Project 2: Does Graph Contrastive Learning improve robustness of GNNs?

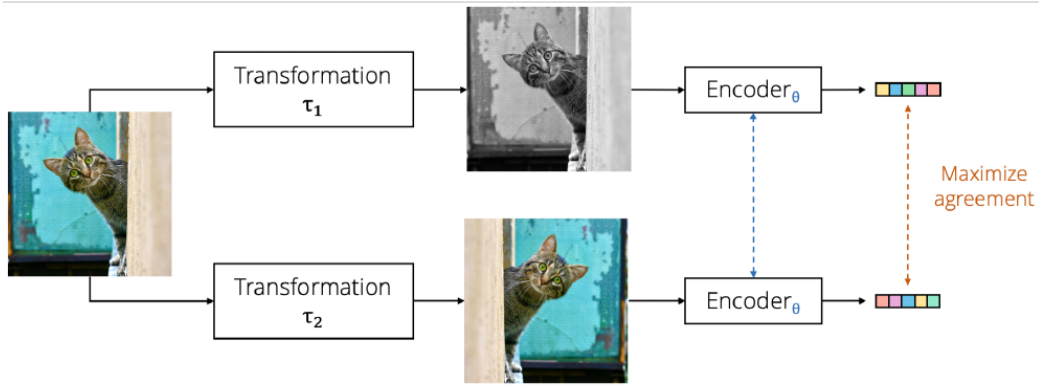
Background

Since the beginning on “neural network’s era” the question about reliability and and robustness of the models was discussed [1, 2] and studied quiet a lot. There exist a lot of examples of neural networks being fooled by injective small and unnoticeble perturbations into data. Especially for images it can be easily seen, as on the example below from [1]: all the instances in the right column are predicted to be an ostrich.

There were suggested different methods to improve the robustness of the models by, for example, adding perturbed instances from the initial data into training dataset or adjusting objective function of the model, so that it optimizes worst-case loss [8].

Semi-supervised learning tasks with partially labeled graph structured data have seen significant improvement with the use of Graph neural networks (GNNs). However, the majority of the research has been centered around (semi-) supervised learning, which leads to several limitations such as dependence on labels, inadequate generalization, and low robustness. Labeling graph data for training is a challenging task, and the presence of inaccurate labels can lead to the training process generating erroneous GNN models for node classification.

Consequently **self-supervised learning**, which was originally developed as a response to the problem of under-labelling, has been found to enable networks to learn more solid representations and thus improve their robustness. In particular, **contrastive learning** has led to a growing amount of work focusing on the use of self-supervised learning to improve network robustness.



Contrastive learning leverages the instance-level identity, thus eliminating the extra labour of designing the pretext task. In contrast learning, in order to further make the original data as different as possible from the augmented data, some works have introduced adversarial attacks into the design of the augmentation approach and have successfully achieved the same robustness as supervised adversarial learning on image classification.

Graph Neural Networks have shown astonishing performance in the last couple of years and proved to be a powerful tool, that can be applied in various fields of science and technology [2]. However, robustness of GNNs is very concerning aspect due to specificity of the data itself. The key feature of graph data is that instances (nodes) are not independent as it is for images, texts etc, but interconnected and influence each other. This creates the following opposing situation: on one hand, the prediction is based not only on the features of the node itself, but also on the features of other nodes, that might make the prediction more robust; on the other hand, this fact allows “propagation” of the error from one node to others in a cascading manner[2]. Besides, the difficulty in defining the notion of similarity between two graphs also introduces distinction between the concept of robustness in graphs and images.

Motivation

Self-supervised learning for graph neural networks (GNNs) is a hot topic, as it enables learning from large amounts of unlabeled data. However, the robustness of these models has not been studied extensively, which presents an opportunity for useful research. Furthermore, although we have talked about the great success self-supervised learning achieves in computer vision field, unlike image processing, the recognition of the semantics of a graph is a difficult task, and designing augmentation methods for graphs in graph contrast learning is also an ongoing research topic.

Currently, in GCL, it is common to augment graphs by adding and removing edge or perturbating node attributes. Recently, a few new GCL models with novelly designed augmentation approaches are brought forward. One wonders then: can the current graph contrast learning also enhance the robustness of graph neural networks in the same way? Therefore, we need to attack the robustness of the currently proposed graph contrast learning algorithms and test them.

Workflow

We evaluated several GCL models in the evasion scenario for both graph classification and node classification task. In each specific case, we divided our workflow into three parts.

Training phase

In this part, we trained GCL models in a self-supervised learning way, i.e., we use every sample in the dataset without labels. No split here. We tried to keep the same hyperparameters with those stated in the original papers, or used their default settings in the published codes. We all used Adam as the optimizer for training the GCL models as encoders.

We also trained their backbone network in a traditional supervised learning way, i.e. the control group. For fair comparison, we kept the hyperparameters exactly the same as their counterparts, as well as the optimizer.

