Deep Graph Infomax

Petar Veličković[1,2], **William Fedus**[2,3], **William L. Hamilton**[2,4],
Pietro Liò[1], **Yoshua Bengio**[2] **and R Devon Hjelm**[5,2]
[1]University of Cambridge [2]MILA [3]Google Brain [4]Stanford University [5]MSR

Abstract

We present *Deep Graph Infomax* (DGI), a general approach for learning node representations within graph-structured data in an unsupervised manner. DGI relies on maximizing mutual information between patch representations and corresponding high-level summaries of graphs—both derived using established graph convolutional network architectures. DGI does not rely on random walks, and is readily applicable to both transductive and inductive learning setups. We demonstrate competitive performance on a variety of node classification benchmarks, which at times even exceeds the performance of supervised learning.

1 Introduction

Most successful methods for graph-structured data use supervised learning [2, 9, 1, 15, 5, 27]. However, this is often not possible as most graph data in the wild is unlabeled. In addition, it is often desirable to discover novel or interesting structure from large-scale graphs, and as such, unsupervised graph learning is essential for many important tasks.

Currently, unsupervised representation learning with graphs often relies on random walks [7, 20, 26, 8]. While powerful, this approach suffers from known limitations [21, 7, 20]. In this work, we phrase unsupervised graph learning as one of maximizing *mutual information* (MI). We rely on the Deep InfoMax approach (DIM) [12], which trains an encoder to maximize the MI between a high-level "global" representation and "local" parts of the input. DIM was developed on image data and is implemented with convnets, and here we adapt DIM to the graph domain. Our method, which we call *Deep Graph Infomax* (DGI) is consistently competitive on both transductive and inductive classification tasks, often outperforming both supervised and unsupervised strong baselines in our experiments.

2 DGI Setting and Methodology

Assume that we are provided with a set of *node features*, $\mathbf{X} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N\}$, where N is the number of nodes in the graph and $\vec{x}_i \in \mathbb{R}^F$ represents the features of node i. We are also provided with an *adjacency matrix*, $\mathbf{A} \in \mathbb{R}^{N \times N}$.

Our objective is to learn an *encoder*, $\mathcal{E} : \mathbb{R}^{N \times F} \times \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{N \times F'}$, such that $\mathcal{E}(\mathbf{X}, \mathbf{A}) = \mathbf{H} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}$ represents high-level representations $\vec{h}_i \in \mathbb{R}^{F'}$ for each node i. These representations may then be retrieved and used for downstream tasks, such as node classification. Here we will focus on *graph convolutional* encoders, and we will often refer to $\vec{h}_i$ as *patch representations* centered around a node i.

Our approach relies on *maximizing local mutual information*—that is, we seek to obtain node representations that capture the global information content of the entire graph, represented by a *summary vector*, $\vec{s}$. In order to obtain the graph-level summary vectors, $\vec{s}$, we leverage a *readout*

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

function, $\mathcal{R} : \mathbb{R}^{N \times F} \rightarrow \mathbb{R}^F$, and use it to summarize the obtained patch representations into a *graph-level representation*; i.e., $\vec{s} = \mathcal{R}(\mathcal{E}(\mathbf{X}, \mathbf{A}))$.

As a proxy for maximizing the local MI, we employ a *discriminator*, $\mathcal{D} : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}$, such that $\mathcal{D}(\vec{h}_i, \vec{s})$ represents the probability scores assigned to this patch-summary pair (should be higher for patches contained within the summary). As the objective for $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{D}$, we follow the lead of [12] and use the standard binary cross-entropy loss between the joint and product of marginals:

$$\mathcal{L} = \frac{1}{N+M}\left(\sum_{i=1}^{N} \mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\log \mathcal{D}\left(\vec{h}_i, \vec{s}\right)\right] + \sum_{j=1}^{M} \mathbb{E}_{(\widetilde{\mathbf{X}},\widetilde{\mathbf{A}})}\left[\log\left(1 - \mathcal{D}\left(\vec{\widetilde{h}}_j, \vec{s}\right)\right)\right]\right) \tag{1}$$

Negative samples for $\mathcal{D}$ (samples from the product of marginals) are drawn by pairing the summary $\vec{s}$ from $(\mathbf{X}, \mathbf{A})$ with patch representations $\vec{\widetilde{h}}_j$ of an alternative graph, $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}})$. In a multi-graph setting, such graphs may be obtained as other elements of a training set. However, for a single graph, an explicit (stochastic) *corruption function*, $\mathcal{C} : \mathbb{R}^{N \times F} \times \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{M \times F} \times \mathbb{R}^{M \times M}$ is required to obtain a negative example from the original graph, i.e. $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}}) = \mathcal{C}(\mathbf{X}, \mathbf{A})$.

