The impact of Graph Neural Network task types on the stability of Graph Neural Networks in face of perturbations.

A coded experiment on GNN stability

Vladimir Rullens[1]

Supervisor(s): Elvin Isufi[1]**, Maosheng Yang**[1]**, Mohammad Sabbaqi**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Vladimir Rullens
Final project course: CSE3000 Research Project
Thesis committee: Elvin Isufi, Maosheng Yang, Mohammad Sabbaqi, Klaus Hildebrandt

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

Abstract

Graph Neural Networks (GNN) are Machine Learning models which are trained on graph data in order to handle complex state-of-the-art tasks such as recommender systems and molecular property prediction. However, the graphs that these models are trained on can be perturbed in various ways post training resulting in reductions in performance. This study compares the stability of various Graph Neural Network task types (Node Classification, Link Prediction, and Graph Classification) by investigating each of their performances across a range of perturbation severities performed on graphs. Further experiments explore whether this performance ranking changes for different types of perturbations and GNN architectures. Through results, it is shown that there is a noticeable difference in stability between the investigated tasks. However, it is also shown that under certain conditions, such as different types of perturbations or architectures, the performance ranking may shift. This paper highlights weak points in GNNs that should be explored for stronger defenses against potential attacks.

1 Introduction

Graph Neural Networks, GNNs for short, are a field in Deep Learning where graphs, containing nodes for objects and edges connecting these nodes, are utilized to handle complex problems like recommender systems and molecular property prediction [1, 2, 3, 4]. This is thanks to graphs being able to handle non-Euclidean data, such as social networks [4]. These GNNs tend to make use of Neural Message Passing to perform computations on these graphs, where each node sends their neighbors a ”message” related to i.e. the current node, the edge type and their neighbor, depending on the utilized GNN architecture [5]. The recipient then groups all received messages in some way, and then uses that result to update their current vector [5]. Combining this formula with neural networks allows the GNN to determine a node's structural relation to the rest of the graph [6], allowing for various tasks to be executed such as:

1. Node classification, where nodes are classified individually. [1, 7].
2. Graph classification, where graphs are classified as a whole, like in earlier mentioned molecular property predictions [1, 7].
3. Link prediction, where the existence of a link between edges is predicted, like in earlier mentioned recommender systems [1, 7].
4. Node clustering or Community Detection, where nodes are divided into various clusters based on their similarity to objects in the rest of the graph [1, 7].
5. Influence maximization, where the most influential nodes are detected. [1, 7].

However, these graphs can be influenced by third parties in the form of adverserial attacks, i.e. to change a node's classification to no longer be seen as malicious. This is also considered a ”perturbation” [8]. A question then revolves around the stability of GNNs: how do these GNNs fare against perturbations? Do small perturbations heavily influence their accuracy?

For this topic, some research has been done on types of attacks which can be performed, i.e. Random Walk Column Sum mentioned in [9], which selects nodes which affect classification loss the most through feature perturbations. Some properties which allow GNNs to be more resistant have also been discussed, such as the notion of permutation equivariance mentioned in [10]. Finally, edge removal/perturbation effects have also been compared in the realm of Node Classification, such as in [11]. However, there is still research to be done on which GNN design choices affect GNN stability, as well as more research into the effects on other task types. This could indicate which areas focus should be directed towards the most in the pursuit of improving GNN stability.

This paper will attempt to find GNN stability weakpoints in one aspect of GNNs, namely the choice of the earlier mentioned task types. Here, the main research question that will be tackled is:

How does choice of task types, namely node classification, link prediction, graph classification, impact the stability of a GNN in the face of perturbations?

For this research question the included subquestions are as follows:

Research focused:

- What characteristics of GNN tasks have already been established to affect the stability of GNNs?
- What GNN architectures are utilized to handle the related tasks?

Experiment focused:

- How do the established GNN task types perform compared to each other in terms of stability, when applying the same perturbations to each GNN?
- What core characteristics of these tasks may influence stability based on the established ranking?

Shared:

- What is missing in the state-of-the-art literature discussing the stability of GNNs?
- What is the impact of GNN hyperparameters on the stability of GNNs across the varying task types?