To be specific, we first applied corruption functions/augmentations on clean samples and obtained the perturbed views. Then, we fed both the clean samples and perturbed views into a GNN and obtained vectors as global embeddings for graph classification or patch embeddings, i.e. node embeddings, for node classification. Next, we defined a loss function and calculated the similarity/difference between embeddings. Finally, we optimized the GNN to maximize the agreement of embeddings between clean samples and its perturbed view or minimize the agreement of embeddings between clean samples and its corrupted version.

Evaluation phase

After we obtained an encoder, we froze the encoder. Then, we split dataset and obtain embeddings by passing clean data through the frozen model. Then, we trained a single layer linear classifier as the appending classifier, i. e. linear regression, on the embeddings of the training set. Notice that we did not train an appending classifier for control group, as they were trained in an end-to-end supervised learning way. Next, we tested the trained classifier with the test set and obtained the clean accuracy.

Attack phase

In this phase, we froze both the encoder and classifier. Next, we generated adversarial samples using different attack methods based on the frozen model and calculated the accuracy for adversarial samples. Notice that for graph classification, we only generated adversarial samples for inaccurately classified clean samples.

One thing to mention is that we trained several classifiers upon the same frozen encoder and averaged the accuracies as the final one. According to the original authors, we might choose to train only one LR classifier but repeat from the begining, i.e., training the encoder, for multiple runs to calculate the average accuracy.

Models and datasets

We used different models and datasets for graph and node classification.

	Graph classification	**Node classification**
GCL models	InfoGraph Graph Contrastive learning with Augmentations (GCL) Adversarial GCL (AD-GCL)	Deep Graph InfoMax (DGI) GCL GCL with Adaptive Augmentations (GCA)
Attack models	Random attack Projected Gradient Descent (PGD)	Random attack Projected Randomized Block Coordinate Descent (PRBCD) Greedy Randomized Block Coordinate Descent (GRBCD)
Datasets	PROTEINS, NCI1, DD (from TUDataset)	Cora, CiteSeer, PubMed (from CitationFull)

Contrastive learning models

InfoGraph[16]

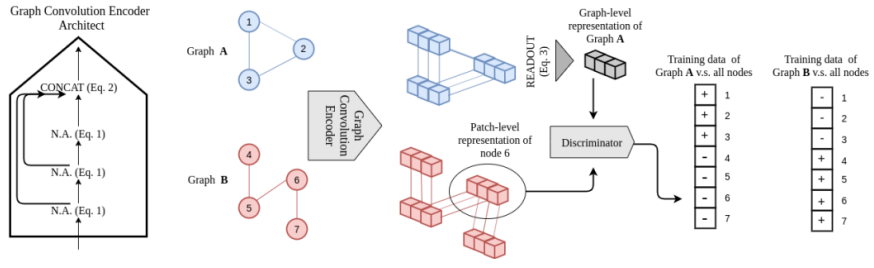


Figure 1: Illustration of InfoGraph. N.A. denotes neighborhood aggregation. An input graph is encoded into a feature map by graph convolutions and jumping concatenation. The discriminator takes a (global representation, patch representation) pair as input and decides whether they are from the same graph. InfoGraph uses a batch-wise fashion to generate all possible positive and negative samples. For example, consider the toy example with 2 input graphs in the batch and 7 nodes (or patch representations) in total. For the global representation of the blue graph, there will be 7 input pairs to the discriminator and same for the red graph. Thus, the discriminator will take 14 (global representation, patch representation) pairs as input in this case.

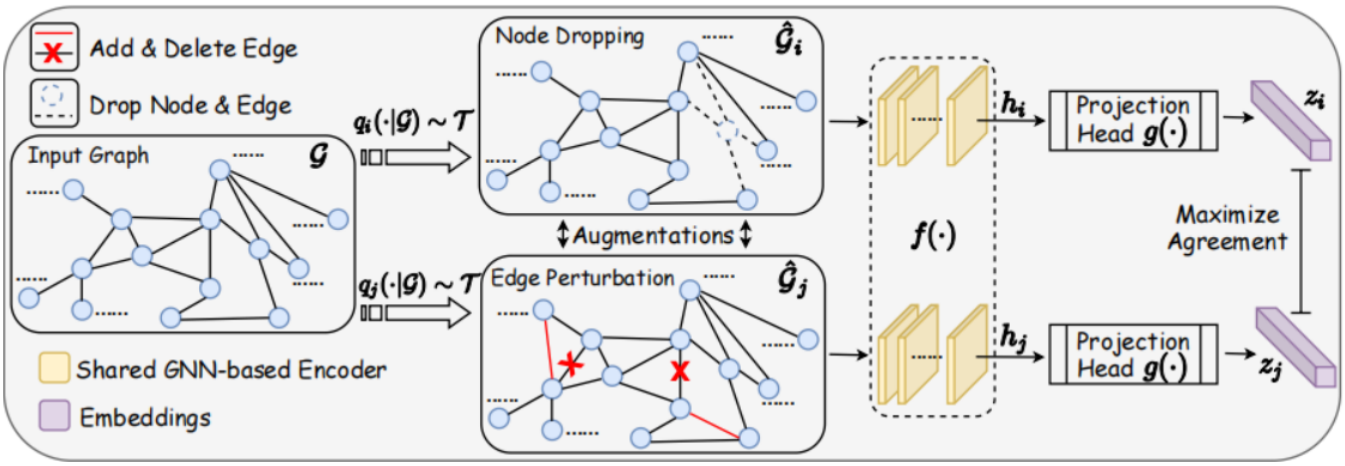
InfoGraph specializes in graph neural networks (GNNs) due to their versatile nature as embedding architectures. GNNs excel at producing node representations by iteratively aggregating information from neighboring nodes within a graph. These aggregated representations, known as patch representations, are acquired by merging the features of nearby nodes. We seek to obtain graph representations by maximizing the mutual information between graph-level and patch-level representations. By doing so, the graph representations can learn to encode aspects of the data that are shared across all substructures. InfoGraph aims to acquire graph representations by maximizing the mutual information between graph-level and patch-level representations. This approach enables the graph representations to capture shared characteristics present in all substructures of the data.

