


Article

Investigating Transfer Learning in Graph Neural Networks

Nishai Kooverjee ^{1*} , Steven James ¹  and Terence van Zyl ² ¹ School of Computer Science and Applied Mathematics, University of the Witwatersrand;
nishai.kooverjee@gmail.com; steven.james@wits.ac.za² Institute for Intelligent Systems, University of Johannesburg; tvanzyl@uj.ac.za

* Correspondence: nishai.kooverjee@gmail.com

Abstract: Graph neural networks (GNNs) build on the success of deep learning models by extending them for use in graph spaces. Transfer learning has proven extremely successful for traditional deep learning problems: resulting in faster training and improved performance. Despite the increasing interest in GNNs and their use cases, there is little research on their transferability. This research demonstrates that transfer learning is effective with GNNs, and describes how source tasks and the choice of GNN impact the ability to learn generalisable knowledge. We perform experiments using real-world and synthetic data within the contexts of node classification and graph classification. To this end, we also provide a general methodology for transfer learning experimentation and present a novel algorithm for generating synthetic graph classification tasks. We compare the performance of GCN, GraphSAGE and GIN across both the synthetic and real-world datasets. Our results demonstrate empirically that GNNs with inductive operations yield statistically significantly improved transfer. Further we show that similarity in community structure between source and target tasks support statistically significant improvements in transfer over and above the use of only the node attributes.

Keywords: graph neural networks; machine learning; transfer learning; multi-task learning

1. Introduction

Deep learning has achieved success in a wide variety of problems, ranging from time-series data to images and video [1]. Data from these tasks are referred to as *Euclidean* [2] and specialised models such as recurrent and convolutional neural networks [3,4] have been designed to leverage the properties of such data.

Despite these successes, not all problems are Euclidean. One particular class of such problems involve *graphs*, which naturally model complex real-world settings involving objects and their relationships. Recently, deep learning approaches have been extended to graph-based domains using graph neural networks (GNNs) [5], which leverage certain topological structures and properties specific to graphs [2]. Since graphs comprise entities and the relationships between them, GNNs are said to learn relational information and may have the ability for relational reasoning [6].

One reason for the success of deep learning models is their ability to transfer to new tasks. In image classification, this transfer leads to more robust models and faster training [7]. Despite the importance of transfer in deep learning, there has been little insight into the nature of transferring *relational knowledge* — that is, the representations learnt by graph neural networks. There is also no comparison of the generalisability of different GNNs when evaluated on downstream task performance. This lack of insight is in part due to the lack of a model-agnostic and task-agnostic framework and standard benchmark datasets and tasks for carrying out transfer learning experiments with GNNs.

We conduct an empirical study within the contexts of *node classification* and *graph classification* to determine whether transfer in GNNs occur and, if so, what factors influence success. In particular, we make the following contributions: First, we provide a methodology and additional metrics for evaluating GNN transfer learning empirically. Second, we provide a novel method for creating synthetic graph classification tasks with community structure. Finally, we evaluate the transferability of several popular GNNs on both real



Citation: Kooverjee, N.; James, S.; van Zyl, T. Investigating Transfer Learning in Graph Neural Networks. *Preprints* 2021, 1, 0. <https://doi.org/>

Received:

Accepted:

Published:

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

and synthetic datasets. Our results demonstrate that we can achieve positive transfer using graph neural networks; and that certain models exploit strong community structure properties present in the source task to yield effective transfer.

1.1. Background

In this work, we consider problems for which the data can be modelled as a graph. A graph $G = \{V, E\}$ consists of a set of N vertices, V , and a set of edges E . Let $v_i \in V$ denote a vertex, and $e_{ij} = (v_i, v_j) \in E$ denote a directed edge from v_i to v_j . The *adjacency matrix*, $A \in \mathbb{R}^{N \times N}$, has $a_{ij} = 1$ where $e_{ij} \in E$, and 0 otherwise. A graph may have *node attributes*, where $X \in \mathbb{R}^{N \times C}$ is the node feature matrix, and $\mathbf{x}_i \in \mathbb{R}^C$ is the attribute vector for node v_i . Similarly, a graph may have an edge attribute matrix X^e [5]. An important measure of a node's connectivity is its *degree*, which is the number of edges the node is connected to [8]. We denote the degree of the i^{th} node as d_i . The degree matrix $D = \text{diag}(d_1, \dots, d_N)$ is the diagonal matrix containing the degrees of all vertices.

1.1.1. Graph Neural Networks

The success of CNNs for computer vision problems motivate the need to formulate a counterpart to the *convolution operator* for graphs [2]. Similarly to that of Euclidean signals, the problem of convolution for graphs can be tackled from either the *spatial domain* or *spectral domain*. One popular model is Graph Convolution Networks (GCNs) [9], which make use of a *spectral* graph convolution operation stacked layer-wise. We also consider two other models, GraphSAGE [10] and Graph Isomorphism Networks (GINs) [11], which utilise *spatial* graph convolutions to aggregate information from a node's neighbourhood. GINs have previously outperformed both GCN and GraphSAGE in experimental performance on social media and biological datasets [11].

These three considered GNNs fall under the category of message-passing neural networks (MPNNs) [12]. To provide comparison of the three GNNs' operations, we rewrite them as MPNN updates:

Model Update Rule

$$\begin{aligned} \text{GCN} \quad h_v^{(k)} &\leftarrow \sigma \left[W \cdot \left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\tilde{d}_u \tilde{d}_v}} h_u^{(k-1)} \right) \right] \\ \text{GraphSAGE} \quad h_v^{(k)} &\leftarrow \sigma \left[W \cdot \left(\frac{1}{\tilde{d}_v^{\text{in}}} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(k-1)} \right) \right] \\ \text{GIN} \quad h_v^{(k)} &\leftarrow \sigma \left[W \cdot \left((1 + \epsilon^{(k)}) \cdot h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)} \right) \right] \end{aligned}$$

where $h_v^{(k)}$ and $W^{(k)}$ are the GNN embedding for node v and the weight matrix at the layer k respectively, $\sigma(\cdot)$ is a non-linear activation function, $\mathcal{N}(v)$ is the neighbourhood of node v , \tilde{d}_v and \tilde{d}_v^{in} are the renormalised degree and in-degree of node v (see Kipf and Welling [9]), and $\epsilon^{(k)}$ is GIN's central node weighting parameter.

1.1.2. Transfer Learning

Deep neural network and machine learning models are usually trained for a specific task from a random initialisation of their parameters. If the task or nature of the input data changes, the network must be retrained from scratch. This retraining differs from humans, who reuse past knowledge and apply it to new contexts and tasks. The ability to reuse knowledge beyond the context in which it was learnt is known as *transfer learning* [13]. Transfer with neural networks can be achieved by simply reusing a previously-trained network's weights. The transferred weights can either be used as a starting point for

training a network (*fine-tuned transfer*), or used as fixed features on the target task (*frozen-weight transfer*). To formalise transfer learning, Pan and Yang [14] define the notion of *domains* and *tasks* as used in this paper.

Taylor and Stone [13] present various metrics for the evaluation of transfer learning, including *Transfer Ratio*, *Jumpstart* and *Asymptotic Performance*. The Transfer Ratio is the ratio of the total cumulative performance of the transfer learner to the base learner, while Jumpstart is the initial performance improvement by the transfer over the base learner. Asymptotic Performance refers to the improvement made in the final learnt performance in the target task. The figure below describes these metrics.