The goal of this research paper is to highlight how much GNN stability is impacted by a change in GNN tasks. Through this it will be shown which tasks are most vulnerable to perturbations, and thereby which areas are in need of the most attention in regards to their improvement. It may also show how consistent GNN architectures are across the varying task types. In Chapter 2, some background information is given to highlight parts of the experiment, detailing the idea

Task	Dataset	Backbone	Graphs	Nodes	Edges	Features	Classes
Node Classification	Planetoid	Pubmed	1	19717	88648	500	3
Graph Classification	TUDatasetEnzymes	MUTAG	188	$\tilde{1}$7.9	$\tilde{3}$9.6	7	2
Link Prediction	Planetoid	Cora	1	2708	10556	1433	7

Table 1: Graphs utilized in research

behind some aspects which are further described in Chapter 3. In Chapter 3, the experiment itself is described, detailing the utilized GNN architectures, perturbation types, and their expected results, as well as their reasoning. In Chapter 4, the environment setup of the experiment will be described alongside its results. In Chapter 5, the ethics behind this research will be described and the reproducibility of this research is discussed. In Chapter 6, the results will be discussed further. Finally, in Chapter 7, the conclusions of this research are reported, as well as any potential future research goals.

2 Methodology

2.1 Graphs

For this research, Pytorch Geometric[1], a Python library dedicated to graph neural networks, is utilized. This library contains a simple format for building GNNs. The documentation page also contains a list of referenced GNN architectures and a list of referenced graph datasets. For every task, a graph dataset is chosen from Pytorch Geometric's dataset library which matches it. This includes the graphs shown in Table 1.

All chosen graphs are homogeneous: There is only one type of node and only one type of edge. Graph classification has notably fewer features for its feature count, which is a characteristic that can be seen on most graph-focused datasets in Pytorch Geometric. These numbers can be found on Pytorch Geometric's dataset cheatsheet[2].

2.2 GNN architectures

Utilizing these graphs, a GNN model is trained on the graph, which is consistent across all tasks. The used GNN models here are the Graph Convolutional Network (GCN) and the Graph Attention Network (GAT).

- Graph Convolutional Networks are a spectral method which utilizes normalized graph Laplacians and first-order Fourier transforms to calculate a node's neighborhood position [12]. Due to its spectral nature, its learning is dependent on the graph's structure [13]. Kipf and Welling [12] derived their neural network propagation rule for Graph Convolutional Networks:

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

  In this equation $H^{(l+1)}$ indicates the features at the l-th layer, with the 0-th layer being equal to the initial node features. $\tilde{A}$ is equal to the adjacency matrix of the graph with added self-connections, while $\tilde{D}$ is the degree matrix. Finally, W is the trainable weight matrix of layer l, with σ being the activation function.

  The GNN layout utilized for the course of this experiment is a 3-layer GCN (Computing a neighborhood depth of 3 nodes) with 64 hidden channels (variables) per GNN layer.

- Graph Attention Networks are a non-spectral method which utilizes a set of k "attention heads" serving as edge weights to learn a node's neighborhood locally [13]. Here an attention mechanism calculates the importance of one node's features compared to another by acting as a neural network.

  Velickovic et al. [13] note the GAT architecture as described here. A neural network a, called a self-attention mechanism, can compute attention coefficients as follows:

  $$\alpha_{ij} = \text{softmax}_j(a(Wh_i, Wh_j)) \quad (2)$$

  Here, a is the self-attention mechanism, with W being the weight matrix and h being a set of features for node i/j. Computed is the attention coefficient between two nodes, which serves as an edge weight. The full function for calculating attention coefficients within the GAT framework is then as follows:

  $$\alpha_{ij} = \frac{\text{exp}(\text{LeakyReLU}(a^T[Wh_i||Wh_j]))}{\sum_{k\in N(i)}\text{exp}(\text{LeakyReLU}(a^T[Wh_i||Wh_k]))} \quad (3)$$

  In this equation $N(i)$ is the set of nodes within the current node's neighborhood. The above α_{ij} is utilized in the following function for multi-head attention networks, where $\acute{h}_i$, indicating the features at the next layer, is computed as follows:

  $$\acute{h}_i = \overset{K}{\underset{k=1}{||}} \sigma(\sum_{j\in N(i)} \alpha^k_{i,j}W^k h_j) \quad (4)$$

  The concatenation symbol || indicates concatenation over the k available attention heads (here 5).

  This method does not rely on the graph's global structure [13]. The GNN layout utilized for the course of this experiment is a 3-layer GAT with 64 hidden channels and 5 attention heads. This multi-head attention structure was deemed to allow the learning process to be more stabilized [13].