It is important to note that InfoGraph bears similarities to Deep Graph Infomax (DGI) for unsupervised node embeddings. However, there are notable design differences due to our specific problem focus. Firstly, in DGI, random sampling is utilized to obtain negative samples, as their primary objective is learning node embeddings within a graph. Contrastive methods, on the other hand, require a large number of negative samples to be competitive. Hence, our approach relies on batch-wise generation of negative samples, which is crucial for learning graph embeddings from numerous graph instances.

Secondly, the choice of graph convolution encoders is also critical. InfoGraph employs Graph Isomorphism Network (GIN), whereas DGI uses Graph Convolutional Network). GIN offers a more suitable inductive bias for graph-level applications. Careful consideration of graph neural network designs ensures that graph representations possess discriminative qualities when compared to other graph instances.

Graph Contrastive learning with Augmentations (GCL)[14]

Overall, we train the encoder by comparing each graph and its augmented variant with maximum agreement.



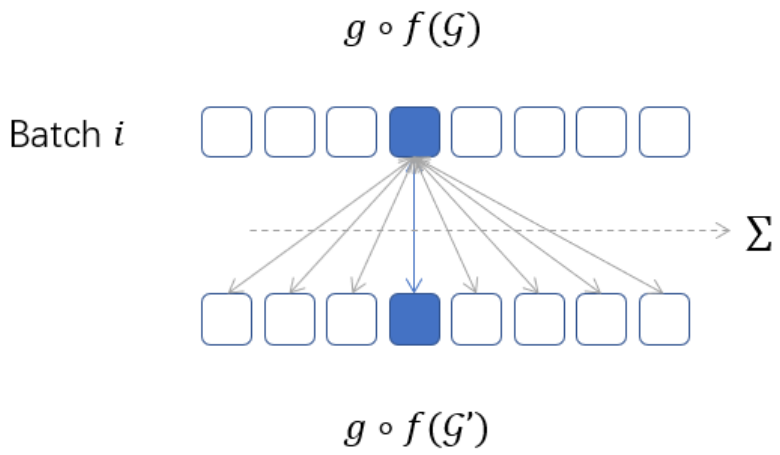
In details, only one augmentation are applied to each graph. Augmented graphs are compared again its original ones. According to different datasets, the augmentation will be chosen from a combination with equal probability. The combinations are shown as follows:

Dataset	Augmentations
DD	drop_nodes (with ratio 0.2) or subgraph (with size_ratio 0.2)
NCI1	drop_nodes (with ratio 0.2) or subgraph (with size_ratio 0.2)
PROTEINS	drop_nodes (with ratio 0.2) or subgraph (with size_ratio 0.2)
REDDIT-BINARY	drop_nodes (with ratio 0.2), edge_perturbation (with ratio 0.2 and only dropping) or subgraph (with size_ratio 0.2)
REDDIT-5K	drop_nodes (with ratio 0.2), edge_perturbation (with ratio 0.2 and only dropping) or subgraph (with size_ratio 0.2)

By using graph convolutional networks, we obtain global representations for each graph. After projection, we use InfoNCE to calculate the loss. The formula of InfoNCE is as follows:

$$\mathcal{L}_{\mathrm{N}} = -\underset{X}{\mathbb{E}}\left[\log \frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)}\right]$$