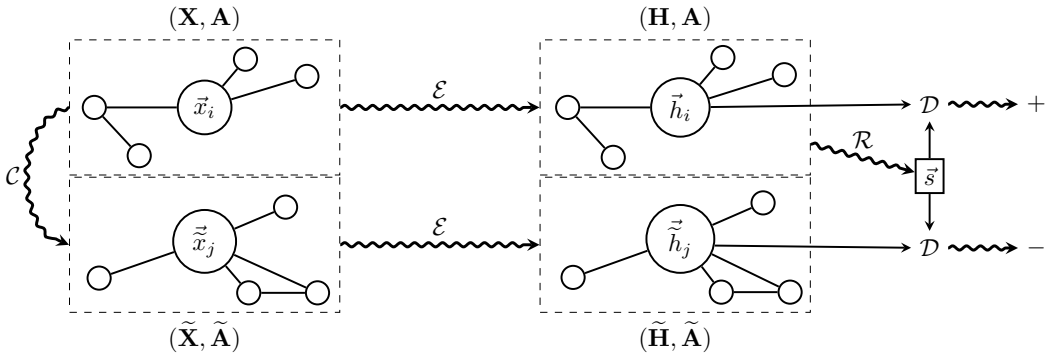

Figure 1: A high-level overview of Deep Graph Infomax.

Assuming the single-graph setup (i.e., $(\mathbf{X}, \mathbf{A})$ provided as input), the DGI procedure is (see Figure 1): 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: $\mathbf{H} = \mathcal{E}(\mathbf{X}, \mathbf{A}) = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}$. 3. Obtain *negative* patch representations, $\vec{\widetilde{h}}_j$: $\widetilde{\mathbf{H}} = \mathcal{E}(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}}) = \{\vec{\widetilde{h}}_1, \vec{\widetilde{h}}_2, \ldots, \vec{\widetilde{h}}_M\}$. 4. Summarize the input graph: $\vec{s} = \mathcal{R}(\mathbf{H})$. 5. Update parameters of $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{D}$ by applying gradient descent to maximize Equation 1.

3 Experimental setup

We follow the experimental setup described in [15] and [8] on the following benchmark tasks: (1) classifying research papers into topics on the Cora, Citeseer and Pubmed citation networks [23]; (2) predicting the community structure of a social network modeled with Reddit posts; and (3) classifying protein roles within protein-protein interaction (PPI) networks [32], requiring generalisation to unseen networks. The quality of the learnt embeddings is evaluated using a logistic regression classifier.

Transductive learning. For the transductive learning tasks (Cora, Citeseer and Pubmed), our encoder is a one-layer Graph Convolutional Network (GCN) model [15], with the following propagation rule:

$$\mathcal{E}(\mathbf{X}, \mathbf{A}) = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{\Theta}\right) \tag{2}$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with inserted self-loops and $\hat{\mathbf{D}}$ is its corresponding degree matrix; i.e. $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$. For the nonlinearity, σ, we have applied the parametric ReLU (PReLU) function [10], and $\mathbf{\Theta} \in \mathbb{R}^{F \times F'}$ is a learnable linear transformation applied to every node, with $F' = 512$ features being computed (specially, $F' = 256$ on Pubmed due to memory limitations).

The corruption function used in this setting is designed to encourage the representations to properly encode structural similarities of different nodes in the graph; for this purpose, $\mathcal{C}$ preserves the original

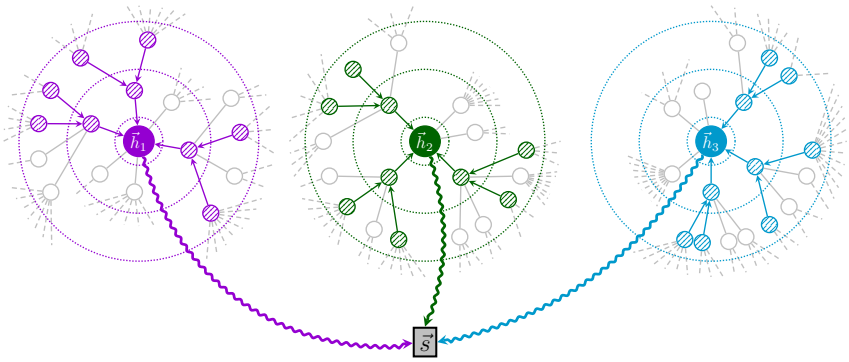

Figure 2: The DGI setup on large graphs (such as Reddit).