[1] https://pytorch-geometric.readthedocs.io/en/latest/

[2] https://pytorch-geometric.readthedocs.io/en/latest/cheatsheet/data_cheatsheet.html

2.3 Task type implementations

While the GNN structure is consistent across all task types, different tasks require these GNNs to be used differently.

- For Node Classification, the GNN model is followed by a linear classifier, which takes the previous hidden layers and classifies them in the output layer between the x available classes in the dataset (here 3).
- For Graph Classification, the GNN model is followed by the global_mean_pool method, which pools all final node embeddings into one averaged embedding [14]. This singular node embedding is then classified between the x available classes through a linear classifier.
- For Link Prediction, a Graph Auto-Encoder (GAE) is utilized. Graph Auto-Encoders encode the node features of a graph by putting the graph through a GNN. Utilizing this encoded format the adjacency matrix of the graph is reconstructed to obtain all potential edges, to which the training/test sets can be compared [15].

  For Link Prediction, this experiment utilizes an existing implementation[3], which creates predictions as float values rather than a binary 1/0 for whether or not there is a link. To accompany this, the original implementation utilized the roc_auc_score, which is a threshold-free evaluation metric [16]. However, in order to reliably compare it to the other tasks this was converted to a threshold-dependent metric, namely the micro-averaged F1 score, the same one used for the other tasks. To make this work, a threshold had to be picked for the predicted float values. Since these float values range from 0.5 to 1.0, a straight-forward threshold was taken at 0.75, in order to give a fair split between links and non-links. This is to avoid overfitting to the dataset with a trained threshold.

2.4 Perturbation methods

After training is finalized, the graph is perturbed for a range of perturbation severities, i.e. randomly removing 0%, 10%, 20%... of all edges on the graph. The perturbation methods utilized are a variety of perturbation types: Edge Removal, Node Removal, and Edge Rewiring, where each method is done in a randomized format.

Edge Removal and Node Removal here are self-explanatory: A percentage of all edges/nodes on the graph are directly removed, giving nodes less of their neighborhood to work with. The randomization is determined by random.choice, which is accessible by importing random. In this experiment, this generated a list of size 'node/edge count' with x̃% of all kept nodes/edges being set to True, and the rest being False.

Edge Rewiring keeps all nodes, only modifying the placement of edges while making sure the number of edges each node has (the degree of the node) stays the same. The utilized method is similar to an attack performed by Ma et al. [17], which showcased great results. To describe the process: For a single perturbation 2 edges are taken, spanning across nodes A and B for edge 1, and nodes C and D for edge 2. If no connection yet exists for AC and BD, the edges between AB and CD are moved to these new positions. If AC or BD is already taken, AD and BC are also checked for the same effect. This is done on $x\%$ of all edges. This method keeps node features in check and still gives the GNN a neighborhood to work with, but forces the GNN to classify on incorrect neighborhoods. Each perturbation method here is re-performed on the original dataset for every trial.

The performance of the already trained GNN is then evaluated on both the initial graph and its perturbed versions, after which their results are compared. By doing this for each task type over a range of perturbation strengths, a graph figure will be obtained which showcases the degradation in performance for each task type as perturbations become more severe, allowing for insights on which task types are influenced most by perturbations, as per the main research question defined in Section 1.

3 Experiments

The following experiments are focused on comparing the performance of task types while utilizing micro F1 loss as the metric, where micro F1 loss utilizes the following function [18]:

$$\frac{\sum TP}{\sum TP + \frac{1}{2}(\sum FP + \sum FN)} \tag{5}$$

Here, TP, FP, FN refer to True Positive, False Positive, and False Negative respectively, with the summation being done over all classes.

3.1 Initial Task Type Performance Comparison

In the first experiment, all task types are compared to one another under one GNN architecture and perturbation method. Here, the GCN architecture is utilized as it runs faster than the GAT architecture mentioned in Section 2 while giving similar initial performance results. To reiterate, the utilized GCN architecture contains 3 layers with 64 hidden channels per layer. The chosen perturbation method is Edge Removal. The goal of this perturbation method is to remove connections between nodes, resulting in GNNs being given fewer nodes to work within a node's neighborhood.