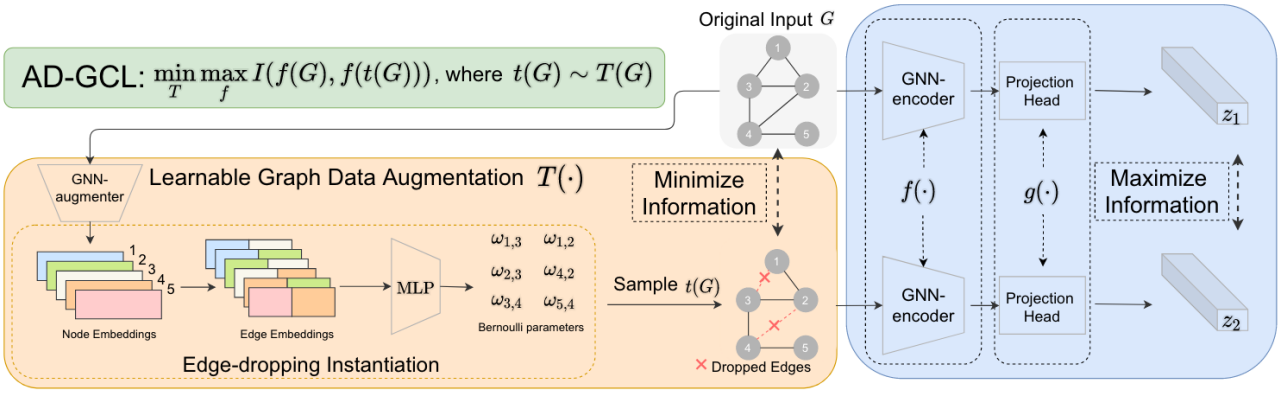
Basically, we calculate cosine similarity between one batch and its augmented variant. Then we sum up the similarity and the loss will be low if similarity of the positive pair take the majority of the sum similarity.



For evaluation, we obtain the embeddings by passing all original graphs through the encoder. Then we train a single layer linear regression classifier with the embeddings and labels and test the accuracy of the classifier as our evaluation of the "quality of the embeddings". We made 5 experiments and took the average as the final result. The result will be shown below.

A obvious drawback of this model is the design of the augmentations is totally empirical. Besides, we observed that the result of single experiments could be unstable.

Adversarial GCL (*AD-GCL*)[5]



The *key feature of ADGCL*, that distinguishes it from other graph contrastive learning methods, is that augmenters is being trained along with the encoder. This brings us to the setting of min-max problem, where, on one hand, encoder tries to maximise the mutual information (agreement) between embeddings of original and perturbed graphs and augmenter, on the other hand, tries to minimise the same value:

$$\min_{T \in \mathcal{T}} \max_{f} I(f(G); f(t(G))), \quad \text{where } G \sim \mathbb{P}_{\mathcal{G}}, t(G) \sim T(G)$$

The min-max principle in AD-GCL aims to train the encoder such that even with a very aggressive GDA (i.e., where t(G) is very different from G), the mutual information between the perturbed graph and the original graph can be maximized. AD-GCL augmenter views original graph as an anchor and tries to push the perturbed instance as far from the anchor as it's possible. Mutual information, that cannot be calculated explicitly, is estimated by InfoNCE loss, which is known to be a lower bound of the mutual information and is frequently used for contrastive learning.

Technically, different types of augmentations (node dropping, edge dropping and adding, feature masking etc) could be defined for the model, but the authors of the method use only edge dropping, claiming, that this approach keeps the maximal information and connection for downstream tasks and work the best on the used datasets (molecules and social networks). However, they notice, that in another applications another augmentation techniques can be suitable and show better performance.

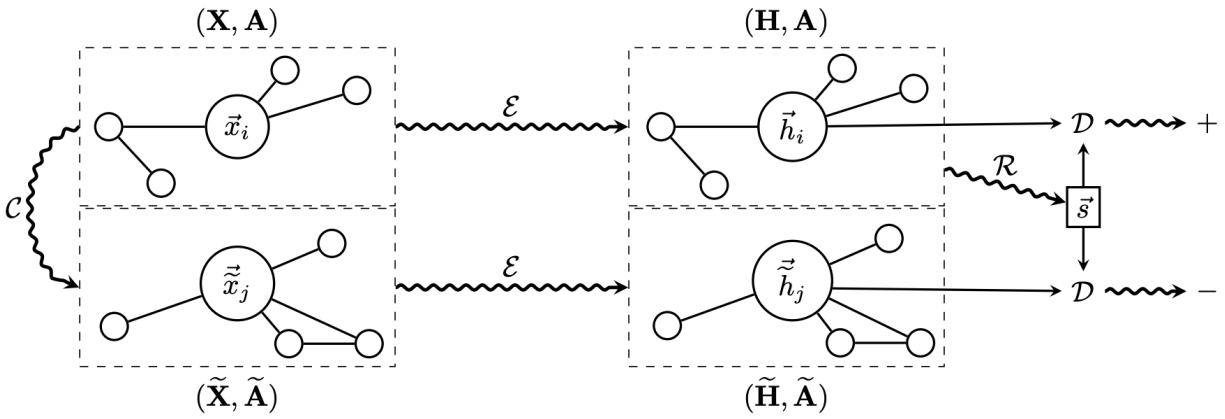
When defining augmenter it is especially important to introduce some constraints (regularization), that will ensure, that augmented instances are not too far from the anchor. They suggest the following regularization term, where ω_e is the probability of dropping edge e:

$$\min_{\Phi} \max_{\Theta} I(f_\Theta(G); f_\Theta(t(G))) + \lambda_{\text{reg}} \mathbb{E}_G \big[\sum_{e \in E} \omega_e / |E| \big], \text{ where } G \sim \mathbb{P}_{\mathcal{G}}, t(G) \sim T_\Phi(G)$$

Two types of AD-GCL are considered: AD-GCL-FIX, where regularization constant is fixed and set to 5, and AD-GCL-OPT, where this constant is treated as a hyperparameter and tuned. It is interesting to notice, that in case of unsupervised learning, that is the main topic of our project, both sub methods show pretty similar results and, therefore, in our analysis we used AD-GCL-FIX.