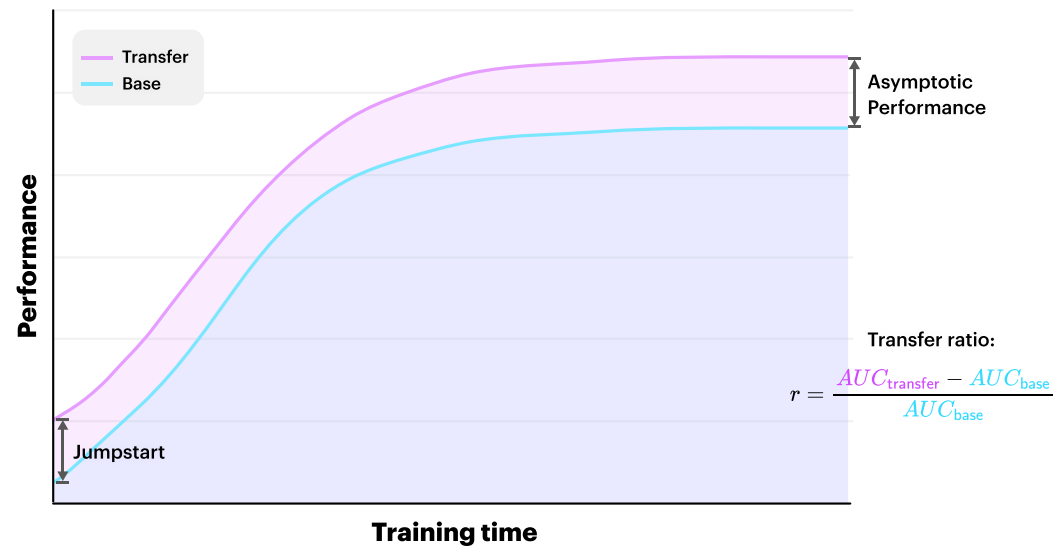


Figure 1. An illustration of the *jumpstart*, *asymptotic performance*, and *transfer ratio* metrics. The transfer ratio is computed using the area under the curve (AUC).

1.1.3. Related Work

Bengio [15] notes that deep learning algorithms seem well suited to transfer learning through learning ‘abstract’ representations. Knowledge is represented at multiple levels, with higher level representations learnt as compositions of lower level features [16]. Kornblith *et al.* [17] and Huh *et al.* [18] investigate transfer learning with CNNs pretrained on ImageNet [19] and find that networks train faster, and achieve improved accuracy as a result.

Partly due to recency, and partly due to the multitude of approaches, not much research exists which investigates transfer learning for GNNs. Hamilton [20] note that little success has been achieved by pretraining GNNs. This may be due to the fact that randomly initialised GNNs extract features that are just as useful as a trained network’s [21].

However, Lee *et al.* [22] do propose a framework to transfer spectral information between source and target tasks for node classification. Experiments on real-world datasets show transfer is most effective when the source and target graphs are similar. Hu *et al.* [23] develop a framework to effectively pretrain GNNs for downstream tasks by pretraining a GNN at the level of individual nodes and the entire graphs. Thus, they describe two techniques to exploit node-level knowledge: context prediction and attribute masking; as well as two approaches for graph-level pretraining. More recently, Dai *et al.* [24] present AdaGCN: a framework for transfer learning based on adversarial domain adaption with GCNs.

2. Materials and Methods

Errica *et al.* [25] note that the experimental settings from GNN research papers are often ambiguous, and the results are not reproducible. Reproducibility is, therefore, a core objective, and as such, we provide the code for our experiments. We utilise the

PyTorch-Geometric [26] library for efficient GPU-optimised implementations of the three selected GNNs [27]. We track our experiments using Comet .ml, and make them publicly available for transparency.

Our experiments pretrain GNNs on various source tasks and fine-tune them on a target task. All experiments are conducted in a fully supervised setting. The experiments track the performance of pretrained models as well as the randomly-initialised models across training. This methodology allows us to compare transferability in terms of the transfer learning metrics described earlier. We make use of synthetic and real-world data in our experiments. To avoid the problems of a lack of meaningful data-splits [25] and the instability of training on small graph datasets [28], we make use of the Open Graph Benchmark (OGB) [29] datasets for our real-world datasets. To ascertain the statistical significance of our results, we employ a pairwise two-sided t-test for identical means of independent sample statistics with the alternative hypothesis ‘greater than’ throughout.

2.1. Node Classification Experimental Design

Node classification involves assigning a class label to an unlabelled node in a graph. Solving this problem involves structural properties of graphs (e.g. node degree), the node’s attributes, or some relationship between them. Practical applications include real-world contexts, such as social networks [30] or knowledge graphs [31]. Below we describe the datasets and experimental methodology used.

2.1.1. Synthetic Data

For meaningful classifications, instances within a single class should be similar. A graph where nodes in the same class are densely connected is said to contain strong *community structure*.

The *Modularity*, defined as

$$M = \frac{1}{2|E|} \sum_{ij} \left(a_{ij} - \frac{d_i d_j}{2|E|} \right) \delta(i, j),$$

is one measure of a graph’s community structure with respect to the *structure* of the network [32], where $\delta(i, j) = 1$ if v_i and v_j belong to the same class, and 0 otherwise.

The *Within Inertia* ratio is another measure of the community structure that takes into account node *attribute* values [33]. Given a partition of the graph’s vertices \mathcal{P} , the Within Inertia is given by:

$$I = \frac{\sum_{C \in \mathcal{P}} \sum_{v \in C} \text{dist}(x_v, g_C)^2}{\sum_{v \in V} \text{dist}(x_v, g)^2},$$

where $\text{dist}(x_{v_i}, x_{v_j})$ is the Euclidean distance between node attribute vectors, g_C is the centre of gravity of the vertices in C , and g the global centre of gravity of all vertices.

To generate synthetic datasets, we make use of *DANCer*: a generator for dynamic attributed networks with community structure [33]. The generator produces graphs with communities using micro-operations (local operations such as removing a node) and macro-operations (community-level operations such as splitting a community). The generator is designed with both structural and attribute homophily [34] in mind. It also models phenomena such as *preferential attachment*, where nodes are more likely to connect with nearby or highly connected nodes [35]. These properties make DANCer an ideal generator for our purposes.

The authors provide four benchmark configurations: one for each combination of strong and weak structural and attribute community structure. We fix a single target task with these datasets (Configuration 4), and pretrain on the other three configurations as described in Table 1. The parentheses indicate the strength of Modularity and Within Inertia respectively — e.g. $M_{\uparrow}I_{\downarrow}$ indicates strong Modularity and weak Within Inertia.

	Modularity		Within Inertia	
Configuration 1 ($M_{\uparrow}I_{\uparrow}$)	<i>Strong</i>	0.64	<i>Strong</i>	0.37
Configuration 2 ($M_{\uparrow}I_{\downarrow}$)	<i>Strong</i>	0.64	<i>Weak</i>	0.47
Configuration 3 ($M_{\downarrow}I_{\uparrow}$)	<i>Weak</i>	0.32	<i>Strong</i>	0.39
Configuration 4 ($M_{\downarrow}I_{\downarrow}$)	<i>Weak</i>	0.28	<i>Weak</i>	0.99

Table 1: The four configurations of the synthetic node classification datasets. The average modularity and Within Inertia ratios are computed on the generated datasets.

2.1.2. Real-World Data

A common node classification domain is *citation networks*, where each node in the graph represents a publication, and edges indicates citations. We perform several transfer learning experiments using OGB real-world citation networks, and complement the experiments with synthetic data. In particular, we select Arxiv and MAG—both are directed citation networks, where each node has an attribute vector containing a 128-dimensional word embedding of the paper. In addition, a paper’s year of publication is also associated with its node in the network.

Arxiv contains 169,343 Computer Science papers; and the task is to predict which of the 40 subject areas a paper belongs to. MAG is taken from a subset of the Microsoft-Academic-Graph [36], and contains four types of node entities: papers, authors, institution and field of study. For consistency we will only make use of the papers, which consist of 736,389 nodes, making it a much larger and more complex network than Arxiv. The task here is to predict which of 349 venues (conferences or journals) each paper belongs to. OGB also provides model evaluators, which use the standard *accuracy score*.

To evaluate how MAG transfers to itself, we split it into a *source* and a *target* graph. Papers from 2010–2014 are placed in the source split, and those from 2015–2019 belong to the target split. Any edges between nodes in separate splits are removed. Table 2 lists the statistics of the above datasets.

	Nodes	Edges	Features	Modularity	Within Inertia	Classes	Metric
Arxiv	169,343	1,166,243	128	0.495	0.890	40	Accuracy
MAG (Source)	402,598	1,615,644	128	0.299	0.813	349	Accuracy
MAG (Target)	333,791	1,390,589	128	0.286	0.806	349	Accuracy

Table 2: Statistics of real-world node classification datasets used in our experiments.