adjacency matrix ($\widetilde{\mathbf{A}} = \mathbf{A}$), whereas the corrupted features, $\widetilde{\mathbf{X}}$, are obtained by row-wise shuffling of $\mathbf{X}$. That is, the corrupted graph consists of exactly the same nodes as the original graph, but they are located in different places in the graph, and will therefore receive different patch representations. We demonstrate DGI is stable to other choices of corruption functions in Appendix C.

Inductive learning on large graphs. For inductive learning, we may no longer use the GCN update rule in our encoder (as the learned filters rely on a fixed and known adjacency matrix); instead, we apply the *mean-pooling* propagation rule, as used by GraphSAGE-GCN [8]:

$$\text{MP}(\mathbf{X}, \mathbf{A}) = \hat{\mathbf{D}}^{-1}\hat{\mathbf{A}}\mathbf{X}\mathbf{\Theta} \tag{3}$$

For Reddit, our encoder is a three-layer mean-pooling model with skip connections [11]:

$$\widetilde{\text{MP}}(\mathbf{X}, \mathbf{A}) = \sigma\left(\mathbf{X}\mathbf{\Theta}' \| \text{MP}(\mathbf{X}, \mathbf{A})\right) \qquad \mathcal{E}(\mathbf{X}, \mathbf{A}) = \widetilde{\text{MP}}_3(\widetilde{\text{MP}}_2(\widetilde{\text{MP}}_1(\mathbf{X}, \mathbf{A}), \mathbf{A}), \mathbf{A}) \tag{4}$$

where $\|$ is featurewise concatenation (i.e. the central node and its neighborhood are handled separately). We compute $F' = 512$ features in each MP layer, with the PReLU activation for σ. Given the large scale of the dataset, it will not fit into GPU memory entirely. Therefore, we use the subsampling approach of [8], where a minibatch of nodes is first selected, and then a subgraph centered around each of them is obtained by *sampling node neighborhoods with replacement*. Specifically, we sample 10, 10 and 25 neighbors at the first, second and third level, respectively—thus, each subsampled patch has 1 + 10 + 100 + 2500 = 2611 nodes. Only the computations necessary for deriving the central node i's patch representation, $\vec{h}_i$, are performed. These representations are then used to derive the summary vector, $\vec{s}$, for the minibatch (Figure 2). We used minibatches of 256 nodes throughout training. Analogously to the previous setup, our corruption function row-wise shuffles the feature matrices of all subsampled patches.

Inductive learning on multiple graphs. For the PPI dataset, inspired by previous successful supervised architectures [27], our encoder is a three-layer mean-pooling model with dense skip connections [11, 13]:

$$\begin{aligned} \mathbf{H}_1 = \sigma\left(\text{MP}_1(\mathbf{X}, \mathbf{A})\right); \quad \mathbf{H}_2 = \sigma\left(\text{MP}_2(\mathbf{H}_1 + \mathbf{X}\mathbf{W}_{\text{skip}}, \mathbf{A})\right); \\ \mathcal{E}(\mathbf{X}, \mathbf{A}) = \sigma\left(\text{MP}_3(\mathbf{H}_2 + \mathbf{H}_1 + \mathbf{X}\mathbf{W}_{\text{skip}}, \mathbf{A})\right) \end{aligned} \tag{5}$$

where $\mathbf{W}_{\text{skip}}$ is a learnable projection matrix, and MP is as defined in Equation 3. We compute $F' = 512$ features in each MP layer, using the PReLU activation for σ.

Here we use *randomly sampled training graphs* as negative examples. To further expand the pool of negative examples, we also apply dropout [24] to the input features of the sampled graph. We found it beneficial to standardize the learnt embeddings across the training set prior to providing them to the logistic regression model.