Deep Graph Infomax (DGI) [11]

DGI is an unsupervised approach to learning node representations in graph-structured data by maximizing mutual information between patch representations and high-level summaries of graphs, using established graph convolutional network architectures. The learned patch representations capture subgraphs around nodes of interest and can be reused for downstream node-wise learning tasks. Unlike other methods, DGI does not rely on random walk objectives and can be applied to both transductive and inductive learning setups.



Assuming the single-graph setup (i.e., $(\mathbf{X}, \mathbf{A})$ provided as input), we will now summarize the steps of the Deep Graph Infomax procedure:

1. Sample a negative example by using the corruption function: $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}}) \sim \mathcal{C}(\mathbf{X}, \mathbf{A})$.
2. Obtain patch representations, $\vec{h}_i$ for the input graph by passing it through the encoder: $\mathbf{H} = \mathcal{E}(\mathbf{X}, \mathbf{A}) = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}$.
3. Obtain patch representations, $\vec{\widetilde{h}}_j$ for the negative example by passing it through the encoder: $\widetilde{\mathbf{H}} = \mathcal{E}(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}}) = \{\vec{\widetilde{h}}_1, \vec{\widetilde{h}}_2, \ldots, \vec{\widetilde{h}}_M\}$.
4. Summarize the input graph by passing its patch representations through the readout function: $\vec{s} = \mathcal{R}(\mathbf{H})$.
5. Update parameters of $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{D}$ by applying gradient descent to maximize Equation 1.

To some extent, the corruption function *C* here works like a negative augmentation. The basic idea is that the global representation extracted from the original graph should achieve high mutual information with its own patch representations while achieving low mutual information with patch representation from its corrupted variant.

One key advantage of DGI is that it does not rely on random walk objectives, which are commonly used in previous unsupervised learning approaches with graph convolutional networks (GCNs). This makes DGI applicable to both transductive learning setups, where all nodes are observed during training, and inductive learning setups, where new nodes can be added at inference time.

The abstract highlights that DGI achieves competitive performance on various node classification benchmarks, and in some cases, even outperforms supervised learning methods. This suggests that DGI is effective in learning useful representations of nodes in graph-structured data without the need for labeled examples.

Adversarial attack models

Random attack

Here, we just randomly flip edges in a graph based on some given ratio.

Projected Gradient Descent

The goal of the attacker is to maximise the loss as opposed to the training process, where loss is being minimized:

$min_{s \in \{0,1\}^{N^2}} I(G, G'(s))$

The main difference from the AD-GCL training process is that the model's parameters are not being updated here.

The problem is solved by applying classical gradient descent method, but gradients are taken **with respect to data, not parameters**. Also we can notice, that this optimization problem is discrete, so to solve it convex relaxation is applied and then the perturbed graph can be obtained by sampling for each edge if it should be perturbed or not with probability s_i:

$min_{s \in [0,1]^{N^2}} I(G, G'(s))$

Every time we firstly make a gradient step and get a new value for s and then project it back onto the valid space ($s \in [0,1]^{N^2}$ in our case).

It is also important to notice, that the idea of the attack is to be "unnoticeable", which in case of graphs translates into the constraint onto the amount of perturbed edges (the constraint is applied during the sampling procedure, after PGD itself).

Projected Randomized Block Coordinate Descent (PRBCD)[13]

Projected Gradient Descent can be computationally demanding if the graph is gaint with many nodes. To make the calculation possible, **this method only optimizes a subset of all variables at a time**. The algorithm is as follows:

Algorithm 1 Projected Randomized Block Coordinate Descent (PR-BCD)

1: **Input:** Gr. $(\boldsymbol{A}, \boldsymbol{X})$, lab. $\boldsymbol{y}$, GNN $f_\theta(\cdot)$, loss $\mathcal{L}$
2: **Parameter:** budget Δ, block size b, epochs E & $E_{\text{res.}}$, heuristic $h(\dots)$, learning rate α_t
3: Draw w/o replacement $\boldsymbol{i}_0 \in \{0, 1, \dots, n^2 - 1\}^b$
4: Initialize zeros for $\boldsymbol{p}_0 \in \mathbb{R}^b$
5: **for** $t \in \{1, 2, \dots, E\}$ **do**
6: $\quad \hat{\boldsymbol{y}} \leftarrow f_\theta(\boldsymbol{A} \oplus \boldsymbol{p}_{t-1}, \boldsymbol{X})$
7: $\quad \boldsymbol{p}_t \leftarrow \boldsymbol{p}_{t-1} + \alpha_t \nabla_{\boldsymbol{p}_{t-1}[\boldsymbol{i}_{t-1}]} \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$
8: $\quad$ Projection $\boldsymbol{p}_t \leftarrow \Pi_{\mathbb{E}[\text{Bernoulli}(\boldsymbol{p}_t)] \leq \Delta}(\boldsymbol{p}_t)$
9: $\quad \boldsymbol{i}_t \leftarrow \boldsymbol{i}_{t-1}$
10: $\quad$ **if** $t \leq E_{\text{res.}}$ **then**
11: $\quad\quad \text{mask}_{\text{res.}} \leftarrow h(\boldsymbol{p}_t)$
12: $\quad\quad \boldsymbol{p}_t[\text{mask}_{\text{res.}}] \leftarrow \boldsymbol{0}$
13: $\quad\quad$ Resample $\boldsymbol{i}_t[\text{mask}_{\text{res.}}]$
14: $\boldsymbol{P} \sim \text{Bernoulli}(\boldsymbol{p}_E)$ s.t. $\sum \boldsymbol{P} \leq \Delta$
15: Return $\boldsymbol{A} \oplus \boldsymbol{P}$