2.1.3. Experimental Methodology

Table 3 presents the source and target tasks for our experiments using both real-world and synthetic datasets. We evaluate transfer from both Arxiv and the MAG source split to the target MAG split. Lastly, to investigate how important attributes are for our node classification tasks, we damage the node attributes for both Arxiv and the MAG source graph. To damage the attributes, we replace the attributes with Gaussian distributed random noise with a mean of 0 and a standard deviation of 1.

We perform 10 runs of each of the six sets of experiments for each of the three GNNs. We use the same network architecture used by OGB [29] for their experiments on Arxiv and MAG, which allows us to compare the performance for the base models as a sanity check. The network comprises of three GNN layers: with an input dimensionality of 128, an output dimensionality of 349 (for MAG), and a hidden dimensionality of 256. We train the networks on the target task for 2000 epochs using the Adam optimiser [37]. The best

	#	Source Task	Target Task
Real-world	1	Base [Random seed]	
	2	Arxiv	
	3	Arxiv [Damaged features]	MAG
	4	MAG (<i>Source split</i>) [Old layer]	(<i>Target Split</i>)
	5	MAG (<i>Source split</i>)	
	6	MAG (<i>Source split</i>) [Damaged features]	
Synthetic	1	Base [Random seed]	
	2	Configuration 1 ($M_{\uparrow}I_{\uparrow}$)	Configuration 4
	3	Configuration 2 ($M_{\uparrow}I_{\downarrow}$)	($M_{\downarrow}I_{\downarrow}$)
	4	Configuration 3 ($M_{\downarrow}I_{\uparrow}$)	

Table 3: Experiments conducted for node classification using both real-world and synthetic datasets.

performing learning rate for each GNN is selected and fixed across our 6 experiment sets. GCN, GraphSAGE and GIN are all trained with a learning rate of 0.001 in this case.

For synthetic data experiments, we generate 10 unique graphs for each of *Configurations 1, 2, and 3*. Throughout, we use a single instance of *Configuration 4* so that the target task is fixed. The task is 5-class node classification, and we train the models for 2000 epochs using the Adam optimiser with a learning rate of 0.01 for GCN and GraphSAGE, and 0.001 for GIN

2.2. Graph Classification Experimental Design

Graph classification is the problem of categorising whole graphs. To investigate GNN transfer for graph classification, we conduct experiments involving both real-world and synthetic problems. We follow the same experimental procedure as for node classification. We want to evaluate whether a dataset contains graphs with community structure at a *graph-level*. With this aim, we present an extension of the concept of community structure from node to graph level for both structural and attribute properties. The Within Inertia is a general measure of community structure since it does not depend on graph-theoretic properties, but only on the Euclidean distance between data points. Given a dataset D , and a partition of classes \mathcal{P} , a general form of the Within Inertia can be written as:

$$\frac{\sum_{C \in \mathcal{P}} \sum_{i \in C} \text{dist}(\rho_i, g_C)^2}{\sum_{i \in D} \text{dist}(\rho_i, g)^2}, \quad (1)$$

where ρ_i is a graph property we are interested in measuring for graph i in class C , g_C is the centre of gravity of the graphs in C , and g is the global center of gravity of all the graphs. The Euclidean distance $\text{dist}(\cdot)$ may be replaced by other distance metrics if the generation process is unknown [38,39].

We replace ρ_i in the Eq. 1 with any property we want to measure community structure for. For *attribute* community structure we substitute the mean of each graph's attribute matrix, \bar{X} , for ρ_i :

$$I^A = \frac{\sum_{C \in \mathcal{P}} \sum_{i \in C} \text{dist}(\bar{X}_i, g_C)^2}{\sum_{i \in D} \text{dist}(\bar{X}_i, g)^2}.$$

There is no consensus on how to approach *structural* community structure. There are numerous ways of measuring graph similarity [40], the most common being graph-edit distance [41]. Another approach is to compare graph spectra. However, these methods are slow to compute and are thus not useful for measuring entire datasets with multiple graphs.

Modularity takes node degrees as a valuable property for determining community structure for nodes. Since we want to measure community structure at the graph level, we use the average node degree for a graph. We substitute the average node degree, \bar{d}_i , for ρ_i in Eq. 1. This serves as the *structural* community structure measure for our datasets:

$$I^S = \frac{\sum_{C \in \mathcal{P}} \sum_{i \in C} \text{dist}(\bar{d}_i, g_C)^2}{\sum_{i \in D} \text{dist}(\bar{d}_i, g)^2}.$$

2.2.1. Synthetic Data

We present a novel process for generating synthetic datasets for the task of graph classification. To create a meaningful graph classification task, we parameterise a generator to control the community structure for both structure and attributes. To do this, we generate the dataset using the following four steps:

1. Create a random n -class classification problem with a sample X and labels y .
2. For each label y_i in y , generate several graphs and set their node attributes to the relevant example from X . Label these graphs with y_i .
3. Optionally, swap the labels assigned to some of the graphs to weaken community structure.
4. Optionally, replace node attributes with noise to weaken attribute community structure.

Steps 1 and 2 create community structure for attributes and structure respectively, and the final two steps weaken the community structure if selected to do so. Figure 2 illustrates this process, where the blue blocks create community structure and the pink blocks weaken it.

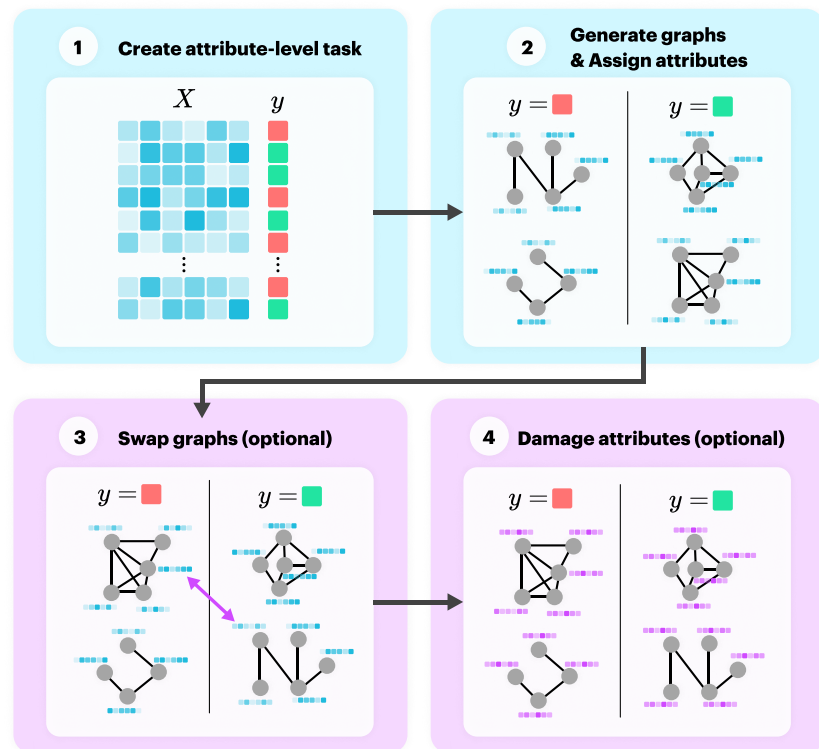


Figure 2. Generation process for synthetic graph classification datasets

The generator takes the following parameters as input:

- `num_classes`: the number of classes or labels in the dataset,

- `n_per_class`: the number of graphs to generate per class,
- `n_features`: the length of the node feature vector,
- `percent_swap`: the percentage of graphs to swap,
- `percent_damage`: the percentage of graphs where node attributes are to be damaged.

The `num_classes`, `n_per_class` and `n_features` parameters allow for dataset level properties to be varied, while `percent_swap` and `percent_damage` influence the structural and attribute community structure respectively. We fix the size of the graphs at 30 nodes. More details regarding this generation process are provided in the appendix.