Readout, discriminator, and additional training details. Across all three experimental settings, we employed identical readout functions and discriminator architectures. For the readout function, we use a simple averaging of all the nodes' features, followed by a logistic sigmoid nonlinearity. The discriminator scores summary-patch representation pairs by applying a simple bilinear scoring function (similar to the scoring used by [17]): $\mathcal{D}(\vec{h}_i, \vec{s}) = \sigma\left(\vec{h}_i^T \mathbf{W} \vec{s}\right)$, where $\mathbf{W}$ is a learnable scoring matrix and σ is the logistic sigmoid nonlinearity.

Figure 3: t-SNE embeddings of the nodes in the Cora dataset from the raw features (**left**), features from a randomly initialized DGI model (**middle**), and a learned DGI model (**right**).

All models are initialized using Glorot initialization [6] and trained to maximize the MI (Equation 1) using Adam [14] with an initial learning rate of 0.001 (specially, 10^{-5} on Reddit). On the transductive datasets, we use an early stopping strategy on the observed *training* loss, with a patience of 20 epochs. On the inductive datasets we train for a fixed number of epochs (150 on Reddit, 20 on PPI).

Table 1: Summary of results in terms of classification accuracies (on transductive tasks) or micro-averaged F_1 scores (on inductive tasks). In the first column, we highlight the kind of data available to each method during training ($\mathbf{X}$: features, $\mathbf{A}$: adjacency matrix, $\mathbf{Y}$: labels).

		Transductive					*Inductive*	
Input	**Method**	**Cora**	**Citeseer**	**Pubmed**	**Input**	**Method**	**Reddit**	**PPI**
$\mathbf{X}$	Raw features	47.9 ± 0.4%	49.3 ± 0.2%	69.1 ± 0.3%	$\mathbf{X}$	Raw features	0.585	0.422
$\mathbf{A}$, $\mathbf{Y}$	LP [31]	68.0%	45.3%	63.0%	$\mathbf{A}$	DeepWalk [20]	0.324	—
$\mathbf{A}$	DeepWalk [20]	67.2%	43.2%	65.3%	$\mathbf{X}$, $\mathbf{A}$	DeepWalk + features	0.691	—
$\mathbf{X}$, $\mathbf{A}$	DeepWalk + features	70.7 ± 0.6%	51.4 ± 0.5%	74.3 ± 0.9%	$\mathbf{X}$, $\mathbf{A}$	GraphSAGE (best) [8]	0.908	0.502
$\mathbf{X}$, $\mathbf{A}$	Random-Init (ours)	69.3 ± 1.4%	61.9 ± 1.6%	69.6 ± 1.9%	$\mathbf{X}$, $\mathbf{A}$	Random-Init (ours)	0.933 ± 0.001	0.626 ± 0.002
$\mathbf{X}$, $\mathbf{A}$	**DGI** (ours)	**82.3** ± 0.6%	**71.8** ± 0.7%	**76.8** ± 0.6%	$\mathbf{X}$, $\mathbf{A}$	**DGI** (ours)	**0.940** ± 0.001	**0.638** ± 0.002
$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	GCN [15]	81.5%	70.3%	79.0%	$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	FastGCN [3]	0.937	—
$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	Planetoid [29]	75.7%	64.7%	77.2%	$\mathbf{X}$, $\mathbf{A}$, $\mathbf{Y}$	Avg. pooling [30]	0.958 ± 0.001	0.969 ± 0.002

4 Results

The results of our comparative evaluation experiments are summarized in Table 1.

For the transductive tasks, we report the mean classification accuracy (with standard deviation) on the test nodes of our method after 50 runs of training (followed by logistic regression), and reuse the metrics already reported in [15] for the performance of DeepWalk, GCN, Label Propagation (LP) [31] and Planetoid [29]. For the inductive tasks, we report the micro-averaged F_1 score on the test nodes, averaged after 50 runs of training, and reuse the metrics already reported in [8] for the other techniques. We compare against the unsupervised GraphSAGE approaches. We also provide supervised results for two related architectures—FastGCN [3] and Avg. pooling [30].

Our results demonstrate strong performance being achieved across all five datasets. We particularly note that the DGI approach is competitive *with the fully supervised setting*, even exceeding its performance on the Cora and Citeseer datasets. We assume that these benefits stem from the fact that, indirectly, the DGI approach allows for every node to have access to structural properties of the entire graph, whereas the supervised GCN is limited to only two-layer neighborhoods (by the extreme sparsity of the training signal and the corresponding threat of overfitting). Notably, on PPI the gap to supervised learning is still large—we believe this can be attributed to the extreme sparsity of available node features (over 40% of the nodes having all-zero features), that our encoder heavily relies on. We further demonstrate that our model is capable of improving on a randomly initialised encoder (*Random-Init*) which often represents a strong baseline, owing to its links to the Weisfeiler-Lehman graph isomorphism test [28].