After initialization, the algorithm can be divided into two parts:

- (line 6-8) Optimize the loss over a subset of all variables, which is indicated by i_t. Do projection to meet the budget.
- (line 11-13) Keep those entries with high probability of being perturbed (large elements in P_t). Resample in the left entries to fill the vacancy.

The second part only apply when t is smaller than some threshold in the early stage of the algorithm. After the threshold E_{res} only fine-tuning applies.

The final sample will be generated based on the probability p_E.

Greedy Randomized Block Coordinate Descent (GRBCD)[13]

GRBCD shares most of the properties and requirements with PRBCD. It also uses an efficient gradient based approach. However, it greedily flips edges based on the gradient towards the adjacency matrix.

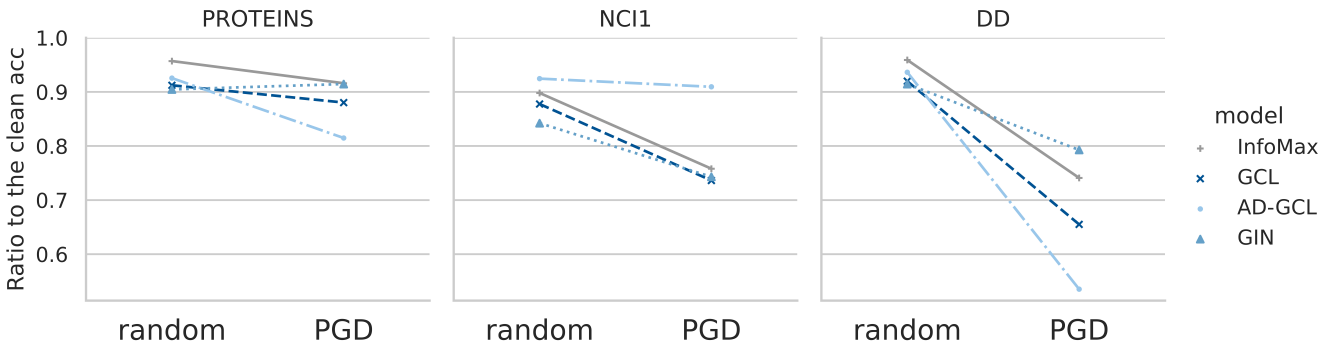
Results and observations

We have done series of experiments and summarized some interesting observations. We described the robustness of models by drop ratio - the more the accuracy drops under an attack, the less robust the model is against that attack.

$$R_{drop} = \frac{acc_{clean} - acc_{adv}}{acc_{clean}}$$

Graph classification

Contrastive learning boosts robustness against random attack but fails against more sophisticated attacks.