Similarly to the synthetic node classification datasets, we generate four configurations for each strong and weak community structure combination (see Table 4). We again indicate the strength of the Structural and Attribute Within Inertia using arrows. For example, $I_{\uparrow\downarrow}^S I_{\uparrow\uparrow}^A$ indicates strong Structural Within Inertia and weak Attribute Within Inertia. We select Configuration 8 ($I_{\uparrow\uparrow}^S I_{\uparrow\uparrow}^A$) as our target task, and use the remaining three as source tasks.

	w.i.-struct	w.i.-attr	percent_swap	percent_damage
Configuration 5 ($I_{\uparrow\downarrow}^S I_{\uparrow\uparrow}^A$)	Weak	Weak	0.95	0.95
Configuration 6 ($I_{\uparrow\downarrow}^S I_{\downarrow\downarrow}^A$)	Strong	Weak	0.92	0.95
Configuration 7 ($I_{\uparrow\downarrow}^S I_{\uparrow\downarrow}^A$)	Weak	Strong	0.95	0.92
Configuration 8 ($I_{\uparrow\uparrow}^S I_{\uparrow\uparrow}^A$)	Strong	Strong	0.92	0.92

Table 4: The four configurations of synthetic graph classification datasets.

2.2.2. Real-World Data

Many real-world problems within bioinformatics and chemistry present themselves as graph classification problems. We select datasets from OGB where the task is *molecular property prediction*: BBBP and HIV. BBBP (Blood-Brain Barrier Penetration) is a physiological dataset where the task is to predict whether a given compound penetrates the blood-brain barrier or not [42]. HIV is a biophysics dataset where the task is to predict whether a compound has anti-HIV activity or not.

Both datasets are pre-processed in the same manner and are both binary classification tasks. Nodes are atoms and chemical bonds are edges, while node attributes are 9-dimensional and contain information about atomic properties. BBBP is a much smaller dataset than HIV so we split HIV into a source and target split similar to the real-world node classification experiments: half the HIV is randomly sampled for each split, and this sample is kept fixed. A summary of the datasets is given in Table 5.

	No. Graphs	Average Nodes	Features	I^S	I^A	Classes	Metric
BBBP	2,039	24.06	9	0.99	0.98	2	ROC-AUC
HIV (Source split)	20,563	25.49	9	0.99	0.99	2	ROC-AUC
HIV (Target split)	20,564	25.53	9	0.99	0.99	2	ROC-AUC

Table 5: Statistics of real-world graph classification datasets used in our experiments.

2.2.3. Experimental Methodology

We conduct experiments similar to those for real-world node classification. The target task, HIV target split is fixed, and we pretrain our GNNs on BBBP and the HIV source split and then evaluate them on the target task. We also evaluate the transfer performance where the models are pretrained on the source datasets with *damaged* node attributes: i.e. where the node attributes are replaced by Gaussian distributed random noise with a mean of 0 and a standard deviation of 1. The experiments are described in Table 6.

	#	Source Task	Target Task
Real-world	1	Base [Random seed]	HIV (Target Split)
	2	BBBP	
	3	BBBP [Damaged features]	
	4	HIV (Source split)	
	5	HIV (Source split) [Damaged features]	
Synthetic	1	Base [Random seed]	Configuration 8 ($I_{\uparrow}^S I_{\uparrow}^A$)
	2	Configuration 5 ($I_{\downarrow}^S I_{\downarrow}^A$)	
	3	Configuration 6 ($I_{\downarrow}^S I_{\downarrow}^A$)	
	4	Configuration 7 ($I_{\downarrow}^S I_{\uparrow}^A$)	

Table 6: Experiments conducted for graph classification using both real-world and synthetic datasets.

3. Results

3.1. Node Classification Results

3.1.1. Real-World Data

Model	Source Task → MAG-Target	Transfer Ratio	Jumpstart	Asymptotic Performance
GCN	Control MAG-Source [Damaged]	0.011 0.006	0.000 0.001	-0.001 0.002
	MAG-Source [Old layer]	0.042 0.007	0.228 0.023	0.003 0.003
	MAG-Source	0.032 0.011	0.001 0.002	0.002 0.003
	Arxiv	0.021 0.007	0.001 0.001	0.008 0.002
	Arxiv [Damaged]	0.028 0.006	0.000 0.001	0.009 0.002
G'SAGE	Control MAG-Source [Damaged]	0.023 0.037	0.000 0.001	0.000 0.009
	MAG-Source [Old layer]	0.044 0.041	0.239 0.018	0.001 0.010
	MAG-Source	0.046 0.041	0.000 0.001	0.003 0.011
	Arxiv	0.028 0.040	0.001 0.001	0.007 0.010
	Arxiv [Damaged]	-0.050 0.033	0.001 0.001	-0.021 0.009
GIN	Control MAG-Source [Damaged]	0.030 0.011	-0.002 0.004	0.000 0.002
	MAG-Source [Old layer]	0.183 0.008	0.226 0.004	0.024 0.002
	MAG-Source	0.089 0.007	-0.001 0.004	0.009 0.001
	Arxiv	0.048 0.009	-0.001 0.004	0.005 0.001
	Arxiv [Damaged]	0.061 0.016	-0.001 0.004	0.006 0.003

Table 7: Transfer metrics for real-world node classification experiments (10 runs). Bold results are positive and statistically greater than the control at $p = 0.1$. We evaluate significance for each model/metric combination.

In Figure 3 we see the training curves for the experiments in Table 3. We note for GCN, all the pretrained model curves rise above the base model, indicating we have positive knowledge transfer. This phenomenon is noted for both GraphSAGE (except for Arxiv [Damaged]), and GIN. Since the pretraining datasets are all citation networks, positive transfer is a reasonable outcome.

Concerning the Transfer Ratios in Table 7 we note that only GIN and GCN have significant transfer from the completely new Arxiv dataset. Interestingly for GCN and GIN we note that Arxiv with damaged attributes, in absolute terms, performs somewhat better than just Arxiv, indicating that graph characteristics beyond attribute values are being used for transfer. Although some of the GraphSAGE Transfer Ratios are positive, these are not statistically better than the control. Turning to Table 8 we note that GIN statistically

