Deep Graph Infomax (ICLR 2019)

Petar Veličković et al.

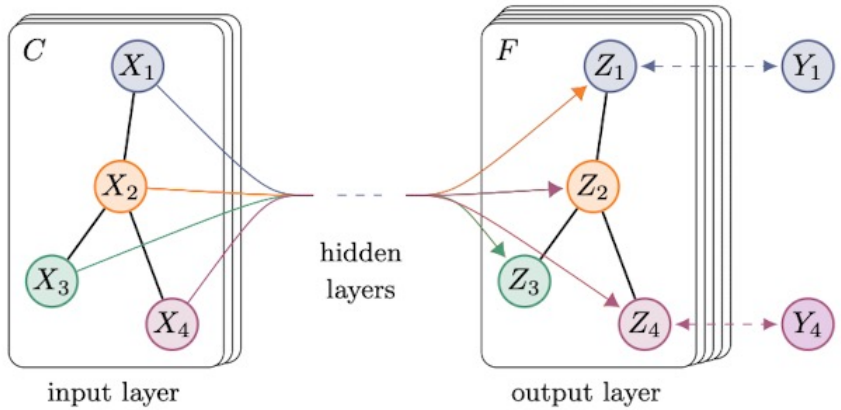
JongGeun Lee

Overview

1. Introduction
2. DGI Methodology
3. Classification Performance
4. Qualitative Analysis
5. Conclusion
6. Implementation

1. Introduction

*GCN



(a) Graph Convolutional Network

- ✓ A Novel approach but supervised learning
- ✓ Often not possible as most graph data in the wild is unlabeled

*Random Walk-based

- ✓ Methods: DeepWalk, Node2Vec, Metapath2vec
- ✓ Unsupervised but limitations

1. Over-emphasize proximity information at the expense of structural information
2. Performance is highly dependent on hyperparameter
3. Unclear whether the random walk objective provides a real useful signal compared to the GCN encoder.

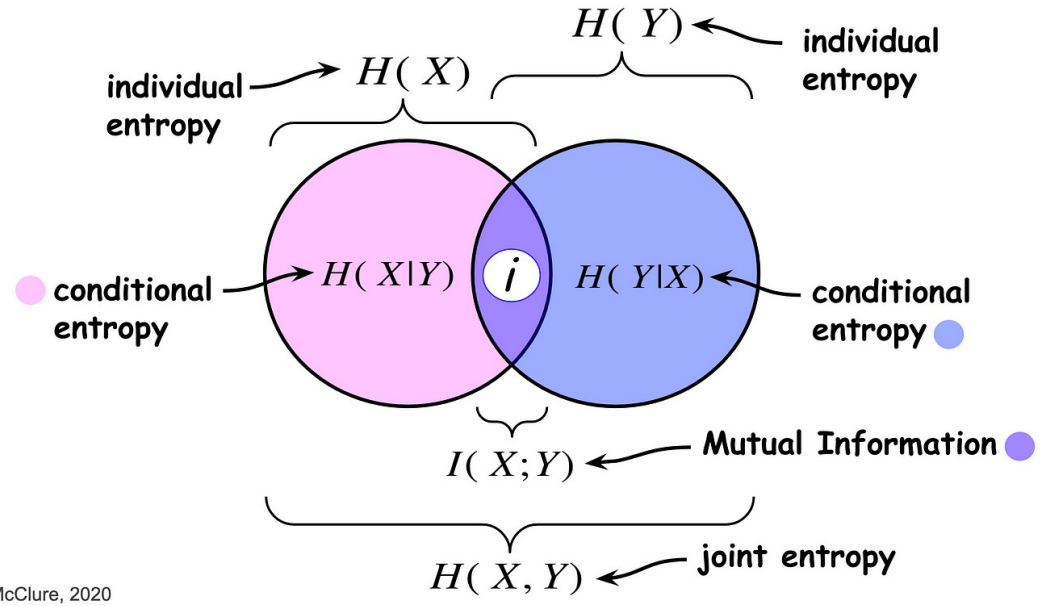
Alternative objective for unsupervised graph learning

: Mutual Information

1. Introduction – Information Theory

- Information
  - ✓ Information represents the degree of surprise
  - ✓ unusual event conveys more information.
  - ✓ Self information $h(x) = -log_2\, p(x)$
- Entropy
  - ✓ generalizing self information on a single random variable
  - ✓ the expected uncertainty of X $H(X) = -\sum_x p(x) \log p(x) = -\mathbb{E}[\, \log(p(x))\,]$
- Mutual Information
  - ✓ Entropy of paired random variables
  - ✓ Joint entropy $H(X,Y) = -\sum_{x,y} p(x,y) \log p(x,y).$
  - ✓ Conditional Entropy $H(X|Y) = -\sum_{x,y} p(x,y) \log p(x|y) = -\mathbb{E}[\, \log(p(x|y))\,]$
  - ✓ Mutual Information

$$\begin{aligned} I(X;Y) &= \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \\ &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \\ &= H(X) + H(Y) - H(X,Y). \end{aligned}$$