3.2 Perturbation Effect on Performance Comparison

In the second experiment, more perturbation methods are tested to see if the chosen perturbation method has an effect on the ranking between task types. The remaining two perturbation methods explained in Section 2 are utilized here, Node Removal and Edge Rewiring. Edge Rewiring is deemed the most interesting here as it is a significantly different type of perturbation, which tampers with a GNN's neighborhood assessment directly. This is due to the fact that node connections are greatly altered, which results in each node having connections to new (incorrect) neighborhoods. This is different from what happens in the other perturbation methods where nodes tend to get isolated, subjecting them more to

[3] https://github.com/AntonioLonga/PytorchGeometricTutorial/blob /main/Tutorial12/Tutorial12%20GAE%20for%20link%20prediction .ipynb

their initial set of features. As a result, the stability of the individual task types is expected to be impacted differently from before.

3.3 Architecture Effect on Performance Comparison

In the third experiment, architectures are compared to see if these have an effect on the ranking between task types. Here the GAT is included in the research and compared against the GCN's initial results. The provided GAT architecture contained 3 layers with 64 hidden channels per layer and 5 attention heads. This was chosen to keep the number of layers and hidden channels consistent while utilizing the GAT's main feature, the attention head. As the GAT is a non-spectral architecture, and thereby does not rely on the graph's global structure [13], results are expected to be different.

4 Setup and Results

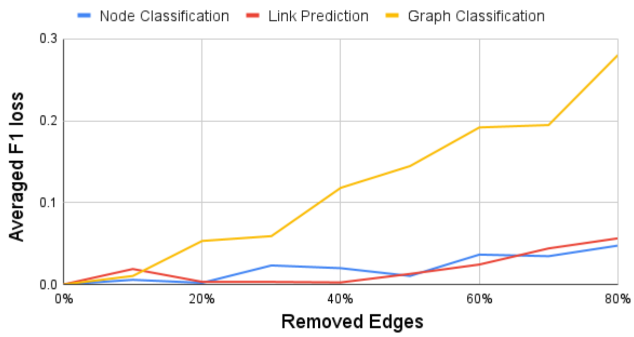

Figure 1: GCN micro F1 loss for a range of Edge Removal strengths on various tasks. 25 trials per data point, seperated by 10% each.

The first experiment compared the various tasks among one another for a given form of perturbation: Randomized Edge Removal. This perturbation was done by obtaining a randomized True/False list utilizing random.choice from the 'random' import, where the length of the list equals the number of edges available. Here the percentage of kept edges (True) and lost edges (False) could be specified, and the False edges were removed by their ID. The databases utilized are specified in Section 2, Table 1. The experiments were conducted 25 times per measurement point, utilizing a GCN with 3 layers, each with 64 hidden channels. For every measurement point, the GNN was retrained with a new perturbed dataset to test on. Results are evaluated on a 6-core Intel(R) Core(TM) i7-9750H CPU with 16GB memory and an NVIDIA Quadro P2000 GPU. The goal of this experiment is to identify whether there is a difference in stability between the given tasks.

In Figure 1 the effect of Edge Removal can be seen on the tasks Node Classification, Graph Classification, and Link Prediction. It can be seen that Graph Classification has a significantly greater loss than Node Classification and Link Prediction, never falling below its opponents. On the other hand, both Node Classification and Link Prediction produce similar results, roughly matching one another in terms of accuracy lost. For every task, the showcased trend appears to be linear.

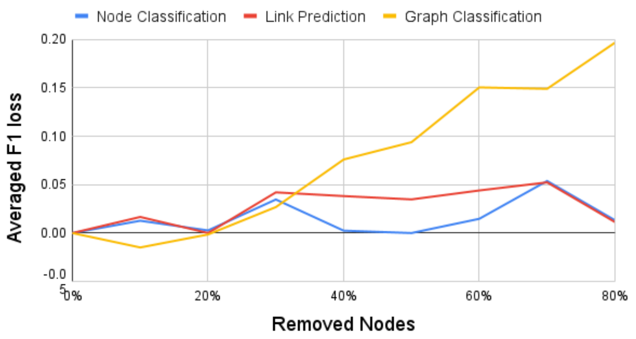

Figure 2: GCN micro F1 loss for a range of Node Removal strengths on various tasks. 25 trials per data point, seperated by 10% each.