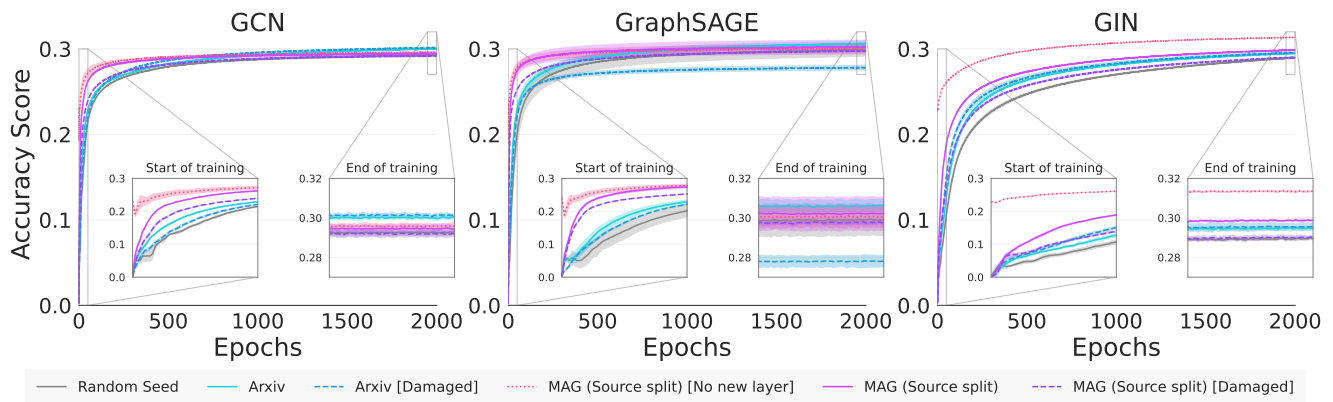


Figure 3. Real-world node classification training curves

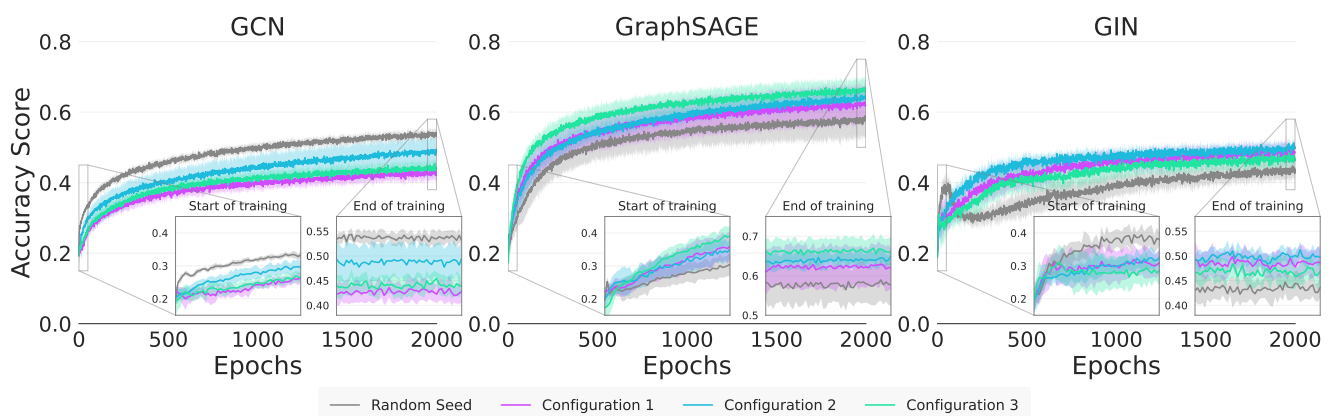


Figure 4. Synthetic node classification training curves

Source Task → <i>MAG-Target</i> (Spectral Dist.)	Model	Transfer Ratio		Jumpstart		Asymptotic Performance	
MAG-Source [Old layer]	GCN	0.042	0.007	0.228	0.023	0.003	0.003
	G'SAGE	0.044	0.041	0.239	0.018	0.001	0.010
	GIN	0.183	0.008	0.226	0.004	0.024	0.002
MAG-Source	GCN	0.032	0.011	0.001	0.002	0.002	0.003
	G'SAGE	0.046	0.041	0.000	0.001	0.003	0.011
	GIN	0.089	0.007	-0.001	0.004	0.009	0.001
Arxiv	GCN	0.021	0.007	0.001	0.001	0.008	0.002
	G'SAGE	0.028	0.040	0.001	0.001	0.007	0.010
	GIN	0.048	0.009	-0.001	0.004	0.005	0.001

Table 8: Transfer metrics for real-world node classification experiments (10 runs). Bold results are not statistically greater than the best at $p = 0.1$. We evaluate significance for each source-task/metric combination.

always outperforms the other GNNs in this metric, indicating that GIN benefits the most from sharing knowledge from the source domains.

The *MAG (Source split)* [Old layer] tasks, in Table 7, have a greater Jumpstart than the rest, since the output layer does not need to be retrained. We note that both GCN

and GraphSAGE show significant Jumpstart with the completely new task Arxiv. In fact GraphSAGE exploits graph characteristics beyond attribute values as evidenced by the Arxiv [Damaged] results. In Table 8 we note that GCN is either on par or significantly better than the other GNNs across all datasets on this metric; however, the result is not compelling.

All GNNs achieve significant positive asymptotic performance above the control on the new Arxiv task. GraphSAGE is inconsistent and does not always show transfer at the end of training. Despite the huge Jumpstart with MAG (*Source split*) [Old layer], only GIN retains a large absolute improvement in Asymptotic Performance. Both GCN and GIN once again exploit structural characteristics beyond the attribute values as evidenced by the Arxiv [Damaged] results. In Table 8 we see a mixture of best performing GNNs with both GCN and GraphSAGE performing well on the completely new Arxiv task.

Takeaway 1: We have statistical evidence that transfer to a new task for node classification does occur across all metrics and GNNs. We demonstrate that GCN and GIN exploit structural rather than attribute information for achieving a positive Transfer Ratio and Asymptotic Performance; as does GraphSAGE for Jumpstart.

Next we consider our synthetic data to interrogate exactly which characteristics of graphs are being transferred by the above GNNs.

3.1.2. Synthetic Data

Model	Source Task	Transfer Ratio	Jumpstart	Asymptotic performance
GCN	C.1 - $M_{\uparrow}I_{\uparrow}$	-0.203 0.031	0.031 0.044	-0.103 0.026
	C.2 - $M_{\uparrow}I_{\downarrow}$	-0.108 0.065	0.012 0.039	-0.036 0.038
	C.3 - $M_{\downarrow}I_{\uparrow}$	-0.176 0.026	0.001 0.068	-0.092 0.020
G'SAGE	C.1 - $M_{\uparrow}I_{\uparrow}$	0.083 0.086	0.026 0.024	0.041 0.042
	C.2 - $M_{\uparrow}I_{\downarrow}$	0.100 0.102	0.004 0.031	0.073 0.051
	C.3 - $M_{\downarrow}I_{\uparrow}$	0.169 0.139	-0.024 0.042	0.082 0.084
GIN	C.1 - $M_{\uparrow}I_{\uparrow}$	0.161 0.099	0.010 0.059	0.050 0.042
	C.2 - $M_{\uparrow}I_{\downarrow}$	0.211 0.076	0.001 0.046	0.061 0.029
	C.3 - $M_{\downarrow}I_{\uparrow}$	0.112 0.070	-0.006 0.060	0.031 0.023

Table 9: Transfer metrics for synthetic node classification (10 runs). Bold results are not statistically greater than the best at $p = 0.1$. We evaluate significance for each model/metric combination.

In Table 9 we note that the performance of GIN is as a result of the strong Modularity in *Configurations 1* ($M_{\uparrow}I_{\uparrow}$) and *2* ($M_{\uparrow}I_{\downarrow}$). For GraphSAGE, this distinction between Modularity and Within Inertia is less clear as the results for *Configurations 2* ($M_{\uparrow}I_{\downarrow}$) and *3* ($M_{\downarrow}I_{\uparrow}$) are statistically equivalent. The absolute performance of *Configuration 3* ($M_{\downarrow}I_{\uparrow}$) for GraphSAGE does suggest that Within Inertia is being exploited, but further investigation is required.

For Jumpstart, we are unable to see any significant difference in Table 9 across all configurations for the GCN and GIN, making it unclear as to which of Modularity or Within Inertia is responsible for the positive transfer. GraphSAGE is significantly better in *Configuration 1* ($M_{\uparrow}I_{\uparrow}$) indicating both the Strong Modularity and Within Inertia are being exploited, supporting our real-world assertion that it exploits graph characteristics beyond attribute values. We do note in Table 9 that all methods show positive Jumpstart

on Configuration 1 ($M_{\uparrow}I_{\uparrow}$) and 2 ($M_{\uparrow}I_{\downarrow}$) suggesting that it might be the Strong Modularity that all the GNNs are exploiting.

With Asymptotic Performance, our results do not allow us to say anything about which of structure or attributes GraphSAGE is exploiting. However, GIN, as evidenced in Table 9, is able to build on the Modularity from Configuration 1 ($M_{\uparrow}I_{\uparrow}$) as the Weak Inertia of Configuration 2 ($M_{\uparrow}I_{\downarrow}$) is not significantly better.

Takeaway 2: In general, GIN predominately exploits Strong Modularity for transfer—while GraphSAGE can exploit both Modularity and Within Inertia for Jumpstart—in support of our real-world data findings.