A standard set of "evolving" t-SNE plots [16] of the embeddings on the Cora dataset is given in Figure 3. The projection obtains a Silhouette score [22] of 0.234, which compares favorably with the previous reported score of 0.158 for Embedding Propagation [4].

We leverage insights from further analyses to retain competitive performance to the supervised GCN even after *half* the dimensions are removed from the patch representations provided by the encoder. These—and several other—qualitative and ablation studies can be found in Appendix B.

Acknowledgments

We would like to thank the developers of PyTorch [18]. PV and PL have received funding from the European Union's Horizon 2020 research and innovation programme PROPAG-AGEING under grant agreement No 634821. We specially thank Hugo Larochelle and Jian Tang for the extremely useful discussions, and Andreea Deac, Arantxa Casanova, Ben Poole, Graham Taylor, Guillem Cucurull, Justin Gilmer, Nithium Thain and Zhaocheng Zhu for reviewing the paper prior to submission.

References

[1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[2] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[3] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[4] Alberto Garcia Duran and Mathias Niepert. Learning graph representations with embedding propagation. In *Advances in Neural Information Processing Systems*, pages 5119–5130, 2017.

[5] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[9] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[13] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[16] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[17] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[18] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[19] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[21] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394. ACM, 2017.

[22] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[23] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[24] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[25] Aravind Subramanian, Pablo Tamayo, Vamsi K Mootha, Sayan Mukherjee, Benjamin L Ebert, Michael A Gillette, Amanda Paulovich, Scott L Pomeroy, Todd R Golub, Eric S Lander, et al. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, 2005.

[26] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[28] Boris Weisfeiler and AA Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[29] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[30] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294*, 2018.

[31] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003.

[32] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.

A Further dataset details

Transductive learning. We utilize three standard citation network benchmark datasets—Cora, Citeseer and Pubmed [23]—and closely follow the transductive experimental setup of [29]. In all of these datasets, nodes correspond to documents and edges to (undirected) citations. Node features correspond to elements of a bag-of-words representation of a document. Each node has a class label. We allow for only 20 nodes per class to be used for training—however, honouring the transductive setup, the unsupervised learning algorithm has access to all of the nodes' feature vectors. The predictive power of the learned representations is evaluated on 1000 test nodes.

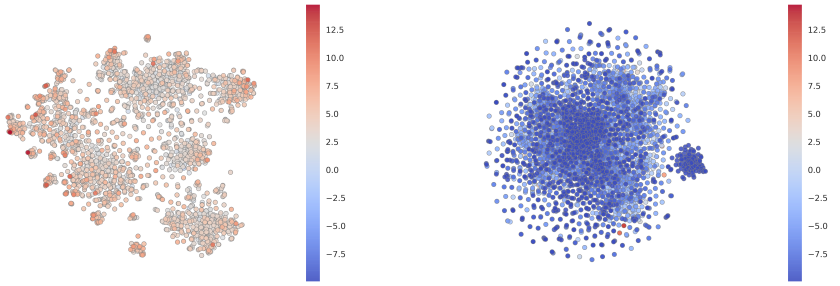

Figure 4: Discriminator scores, $\mathcal{D}\left(\vec{h}_i, \vec{s}\right)$, attributed to each node in the Cora dataset shown over a t-SNE of the DGI algorithm. Shown for both the original graph (**left**) and a negative sample (**right**).

Inductive learning on large graphs. We use a large graph dataset (231,443 nodes and 11,606,919 edges) of Reddit posts created during September 2014 (derived and preprocessed as in [8]). The objective is to predict the posts' community (*"subreddit"*), based on the GloVe embeddings of their content and comments [19], as well as metrics such as score or number of comments. Posts are linked together in the graph if the same user has commented on both. Reusing the inductive setup of [8], posts made in the first 20 days of the month are used for training, while the remaining posts are used for validation or testing and are *invisible* to the training algorithm.

Inductive learning on multiple graphs. We make use of a protein-protein interaction (PPI) dataset that consists of graphs corresponding to different human tissues [32]. The dataset contains 20 graphs for training, 2 for validation and 2 for testing. Critically, testing graphs remain *completely unobserved* during training. To construct the graphs, we used the preprocessed data provided by [8]. Each node has 50 features that are composed of positional gene sets, motif gene sets and immunological signatures. There are 121 labels for each node set from gene ontology, collected from the Molecular Signatures Database [25], and a node can possess several labels simultaneously.