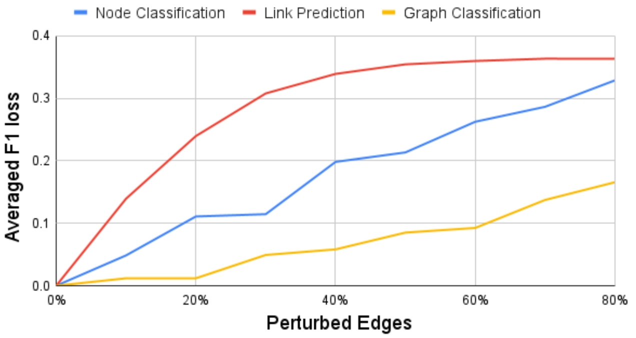

Figure 3: GCN micro F1 loss for a range of Edge Rewiring strengths on various tasks. 10 trials per data point, separated by 10% each.

The next experiment compares the above tasks on more perturbation methods, namely Node Removal and Edge Rewiring. Node Removal utilized the same method as Edge Removal but adapted to nodes instead. Edge Rewiring are explained further in Section 2. This experiment contained the same graphs, but a lowered experiment count of 10 for Edge Rewiring. This was due to the time required to produce perturbed graphs for Node Classification. The goal is to identify whether different perturbation methods produce different results.

In Figure 2 and Figure 3 the effect of Node Removal and Edge Rewiring can be seen on the same tasks. In Figure 2 Node Removal appears to have mostly the same results in terms of ranking, though the effect on Node Classification and Link Prediction is a lot less clear, despite having the same experiment count as Edge Removal. On the other hand, Figure 3 showcases a much clearer different outcome. Here, Node Classification and Link Prediction perform significantly worse than before, now performing worse than Graph Classification, which sees little change from Figure 2. This indicates that under different perturbations, such as the implemented Edge Rewiring, the stability ranking may shift. What is also noteworthy is that, as opposed to the usual linear nature of these results, Link Prediction's effect decelerates over time until around 36%. Taking into account its F score being calculated over 2 classes (link or no link), it may have already reached its worst-case prediction accuracy.

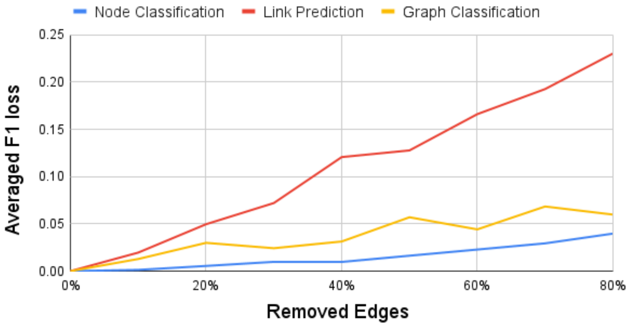

Figure 4: GAT Micro F1 loss for a range of Edge Removal strengths on various tasks. 25 trials per data point, seperated by 10% each.

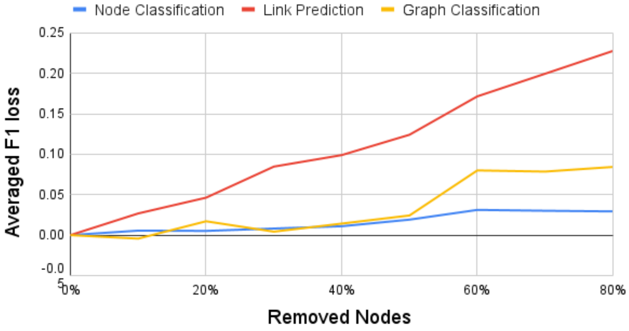

Figure 5: GAT Micro F1 loss for a range of Node Removal strengths on various tasks. 25 trials per data point, seperated by 10% each.

Next, the non-spectral GAT architecture was compared to the GCN. The experiments conducted here utilized a GAT with 3 layers, each with 64 hidden channels, and 5 attention heads. The utilized perturbation methods were kept the same, though the amount of experiments was lowered due to the GAT's processing time. The goal is to identify whether a non-spectral method has different effects on the various tasks.

In Figure 4 the effect of Edge Removal on the GAT architecture can be seen. These results do not align with its GCN counterpart in Figure 1. Graph Classification and Link Prediction have switched positions, placing Link Prediction as the most affected task by Edge Rewiring. However, Graph Classification does not tie Node Prediction, still performing worse than it. Comparing the mentioned graphs showcases that a change in architectures alone had a massive effect on the results.