3.2. Graph Classification Results

3.2.1. Real-World Data

Model	Source Task \rightarrow HIV-Target	Transfer Ratio		Jumpstart		Asymptotic performance	
Control	HIV-Source [Damaged]	-0.002	0.015	0.002	0.017	0.014	0.012
GCN	HIV-Source	0.065	0.010	0.148	0.009	0.031	0.017
	BBBP	0.036	0.012	0.047	0.016	0.026	0.011
	BBBP [Damaged]	-0.007	0.013	-0.008	0.021	0.000	0.011
Control	HIV-Source [Damaged]	-0.069	0.023	-0.029	0.049	-0.038	0.021
G'SAGE	HIV-Source	0.030	0.006	0.160	0.016	0.011	0.014
	BBBP	0.048	0.008	0.072	0.011	0.035	0.009
	BBBP [Damaged]	-0.064	0.058	-0.067	0.052	-0.042	0.064
Control	HIV-Source [Damaged]	-0.197	0.037	0.039	0.048	-0.130	0.051
GIN	HIV-Source	0.033	0.016	0.186	0.045	0.029	0.040
	BBBP	-0.059	0.033	0.026	0.046	-0.081	0.075
	BBBP [Damaged]	-0.157	0.038	-0.013	0.017	-0.136	0.049

Table 10: Transfer metrics for real-world graph classification experiments (10 runs). Bold results are statistically greater than the control at $p = 0.1$. We evaluate significance for each model/metric combination.

Figure 5 shows that GCN and GraphSAGE have more stable training curves than GIN, which is poorer in its performance. We observe clear transfer with GCN and GraphSAGE for both undamaged source tasks. In addition, we observe negative transfer for both damaged source tasks for GCN and GraphSAGE. GIN achieves positive self-transfer with HIV (*Source split*), and negative transfer with the remaining pretrainings. GIN's training curves also show a decay over training with BBBP pretrainings. The training curves indicate GraphSAGE and GIN suffer from worse negative transfer than GCN.

From Table 10, we see that across the transfer metrics, GCN and GraphSAGE achieved significant positive Transfer Ratios for HIV (*Source split*) and BBBP when compared to the control. This result indicates that they are able to transfer to a completely new task. GIN shows transfer from the similar task HIV (*Source split*) but not from the new task BBBP for any of the metrics. The biggest jumpstart is seen with HIV (*Source split*), which is understandable since it is a self-transfer task. None of the GNNs for any of the tasks show

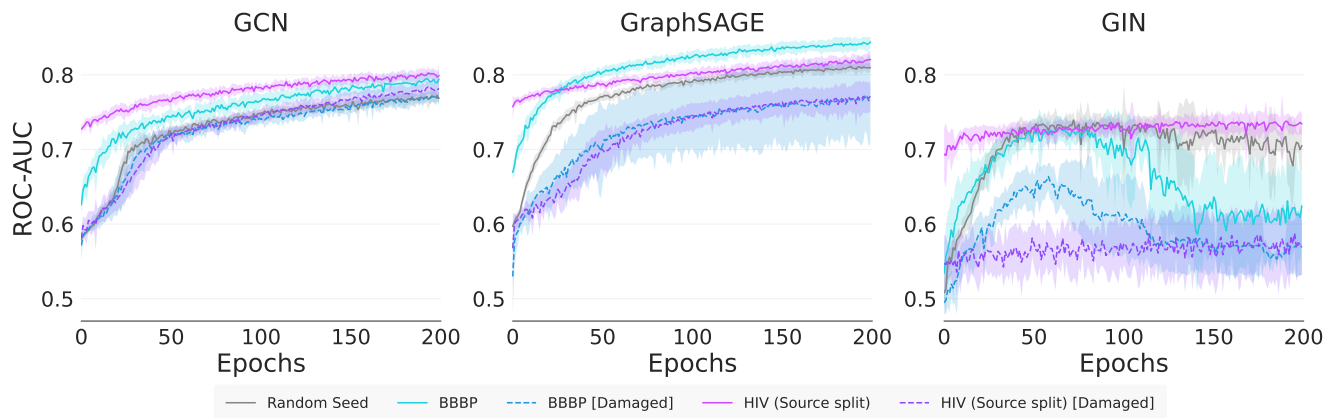


Figure 5. Real-world graph classification training curves

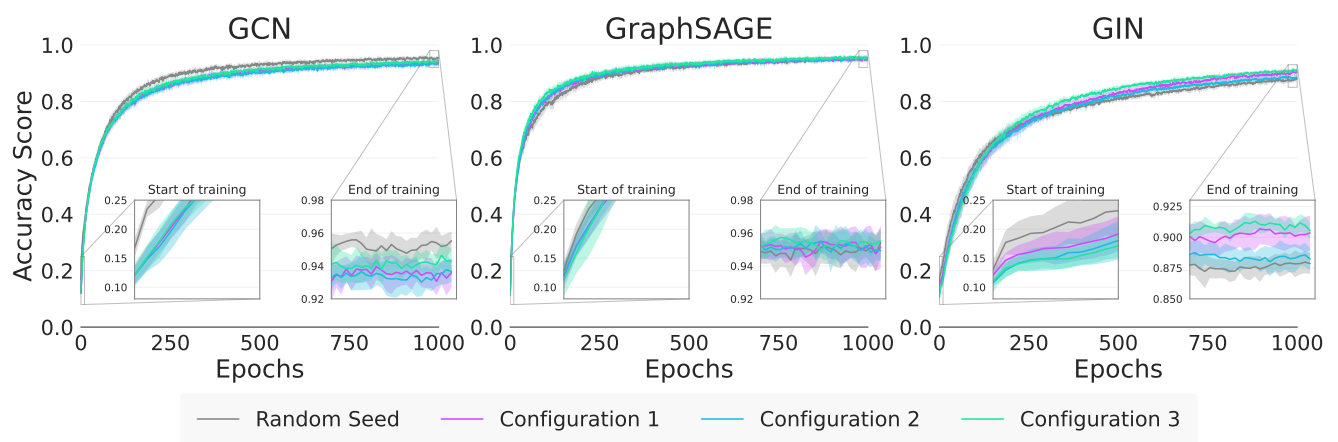


Figure 6. Synthetic graph classification training curves

Source Task → HIV-Target	Model	Transfer Ratio		Jumpstart		Asymptotic performance	
HIV-Source	GCN	0.065	0.010	0.148	0.009	0.031	0.017
	G'SAGE	0.030	0.006	0.160	0.016	0.011	0.014
	GIN	0.033	0.016	0.186	0.045	0.029	0.040
BBBP	GCN	0.036	0.012	0.047	0.016	0.026	0.011
	G'SAGE	0.048	0.008	0.072	0.011	0.035	0.009
	GIN	-0.059	0.033	0.026	0.046	-0.081	0.075

Table 11: Transfer metrics for real-world graph classification experiments (10 runs). Bold results are not statistically greater than the best at $p = 0.1$. We evaluate significance for each source-task/metric combination.

any significant transfer from the damaged task, indicating that the node attributes are likely exploited over graph structural characteristics.

In Table 11 we note that GraphSAGE is significantly better than all other GNNs across all metrics when transferring from the completely new BBBP task. For the transfer

from the more similar HIV (*Source split*) results are mixed with GCN requiring further experimentation to determine outperformance.

Takeaway 3: We have significant statistical evidence to support that transfer happens for graph classification for both GCN and GraphSAGE across all metrics considered. We can also reject the hypothesis that the transfer is as a result of graph structure beyond the node attributes alone.

In the following section, we consider synthetic data to further investigate which structural characteristics of graphs are being transferred by the GNNs.

3.2.2. Synthetic Data

When considering the graph classification Transfer Ratios in Table 12, only GraphSAGE and GIN show any positive transfer. Interestingly, for both models, the result is significant for *Configuration 7* ($I_{\downarrow}^S I_{\uparrow}^A$) indicating that it is the strong Attribute Within Inertia that is being transferred. The amount of Jumpstart achieved by all GNNs is low, indicating that transfer is not achieved at the start of training, and that the pretrained model performs similarly to the base models initially. For graph classification on the synthetic data, we note from Table 12 that none of the considered methods are able to achieve positive Jumpstart. These values are not discussed further here. The values for the asymptotic performance show similar results to the Transfer Ratios.

Since there is minimal Jumpstart, any positive or negative transfer appears to be achieved by the GNNs towards the end of training. GraphSAGE and GIN both achieve positive Asymptotic Improvements as seen Table 12; however for GraphSAGE these values are not significantly different from negative transfer. We note for both GraphSAGE and GIN that the best transfer occurs for *Configuration 5* ($I_{\downarrow}^S I_{\downarrow}^A$) and *7* ($I_{\downarrow}^S I_{\uparrow}^A$). In absolute terms, we might infer that due to the higher values in *Configuration 7* ($I_{\downarrow}^S I_{\uparrow}^A$), transfer results from the strong Attribute Within Inertia. However, these differences are not statistically significant and require further experimentation to confirm.