B Further qualitative analysis

Visualizing discriminator scores. After obtaining the t-SNE visualizations, we turned our attention to the discriminator—and visualized the scores it attached to various nodes, for both the positive and a (randomly sampled) negative example (Figure 4). From here we can make an interesting observation—within the "clusters" of the learnt embeddings on the positive Cora graph, only a handful of "hot" nodes are selected to receive high discriminator scores. This suggests that there may be a clear distinction between embedding dimensions used for discrimination and classification, which we more thoroughly investigate in the next paragraph. In addition, we may observe that, as expected, the model is unable to find any strong structure within a negative example. Lastly, a few negative examples achieve high discriminator scores—a phenomenon caused by the existence of low-degree nodes in Cora (making the probability of a node ending up in an identical context it had in the positive graph non-negligible).

Impact and role of embedding dimensions. Guided by the previous result, we have visualized the embeddings for the top-scoring positive and negative examples (Figure 5). The analysis revealed existence of distinct dimensions in which both the positive and negative examples are *strongly biased*. We hypothesize that, given the random shuffling, the average *expected* activation of a negative example is zero, and therefore strong biases are required to "push" the example down in the discriminator. The positive examples may then use the remaining dimensions to both counteract this bias and encode patch similarity. To substantiate this claim, we order the 512 dimensions based on how distinguishable the positive and negative examples are in them (using p-values obtained from a t-test as a proxy). We then remove these dimensions from the embedding, respecting this order—either starting from the most distinguishable ($p \uparrow$) or least distinguishable dimensions ($p \downarrow$)—monitoring how this affects both classification and discriminator performance (Figure 6). The observed trends largely support our hypothesis: if we start by removing the biased dimensions first ($p \downarrow$), the classification performance

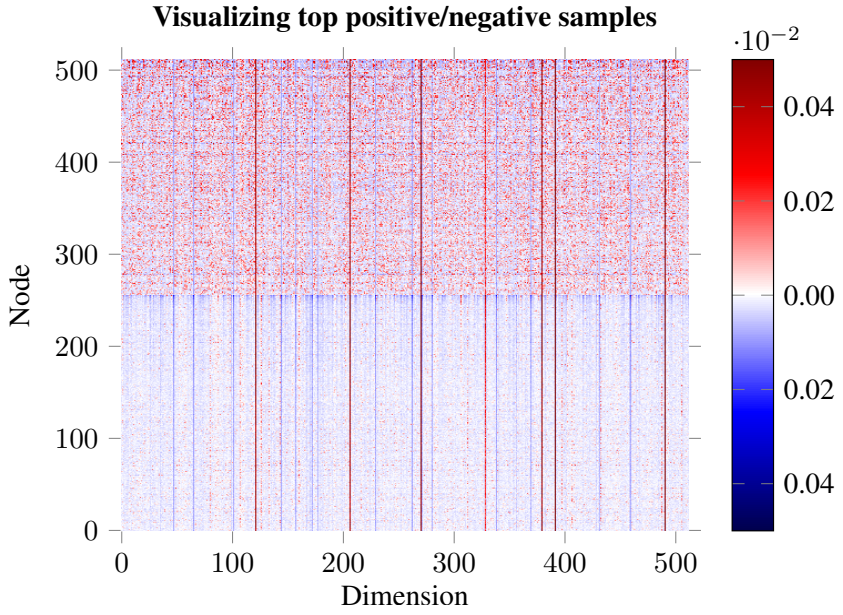

Figure 5: The learnt embeddings of the highest-scored positive examples (*upper half*), and the lowest-scored negative examples (*lower half*).