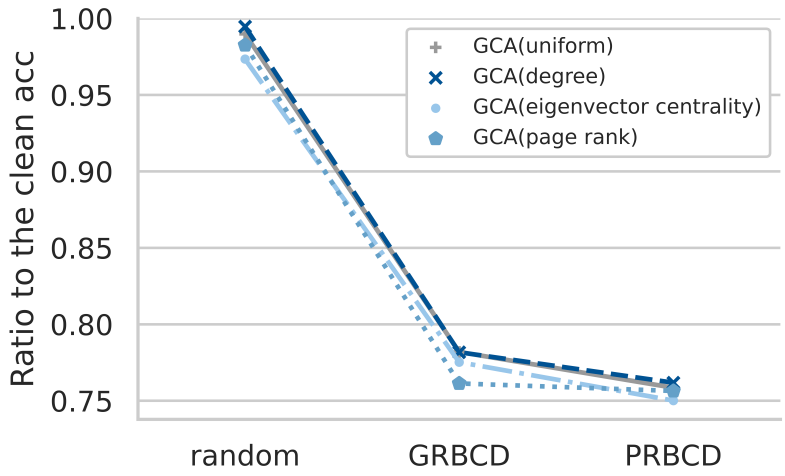


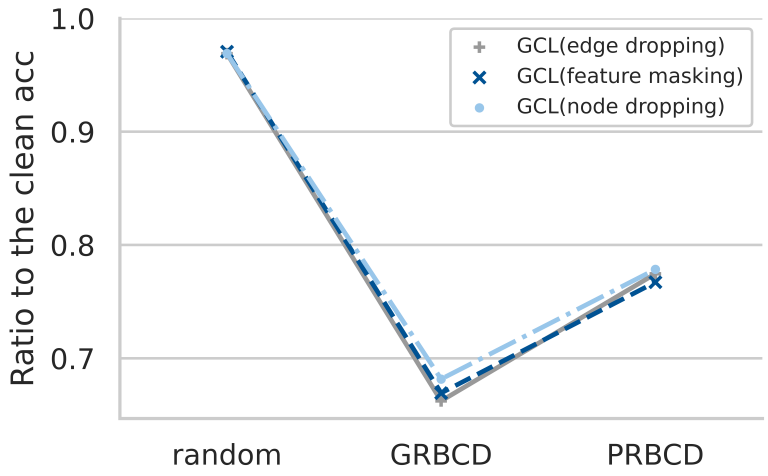
GIN was trained in a supervised learning way as the control group. InfoMax, GCL, and AD-GCL are different contrastive learning models with GIN as the backbone.

We can see that on all three dataset, GIN's performance dropped at most under the random attack. This implies that contrastive learning methods boost robustness against the random attack. However, this did not hold anymore under the PGD attack and GIN even achieved minimun drop ratio on the DD dataset.

Node classification

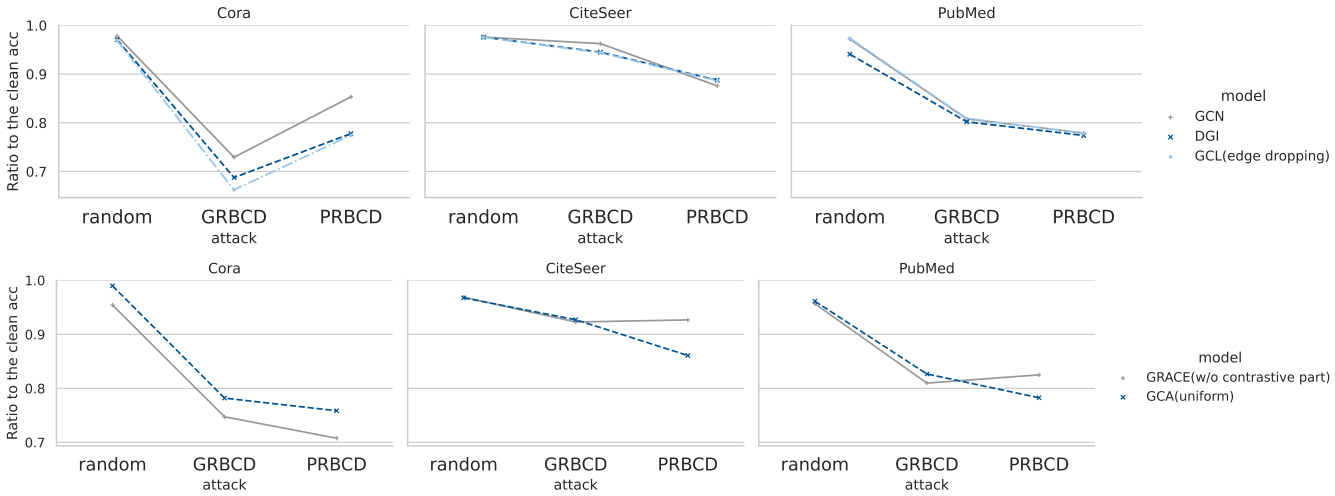
Type of augmentation does not differ in boosting robustness





We firstly hypothesised that models trained with a certain type of augmentation are more robust under similar attacks. However, as we can see, types of augmentation do not make an noticeable difference.