The fact that no Jumpstart but improved Asymptotic Performance is observed is interesting: indicating that the transferred knowledge is not immediately useful but rather leads to better performance on downstream training. This observation would be supported by an argument that the transferred knowledge is not in the linear output layers but instead available in the deeper non-linear feature layers and exposed later in training.

Takeaway 4: There is significant evidence that GraphSAGE and GIN exploit Strong Attribute Within Inertia in order to achieve transfer. These results support our real-world findings with respect to GraphSAGE.

4. Discussion

Our research presented several contributions for understanding transfer learning using graph neural networks. We proposed a framework for evaluating transfer with GNNs by testing various source tasks on a fixed target task. We employed this framework, along with transfer learning metrics and notions of community structure, to evaluate the transferability of three useful GNNs: GCN, GraphSAGE and GIN. We tested and compared these models using real-world and synthetic graph data for node classification and graph classification contexts. In addition, we presented a novel procedure for generating synthetic datasets for graph classification.

All three of the GNNs we selected can transfer knowledge across training on the target task. GCN and GIN can exploit Strong Modularity in the source task, while GraphSAGE can leverage both structural and attribute information for node classification. For graph classification, it is less clear that any model exploits either attribute or structural community structure, and they appear to leverage a combination of the two to achieve transfer.

Model	Source Task	Transfer Ratio		Jumpstart		Asymptotic performance	
GCN	C.5 - $I_{\downarrow}^S I_{\downarrow}^A$	-0.026	0.012	-0.046	0.024	-0.019	0.011
	C.6 - $I_{\downarrow}^S I_{\downarrow}^A$	-0.029	0.009	-0.047	0.022	-0.019	0.009
	C.7 - $I_{\downarrow}^S I_{\uparrow}^A$	-0.020	0.005	-0.046	0.028	-0.012	0.012
G'SAGE	C.5 - $I_{\downarrow}^S I_{\downarrow}^A$	0.007	0.006	-0.009	0.030	-0.003	0.011
	C.6 - $I_{\downarrow}^S I_{\downarrow}^A$	0.008	0.010	-0.006	0.024	-0.005	0.014
	C.7 - $I_{\downarrow}^S I_{\uparrow}^A$	0.015	0.011	-0.016	0.038	0.001	0.011
GIN	C.5 - $I_{\downarrow}^S I_{\downarrow}^A$	0.012	0.021	-0.010	0.020	0.025	0.016
	C.6 - $I_{\downarrow}^S I_{\downarrow}^A$	0.001	0.016	-0.023	0.017	0.004	0.016
	C.7 - $I_{\downarrow}^S I_{\uparrow}^A$	0.027	0.017	-0.021	0.026	0.027	0.013

Table 12: Transfer metrics for synthetic graph classification (10 runs). Bold results are not statistically greater than the best at $p = 0.1$. We evaluate significance for each model/metric combination.

5. Conclusions

This research begins to standardise the procedure and metrics for evaluating transfer learning using GNNs. Research on deep learning with graphs is expanding rapidly, and understanding how we can achieve effective transfer is therefore of great benefit. There remains large scope for future research. Our results considered node and graph classification; but our experiments for transfer learning may be extended to other common graph domains such as link prediction and edge classification. Another avenue for future research is to repeat our experiments with other GNNs such as Graph Attention Network [43], and other Graph Network types described by Battaglia *et al.* [6].

Author Contributions: Conceptualization, N.K., S.J. and T.v.Z.; methodology, N.K.; software, N.K.; validation, N.K.; formal analysis, N.K.; investigation, N.K., S.J. and T.v.Z.; resources, S.J. and T.v.Z.; data curation, N.K.; writing—original draft preparation, N.K., S.J. and T.v.Z.; writing—review and editing, N.K., S.J. and T.v.Z.; visualization, N.K.; supervision, S.J. and T.v.Z.; project administration, S.J. and T.v.Z.; funding acquisition, N.K., S.J. and T.v.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by the National Research Foundation of South Africa grant number 122158.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: <https://ogb.stanford.edu>.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A Data Generation for Graph Classification

As described in the main article, we present a novel approach for synthetic graph classification datasets. We want to be able to control the level of community structure for both structure and attributes. To this end we generate the datasets in the following four steps:

Step 1: Create an attribute-level task

The first step in the generation process is to create a attribute-level classification task, where vectors of length $n_features$ are generated belonging to $num_classes$ classes. The total number of these vectors is $num_classes \times n_per_class \times 30$, so that each node in each graph for each class has an attribute vector that can be assigned to it. We follow Morris *et al.* [44] in generating this attribute level task using the `scikit-learn` library.¹ This tool generates the classification task using a modified algorithm from Guyon [45]. After this step the attributes have a high level of community structure.

Step 2: Generate graphs and assign attributes to the graphs

Now that we have attribute vectors that are labelled, we want to generate graphs to assign them to. We want the graphs in each class to have different average degrees, so that strong community structure in terms of $w.i.struct$ exists. A common and useful graph generation algorithm is the *Barabási-Albert* (BA) model [46]. The BA algorithm models preferential attachment, and takes in two parameters: the number of nodes n , and m the number of edges to attach from a new node to existing nodes while growing the graph.

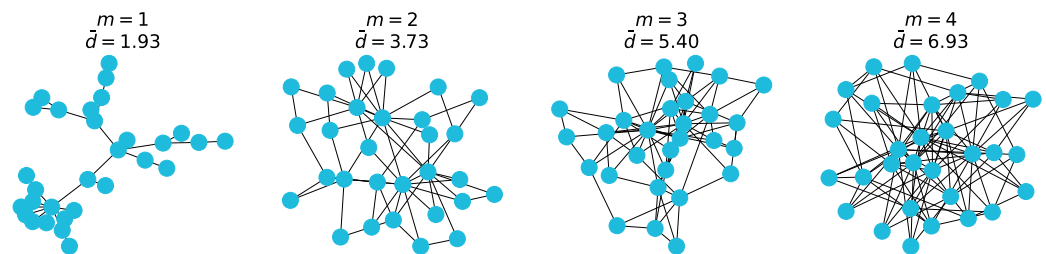


Figure A1. Average degrees \bar{d} for varying m in the Barabási-Albert model.

Varying the m parameter changes the connectivity of a generated graph, and thus its average node degree. This can be seen in the figure above. By assigning graphs generated with different values of m to different classes, we ensure structural community structure with respect to average node degrees.

Once n_per_class graphs are generated for $num_classes$ with different m values, the corresponding attribute vectors from the previous step are assigned to graphs with the same label. At the end of this step, we have a labelled dataset with strong community structure for both nodes and attributes.

Step 3: Swap graphs

This step weakens the structural community structure of the dataset by swapping graphs. This is an optional step, and the extent to which the community structure is weakened is controlled by the `percent_swap` parameter. This parameter may be in the range $[0, 1]$, and the default value is 0 (no graphs are swapped). A random sample of pairs of graphs to swap is selected, corresponding to the specified percentage of the dataset. By swapping more graphs, the classes have less distinct average node degrees compared to one another, resulting in weaker community structure. This is demonstrated in Figure A2.

Step 4: Damage attributes

The final step weakens attribute community structure by replacing node attribute vectors with random noise. This step is also optional, and is controlled by the `percent_damage` parameter. This parameter also has a range of $[0, 1]$, with a default value of 0, and defines the percentage of graphs which will have their node attributes damaged (replaced with random values). The higher the percentage is, the less distinct the attributes from

¹ See https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html.