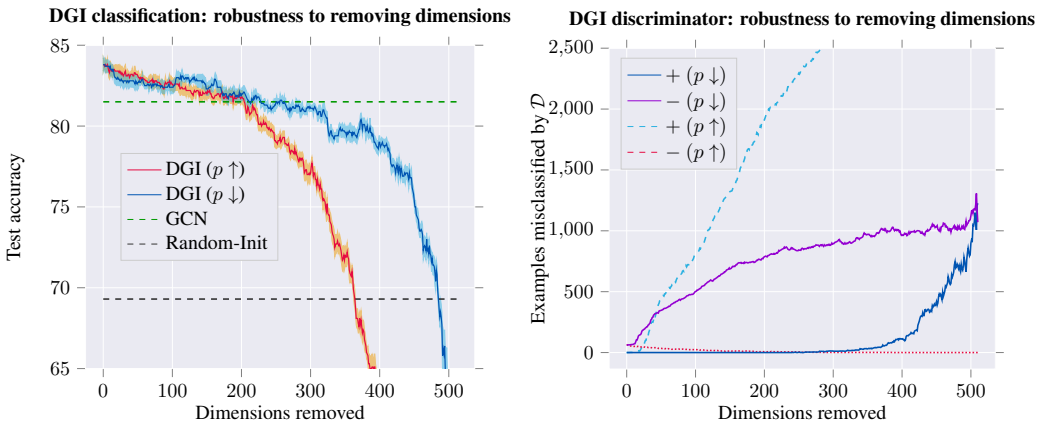

Figure 6: Classification performance (in terms of test accuracy of logistic regression; **left**) and discriminator performance (in terms of number of poorly discriminated positive/negative examples; **right**) on the learnt DGI embeddings, after removing a certain number of dimensions from the embedding—either starting with most distinguishing ($p \uparrow$) or least distinguishing ($p \downarrow$).

holds up for much longer (allowing us to remove over *half* of the embedding dimensions while remaining competitive to the supervised GCN), and the positive examples mostly remain correctly discriminated until well over half the dimensions are removed.

C Robustness to Choice of Corruption Function

Here, we consider alternatives to our corruption function, $\mathcal{C}$, used to produce negative graphs. We generally find that, for the node classification task, DGI is stable and robust to different strategies. However, for learning graph features towards other kinds of tasks, the design of appropriate corruption strategies remains an area of open research.

Our corruption function described in Section 3 preserves the original adjacency matrix ($\widetilde{\mathbf{A}} = \mathbf{A}$) but corrupts the features, $\widetilde{\mathbf{X}}$, via row-wise shuffling of $\mathbf{X}$. In this case, the negative graph is constrained to be isomorphic to the positive graph, which should not have to be mandatory. We can instead produce a negative graph by directly *corrupting* the adjacency matrix.

Therefore, we first consider an alternative corruption function $\mathcal{C}$ which preserves the features ($\widetilde{\mathbf{X}} = \mathbf{X}$) but instead adds or removes edges from the adjacency matrix ($\widetilde{\mathbf{A}} \neq \mathbf{A}$). This is done by sampling, i.i.d., a *switch* parameter $\mathbf{\Sigma}_{ij}$, which determines whether to corrupt the adjacency matrix at position (i, j). Assuming a given *corruption rate*, ρ, we may define $\mathcal{C}$ as performing the following operations:

$$\mathbf{\Sigma}_{ij} \sim \text{Bernoulli}(\rho) \tag{6}$$

$$\widetilde{\mathbf{A}} = \mathbf{A} \oplus \mathbf{\Sigma} \tag{7}$$

where $\oplus$ is the XOR (exclusive OR) operation.

This alternative strategy produces a negative graph with the same features, but different connectivity. Here, the corruption rate of $\rho = 0$ corresponds to an unchanged adjacency matrix (i.e. the positive and negative graphs are *identical* in this case). In this regime, learning is impossible for the discriminator, and the performance of DGI is in line with a randomly initialized DGI model. At higher rates of noise, however, DGI produces competitive embeddings.

We also consider *simultaneous* feature shuffling ($\widetilde{\mathbf{X}} \neq \mathbf{X}$) and adjacency matrix perturbation ($\widetilde{\mathbf{A}} \neq \mathbf{A}$), both as described before. We find that DGI still learns useful features under this compound corruption strategy—as expected, given that feature shuffling is already equivalent to an (isomorphic) adjacency matrix perturbation.

From both studies, we may observe that a certain lower bound on the positive graph perturbation rate is required to obtain competitive node embeddings for the classification task on Cora. Furthermore, the features learned for downstream node classification tasks are most powerful when the negative graph has similar levels of connectivity to the positive graph.

The classification performance peaks when the graph is perturbed to a reasonably high level, but remains *sparse*; i.e. the mixing between the separate 1-step patches is not substantial, and therefore the pool of negative examples is still *diverse* enough. Classification performance is impacted only marginally at higher rates of corruption—corresponding to *dense* negative graphs, and thus a less rich negative example pool—but still considerably outperforming the unsupervised baselines we have considered. This could be seen as further motivation for relying solely on feature shuffling, without adjacency perturbations—given that feature shuffling is a trivial way to guarantee a diverse set of negative examples, without incurring significant computational costs per epoch.