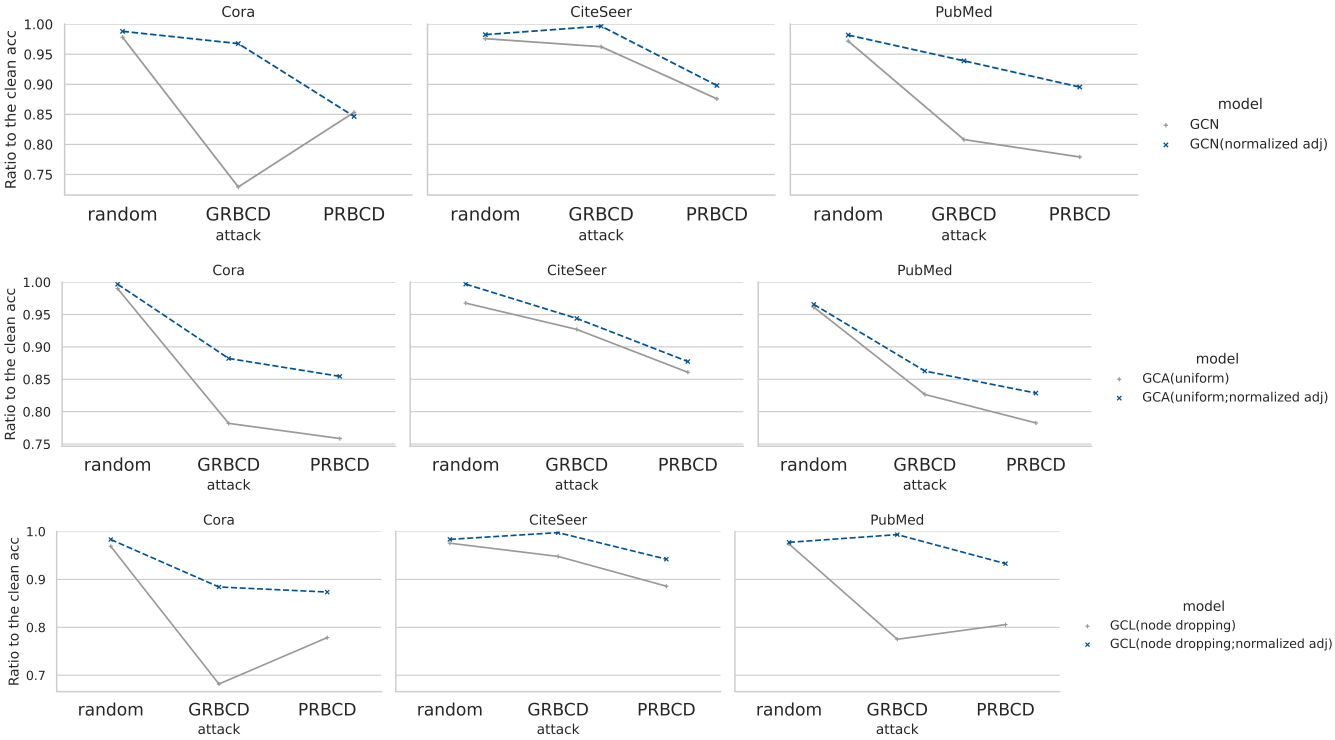
Contrastive learning does not necessarily boost robustness



In the first plot, GCN was the plain backbone use by DGI and GCL. DGI is itself already a contrastive learning method and GCL employs further augmentations based on DGI.

As we can see, contrastive learning methods did not consistently boost robustness.

Employing the normalized adjacency boosts robustness



Here, we can see that models using normalized adjacency all achieved better performance under all kinds of attack on all dataset (except for GCN under PRBCD on Cora dataset, where GCN with normalized adjacency still achieved competitive performance with its counterpart).

Further questions

Based on the observations we got, we think the following two questions would be the potential future work:

- Why does contrastive learning not consistently work against attacks?
- How to exploit normalized adjacency and design more robust models/learning methods?

References

[1] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing Properties of Neural Networks. In *International Conference on Learning Representations*, 2014.

[2] Daniel Z¨ugner, Amir Akbarnejad, and Stephan G¨unnemann. Adversarial Attacks on Neural Net works for Graph Data. In *In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2847–2856, 2018.

[3] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using Self-Supervised Learn ing Can Improve Model Robustness and Uncertainty. In *Advances in Neural Information Processing Systems*, volume 32, page 15637–15648, 2019.

[4] Minseon Kim, Jihoon Tack, and Sung Ju Hwang. Adversarial Self-Supervised Contrastive Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 2983–2994, 2020.

[5] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. Adversarial Graph Augmentation to Improve Graph Contrastive Learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 15920–15933, 2021.

[6] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self Supervised Learning: Generative or Contrastive. *IEEE Transactions on Knowledge and Data En gineering*, 35(1):857–876, 2021.

[7] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip Yu. Graph Self Supervised Learning: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[8] Stephan G¨unnemann. Graph Neural Networks: Adversarial Robustness. *Graph Neural Networks: Foundations, Frontiers, and Applications*, pages 149–176, 2022.

[9] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 2008.

[10] Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven Active Surveying for Collective Classification. In *Workshop on Mining and Learning with Graphs*, 2012.

[11] Petar Velickovi´c, William Fedus, William L. Hamilton, Pietro Pietro Li`o, Yoshua Bengio, and R. Devon Hjelm. Deep Graph Infomax. In *International Conference on Learning Representations*, 2019.

[12] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. Topol ogy Attack and Defense for Graph Neural Networks: An Optimization Perspective. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 3961–3967. International Joint Conferences on Artificial Intelligence Organization, 2019.

[13] Simon Geisler, Tobias Schmidt, Hakan S¸irin, Daniel Z¨ugner, Aleksandar Bojchevski, and Stephan G¨unnemann. Robustness of Graph Neural Networks at Scale. In *Advances in Neural Information Processing Systems*, volume 34, pages 7637–7649. Curran Associates, Inc., 2021.

[14] You, Y., Chen, T., Sui, Y., Chen, T., Wang, Z., & Shen, Y. (2020). Graph contrastive learning with augmentations. *Advances in neural information processing systems*, *33*, 5812-5823.

[15] Kolesnikov, Alexander, et al. 'Revisiting Self-Supervised Visual Representation Learning'. *ArXiv [Cs.CV]*, 2019, http://arxiv.org/abs/1901.09005. arXiv.

[16] Sun, F. Y., Hoffmann, J., Verma, V., & Tang, J. (2019). Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*.