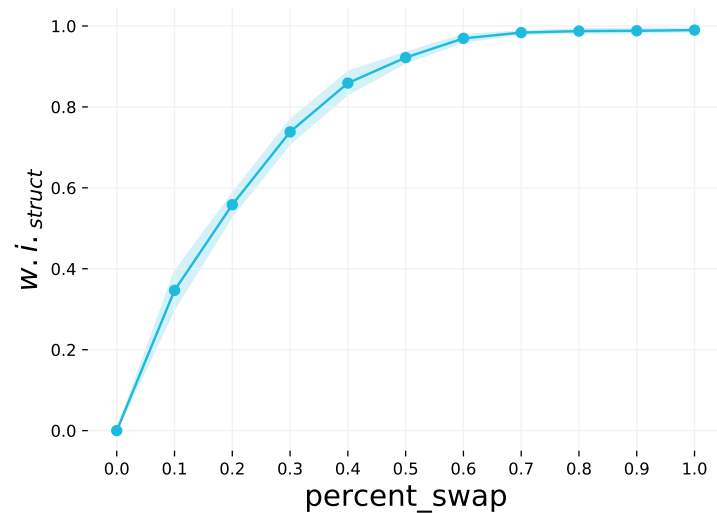


Figure A2. The effect of varying the `percent_swap` parameter on `w.i.struct`. The shaded region is the 1σ interval variance over 10 runs.

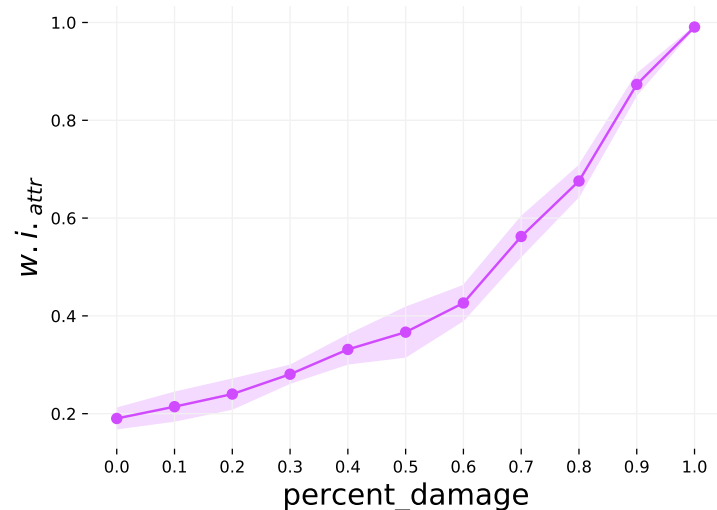


Figure A3. The effect of varying the `percent_damage` parameter on `w.i.attr`. The shaded region is the 1σ variance over 10 runs.

different classes are from one another, and thus a weaker community structure. This is demonstrated in Figure A3.

References

1. Pouyanfar, S.; Sadiq, S.; Yan, Y.; Tian, H.; Tao, Y.; Reyes, M.P.; Shyu, M.L.; Chen, S.C.; Iyengar, S.S. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)* **2018**, *51*, 1–36.
2. Bronstein, M.M.; Bruna, J.; LeCun, Y.; Szlam, A.; Vandergheynst, P. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine* **2017**.
3. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural computation* **1997**.
4. LeCun, Y.; Bengio, Y.; others. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks* **1995**.
5. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Philip, S.Y. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* **2020**.
6. Battaglia, P.W.; Hamrick, J.B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; others. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* **2018**.
7. Hendrycks, D.; Lee, K.; Mazeika, M. Using pre-training can improve model robustness and uncertainty. *International Conference on Machine Learning*, 2019.

8. Barabási, A.L.; others. *Network Science*; Cambridge University Press, 2016.
9. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations* **2017**.
10. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 2017.
11. Xu, K.; Hu, W.; Leskovec, J.; Jegelka, S. How Powerful are Graph Neural Networks? *International Conference on Learning Representations*, 2018.
12. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural message passing for quantum chemistry. *International Conference on Machine Learning* **2017**.
13. Taylor, M.E.; Stone, P. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* **2009**.
14. Pan, S.J.; Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* **2009**.
15. Bengio, Y. Deep learning of representations for unsupervised and transfer learning. *ICML Workshop on Unsupervised and Transfer Learning*, 2012.
16. Yosinski, J.; Clune, J.; Bengio, Y.; Lipson, H. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems*, 2014.
17. Kornblith, S.; Shlens, J.; Le, Q.V. Do better ImageNet models transfer better? *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
18. Huh, M.; Agrawal, P.; Efros, A.A. What makes ImageNet good for transfer learning? *NIPS Large Scale Computer Vision Systems Workshop (Oral)* **2016**.
19. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.
20. Hamilton, W.L. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* **2020**.
21. Velickovic, P.; Fedus, W.; Hamilton, W.L.; Liò, P.; Bengio, Y.; Hjelm, R.D. Deep Graph Infomax. *International Conference on Learning Representations*, 2019.
22. Lee, J.; Kim, H.; Lee, J.; Yoon, S. Transfer learning for deep learning on graph-structured data. *AAAI Conference on Artificial Intelligence*, 2017.
23. Hu, W.; Liu, B.; Gomes, J.; Zitnik, M.; Liang, P.; Pande, V.; Leskovec, J. Strategies for Pre-training Graph Neural Networks. *International Conference on Learning Representations*, 2019.
24. Dai, Q.; Shen, X.; Wu, X.M.; Wang, D. Network Transfer Learning via Adversarial Domain Adaptation with Graph Convolution. *International Conference on Learning Representations* **2019**.
25. Errica, F.; Podda, M.; Bacciu, D.; Micheli, A. A fair comparison of graph neural networks for graph classification. *International Conference on Learning Representations* **2020**.
26. Fey, M.; Lenssen, J.E. Fast Graph Representation Learning with PyTorch Geometric. *International Conference on Learning Representations*, 2019.
27. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; others. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 2019.
28. Dwivedi, V.P.; Joshi, C.K.; Laurent, T.; Bengio, Y.; Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982* **2020**.
29. Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; Leskovec, J. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *Advances in Neural Information Processing Systems* **2020**.
30. Bhagat, S.; Cormode, G.; Muthukrishnan, S. Node classification in social networks. In *Social Network Data Analytics*; Springer, 2011.
31. Ehrlinger, L.; Wöß, W. Towards a Definition of Knowledge Graphs. *SEMANTiCS* **2016**.
32. Clauset, A.; Newman, M.E.; Moore, C. Finding community structure in very large networks. *Physical review E* **2004**.
33. Largeron, C.; Mougél, P.N.; Benyahia, O.; Zaïane, O.R. DANCer: dynamic attributed networks with community structure generation. *Knowledge and Information Systems* **2017**.
34. McPherson, M.; Smith-Lovin, L.; Cook, J.M. Birds of a feather: Homophily in social networks. *Annual Review of Sociology* **2001**.
35. Leskovec, J.; Backstrom, L.; Kumar, R.; Tomkins, A. Microscopic evolution of social networks. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.
36. Wang, K.; Shen, Z.; Huang, C.; Wu, C.H.; Dong, Y.; Kanakia, A. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies* **2020**.
37. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations* **2015**.
38. Kullback, S.; Leibler, R.A. On information and sufficiency. *Annals of Mathematical Statistics* **1951**.
39. Massey Jr, F.J. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association* **1951**.
40. Koutra, D.; Parikh, A.; Ramdas, A.; Xiang, J. Algorithms for graph similarity and subgraph matching. 2011.
41. Abu-Aisheh, Z.; Raveaux, R.; Ramel, J.Y.; Martineau, P. An exact graph edit distance algorithm for solving pattern recognition problems. *International Conference on Pattern Recognition Applications and Methods*, 2015.
42. Martins, I.F.; Teixeira, A.L.; Pinheiro, L.; Falcao, A.O. A Bayesian approach to in silico blood-brain barrier penetration modeling. *Journal of Chemical Information and Modeling* **2012**.

-
43. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *International Conference on Learning Representations* **2017**.
 44. Morris, C.; Kriege, N.M.; Kersting, K.; Mutzel, P. Faster kernels for graphs with continuous attributes via hashing. *International Conference on Data Mining*, 2016.
 45. Guyon, I. Design of experiments of the NIPS 2003 variable selection benchmark. *NIPS Workshop on Feature Extraction and Feature Selection*, 2003.
 46. Barabási, A.L.; Albert, R. Emergence of scaling in random networks. *Science* **1999**.