The results of this study are visualized in Figures 7 and 8.

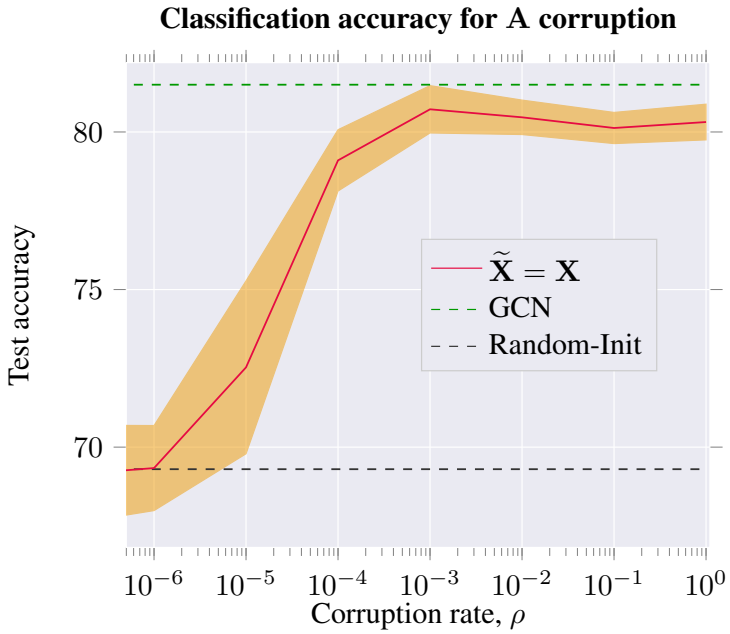

Figure 7: DGI also works under a corruption function that modifies only the adjacency matrix ($\widetilde{\mathbf{A}} \neq \mathbf{A}$) on the Cora dataset. The left range ($\rho \to 0$) corresponds to no modifications of the adjacency matrix—therein, performance approaches that of the randomly initialized DGI model. As ρ increases, DGI produces more useful features, but ultimately fails to outperform the feature-shuffling corruption function. **N.B.** log scale used for ρ.

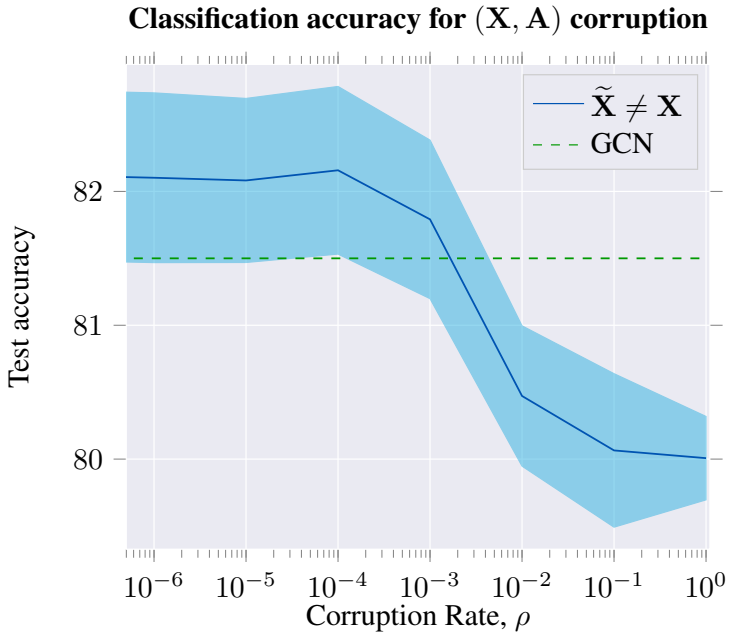

Figure 8: DGI is stable and robust under a corruption function that modifies *both* the feature matrix ($\mathbf{X} \neq \widetilde{\mathbf{X}}$) and the adjacency matrix ($\widetilde{\mathbf{A}} \neq \mathbf{A}$) on the Cora dataset. Corruption functions that preserve sparsity ($\rho \approx \frac{1}{N}$) perform the best. However, DGI still performs well even with large disruptions (where edges are added or removed with probabilities approaching 1). **N.B.** log scale used for ρ.