In Figure 5 and Figure 6 the effect of Node Removal and Edge Rewiring can once again be seen, now on the GAT architecture. This time around, there seems to be no noticeable difference in ranking compared to before. In Figure 5 Link Prediction continues to be the worst-performing task, with Graph Classification only performing a bit worse than Node Classification towards larger perturbations. These results overall seem to line up heavily with Figure 4. Meanwhile, in Figure 6, roughly the same results can be seen as in GCN's counterpart in Figure 3, with Node Classification once again performing worse than Graph Classification. Link Prediction is affected just as heavily by the Edge Rewiring method across both architectures.

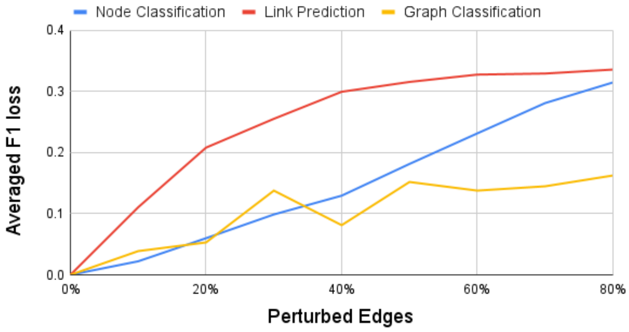

Figure 6: GAT Micro F1 loss for a range of Edge Rewiring strengths on various tasks. 10 trials per data point, seperated by 10% each.

5 Responsible Research

From a reproducibility standpoint, this research paper describes in detail what GNN structures and perturbation methods are utilized, and how their algorithms are laid out. The ways the different tasks are handled from an algorithm viewpoint are also described. Finally, the graphs on which they are executed, as well as where one can find them, are described. While it is believed anyone can recreate this work and test it for themselves, elements such as potential unseen bugs in written code may affect the attempt to reproduce results.

From an ethical standpoint, this research tackles vulnerabilities in GNNs. While this research could be utilized to improve GNN capabilities, it may also be utilized nefariously, resulting in attackers having a more effective job. This research also tackles Machine Learning, specifically in the field of Deep Learning, which falls under Artificial Intelligence. In recent years this is a field that people such as artists have been concerned about due to the ability of AI to create art and potentially take jobs [19]. While GNNs are not necessarily the specific field these people are concerned about, future developments in GNNs may result in more similar effects.

6 Discussion

The results summarized indicate the following:
- GCN Edge Removal and Node Removal: Here Graph Classification is the least stable, followed by Link Prediction and Node Classification being roughly tied.
- GAT Edge Removal and Node Removal: Here Link Prediction is the least stable, followed by Graph Classification and then Node Classification.
- Edge Rewiring: Here Link Prediction is the least stable, followed by Node Classification and then Graph Classification.

As can be seen by these results, the difference in task stability depends on both the chosen perturbation type and architecture. Despite this, the worst-performing architecture in all six experiments was either Link Prediction or Graph Classification, with a significant lead over the other tasks. In no experiment was Node Classification deemed to be the least stable task, always roughly matching a task while in last, or ending up second behind Link Prediction.

When focusing on Edge Removal and Node Removal, Link Prediction performing much better under the GCN framework could be linked to the GCN being a spatial architecture as opposed to the non-spatial GAT. This was further described in Section 2, where it was mentioned that spatial architectures take the global structure into account. The utilized implementation of Link Prediction made use of Graph Auto-Encoders to encode and decode the graph, which may find this characteristic of spatial architectures to be advantageous.

The heavy impact of Edge Rewiring falls in line with what is shown in [17]. The results are comparable to the effect on the IMDB-MULTI database for graph classification, which was the database that was impacted the least.

Most accuracy lines in the graphs were linear, with the exception of Link Prediction in the Edge Rewiring graphs. This could be explained by Link Prediction reaching its worst-case accuracy. Further perturbations can only affect the accuracy minimally from here.

No experiment is perfect, including this one, containing a few shortcomings in its process. First of all, this experiment contained fewer measurements due to limited processing power. This is most noticeable in Figure 2, where Node Classification and Link Prediction have many ups and downs, with some measurement points even hitting negative loss at times in both Node Removal graphs. As a result, these tasks could not be reliably compared in the mentioned figure. This need for more measurement points could also be due to the concept of Node Removal resulting in there being fewer test cases to work with for both Node Classification and Link Prediction.

