}_{nd}$ denote the improvement in the primal function objective when changing the attribute $\dot{\boldsymbol{X}}_{nd}$. As shown in the proof of Thm. 5 with the optimal α_{nd} we exactly choose the values $\boldsymbol{\Delta}_{nd}$ that lead to the largest improvement of the objective function – and we trivially observed $\alpha_{nd} = 1$ for those elements. Thus, an optimal solution can be obtained by perturbing the attribute entries $\dot{\boldsymbol{X}}_{nd}$ from the set $P := \{(n, d) | \alpha_{nd} = 1, \boldsymbol{\Delta}_{nd} > 0\}$, i.e., setting them to $1 - \dot{\boldsymbol{X}}_{nd}$. Thus, by construction we found an optimal solution which is integral, making the original linear program integral w.r.t. the attributes $\tilde{\boldsymbol{X}}_{nd}$. By construction of α_{nd} the set $P = \{(n, d) \in S_Q \mid \boldsymbol{\Delta}_{nd} > 0\}$. □

Proof of Corollary 3. Using Eq. (6.3), the (un-perturbed) $\hat{\boldsymbol{H}}^{(2)}_{mj}$ is $\dot{\boldsymbol{A}}^{(1)}_{m:} \dot{\boldsymbol{X}} \boldsymbol{W}^{(1)}_{:j} + \boldsymbol{b}^{(1)}_j = \sum_n \sum_d \dot{\boldsymbol{A}}^{(1)}_{mn} \dot{\boldsymbol{X}}_{nd} \boldsymbol{W}^{(1)}_{dj} + \boldsymbol{b}^{(1)}_j$ which is simply a linear sum in $\dot{\boldsymbol{X}}_{nd}$. Clearly, for maximizing $\hat{\boldsymbol{H}}^{(2)}_{mj}$, one should only perturb $\dot{\boldsymbol{X}}_{nd}$ if ($\dot{\boldsymbol{A}}^{(1)}_{mn} \boldsymbol{W}^{(1)}_{dj}$ is positive and $\dot{\boldsymbol{X}}_{nd} = 0$) or ($\dot{\boldsymbol{A}}^{(1)}_{mn} \boldsymbol{W}^{(1)}_{dj}$ is negative and $\dot{\boldsymbol{X}}_{nd} = 1$). Thus, the maximal *increase* of $\hat{\boldsymbol{H}}^{(2)}_{mj}$ based on $\dot{\boldsymbol{X}}_{nd}$ one can achieve is $\dot{\boldsymbol{A}}^{(1)}_{mn} \cdot \boldsymbol{W}^{(1)}_{dj}$ if the first condition holds, $-\dot{\boldsymbol{A}}^{(1)}_{mn} \cdot \boldsymbol{W}^{(1)}_{dj}$ if the second holds, and 0 else. This can compactly be written as $\dot{\boldsymbol{A}}^{(1)}_{mn} \cdot ([\boldsymbol{W}^{(1)}_{dj}]_+ \cdot (1 - \dot{\boldsymbol{X}}_{nd}) + [\boldsymbol{W}^{(1)}_{dj}]_- \cdot \dot{\boldsymbol{X}}_{nd})$, which matches the terms in Eq. (6.12). To obtain the maximal *overall* increase in $\hat{\boldsymbol{H}}^{(2)}_{mj}$, and, thus, an upper bound, one simply picks the largest elements that still adhere to the budget constraints (Q,q). Obviously, since this is an admissible perturbation, this upper bound is tight. The proof for the lower bound is accordingly. □

E Certifiable Robustness of Graph Convolutional Networks under Structure Perturbations

E.1 Splitting and Branching Procedure

We follow Al-Khayyal and Falk [3]'s suggested procedures for splitting and branching. That is, for deciding on a problem to branch on, we choose the problem with the smallest lower bound among all open problems. To determine a dimension for the split, we choose the dimension for which the convex envelope has the largest difference to the corresponding value in the optimal solution, i.e. where the approximation is the coarsest. We refer the interested reader to [3] for more details.

E.2 Experimental setup

For all experiments we use Python 3.6.8. For the GCN training we use the hyperparameters shown in Table E.1. For the robust training procedure proposed in [277] we use a local budget $q = 0.01D$ and global budget $Q = 12$, and a batch size of 8 (as suggested by the authors), and train for 1000 epochs. We use NVIDIA (R) GTX 1080 GPUs with 11 GB VRAM and use PyTorch 1.0.1. For all datasets we sample 500 nodes uniformly across the graph and certify robustness on this representative subset.

We use the IBM CPLEX solver[1] for all linear programs, though any optimizer can be used. We further use CVXPY as the interface to Python. We run the solver on a single core on an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz. We abort our branch and bound procedure after 250 iterations and treat the instance as not certifiable.

[1] https://en.wikipedia.org/wiki/CPLEX

Hyperparameter	Value
Learning rate	0.01
Hidden size	32
Hidden layers	1
Dropout	0.5
Training epochs	350
Training set size	10%
Weight decay	0.0001

Table E.1: Hyperparameter configuration