This experiment also did not contain equal class counts for each task, resulting in bottom line accuracy being inconsistent for Node Classification, which contained 3 classes, compared to the other tasks, which contained 2 classes. As a result, its effects could have been higher than on the other tasks. This was attempted to be avoided as much as possible, however no group of databases could be found that had equal class counts. While this is unlikely to have had an effect in the long run, with Node Classification generally being the most stable, it should be noted in case of future research.

Finally, Link Prediction does not have a single agreed upon method. For this experiment, this task was handled by a Graph Auto-Encoder, but perhaps with a different method, there would be different results.

7 Conclusions and Future Work

At the start of this paper the following question was asked: How does the choice of task types, namely node classification, link prediction, and graph classification, impact the stability of a GNN in the face of perturbations? To answer this, the impact of three perturbation methods was assessed over a range of perturbation strengths on both the GCN and GAT architectures.

According to the results, there is no definitive ranking for the investigated tasks. The stability of task types depends on both the chosen architecture and the chosen perturbation method. Despite this, Link Prediction did perform the worst across all perturbation methods on the GAT framework and was the most unstable in 4/6 experiments. Graph Classification followed suit, being the most unstable in 2/6 experiments. Despite this, Node Classification was generally the most stable when compared to the other tasks, even if it was more unstable than Graph Classification on Edge Rewiring. All of this indicates that it may be more advantageous to utilize non-spatial architectures for Graph Classification and spatial architectures for Link Prediction. However, across the board, all need to be improved under Edge Rewiring.

From here a few more direct paths can be taken for further research. First of all, as described in Section 2 and reiterated in Section 6, Link Prediction was done utilizing a Graph Auto-Encoder implementation. As there are more methods, it could be interesting to compare different Link Prediction methods in terms of their stability. This could bring better results than described in this paper. It could also be interesting to explore architectures which are more stable against Edge Rewiring for every single task type, as the effect of this method is most noticeable across the board. However, due to the large impact of this perturbation method, it may be more advantageous to split this up into separate research papers per task type.

References

[1] A. A. Awan, “A Comprehensive Introduction to Graph Neural Networks (GNNs),” 7 2022.

[2] M. Labonne, “Graph Convolutional Networks: Introduction to GNNs — Towards Data Science,” 8 2023.

[3] Y. Wang, Z. Li, and A. Barati Farimani, *Graph Neural Networks for Molecules*, p. 21–66. Springer International Publishing, 2023.

[4] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, “Graph neural networks: A review of methods and applications,” 2021.

[5] D. Grattarola, “A practical introduction to GNNs - Part 2,” 3 2021.

[6] O. Hussein, “Graph Neural Networks Series — Part 4 —The GNNs, Message Passing Over-smoothing,” 5 2023.

[7] A. Daigavane, B. Ravindran, and G. Aggarwal, “Understanding convolutions on graphs,” *Distill (San Francisco, Calif.)*, vol. 6, 8 2021.

[8] H. Kenlay, D. Thano, and X. Dong, “On the stability of graph convolutional neural networks under edge rewiring,” in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, June 2021.

[9] J. Ma, S. Ding, and Q. Mei, “Towards more practical adversarial attacks on graph neural networks,” 2021.

[10] F. Gama, J. Bruna, and A. Ribeiro, “Stability properties of graph neural networks,” *IEEE Transactions on Signal Processing*, vol. 68, p. 5680–5695, 2020.

[11] X. Liu, Y. Zhang, M. Wu, M. Yan, K. He, W. Yan, S. Pan, X. Ye, and D. Fan, “Revisiting edge perturbation for graph neural network in graph data augmentation and attack,” *ArXiv*, vol. abs/2403.07943, 2024.

[12] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2017.

[13] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” 2018.

[14] PyG Team, “pool.global_mean_pool,” 2021.

[15] T. N. Kipf and M. Welling, “Variational graph auto-encoders,” 2016.

[16] T. Zhou, “Discriminating abilities of threshold-free evaluation metrics in link prediction,” *Physica A: Statistical Mechanics and its Applications*, vol. 615, p. 128529, 2023.

[17] Y. Ma, S. Wang, T. Derr, L. Wu, and J. Tang, “Graph adversarial attack via rewiring,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD ’21, (New York, NY, USA), p. 1161–1169, Association for Computing Machinery, 2021.

[18] R. Kundu, “F1 Score in Machine Learning: Intro amp; Calculation,” 4 2024.

[19] V. Thorpe, “‘ChatGPT said I did not exist’: how artists and writers are fighting back against AI,” 3 2023.