Sean McClure, 2020

- KL-Divergence
  - ✓ Distance between two probability distribution

$$D(p(x)||q(x)) = \sum_x p(x) \log \frac{p(x)}{q(x)}.$$

$$I(X;Y) = D(p(x,y)||p(x)p(y)).$$

1. Introduction – Mutual Information

Mutual Information Neural Estimation (MINE, Belghazi et al., 2018)

- ✓ Donsker-Varadhan Representation

$$\mathcal{I}(X;Y) := \mathcal{D}_{KL}(\mathbb{J}||\mathbb{M}) \geq \widehat{\mathcal{I}}_{\omega}^{(DV)}(X;Y) := \mathbb{E}_{\mathbb{J}}[T_{\omega}(x,y)] - \log \mathbb{E}_{\mathbb{M}}[e^{T_{\omega}(x,y)}],$$

- ✓ estimates mutual information by training a classifier to distinguish between samples coming from the joint and the product of marginals

Deep InfoMax (DIM, Hjelm et al., 2018)

- ✓ trains an encoder model to maximize the mutual information between a “global” representation and “local” parts of the input
- ✓ Representation learning of high-dimensional data
- ✓ Jensen-Shannon MI estimator

$$\widehat{\mathcal{I}}_{\omega,\psi}^{(\mathrm{JSD})}(X;E_{\psi}(X)) := \mathbb{E}_{\mathbb{P}}[-\mathrm{sp}(-T_{\psi,\omega}(x,E_{\psi}(x)))] - \mathbb{E}_{\mathbb{P}\times\tilde{\mathbb{P}}}[\mathrm{sp}(T_{\psi,\omega}(x',E_{\psi}(x)))],$$

Deep Graph Infomax

- ✓ DIM to the graph domain

2. DGI Methodology

*Graph-based unsupervised learning

- set of node features $X = \{ \vec{x}_1, \vec{x}_2, \dots , \vec{x}_N \}, \vec{x}_i \in \mathbb{R}^F$
- Adjacency matrix $A \in \mathbb{R}^{N \times N}$
- Encoder $\mathcal{E} : \mathbb{R}^{N \times F} \times \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{N \times F'}$
  - $\mathcal{E}(X, A) = H = \{ \vec{h}_1, \vec{h}_2, ..., \vec{h}_N \}, \vec{h}_i \in \mathbb{R}^{F'}$
  - $\vec{h}_i$: a patch representation which summarizes a patch of the graph centered around node i

*Local Global Mutual Information Maximization

- Readout Function $\mathcal{R} : \mathbb{R}^{N \times F} \rightarrow \mathbb{R}^F$
  - to summarize the obtained patch representations into a graph-level representation
- graph-level summary vector $\vec{s} = \mathcal{R}(\mathcal{E}(X, A))$
- Corruption function $\mathcal{C} : \mathbb{R}^{N \times F} \times \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{M \times F} \times \mathbb{R}^{M \times M}$
- Negative Samples: $(\tilde{X}, \tilde{A}) = \mathcal{C}(X, A)$
- Discriminator $\mathcal{D} : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}$
  - proxy for maximizing the local mutual information
  - should be higher score for $D(\vec{h}_i, \vec{s})$
  - should be lower score for $D(\tilde{h}_i, \vec{s})$

2. DGI Methodology

Jensen-Shannon MI estimator

$$\widehat{\mathcal{I}}^{(\mathrm{JSD})}_{\omega,\psi}(X; E_\psi(X)) := \mathbb{E}_{\mathbb{P}}[-\mathrm{sp}(-T_{\psi,\omega}(x, E_\psi(x)))] - \mathbb{E}_{\mathbb{P}\times\tilde{\mathbb{P}}}[\mathrm{sp}(T_{\psi,\omega}(x', E_\psi(x)))],$$

Noise Contrastive Objective with BCE loss

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

- ✓ Maximizes the mutual information between the local patches and graph level global representation

2. DGI Methodology – Theoretical Motivation

- ✓ The optimal classifier can distinguish whether X and S come from $\mathbb{J}$ *or* $\mathbb{M}$ means that the classifier is capable of detecting the presence of mutual information between the graph structure and its summary.

Lemma 1. *Let* $\{\mathbf{X}^{(k)}\}_{k=1}^{|\mathbf{X}|}$ *be a set of node representations drawn from an empirical probability distribution of graphs,* $p(\mathbf{X})$*, with finite number of elements,* $|\mathbf{X}|$*, such that* $p(\mathbf{X}^{(k)}) = p(\mathbf{X}^{(k')})\ \forall k, k'$*. Let* $\mathcal{R}(\cdot)$ *be a deterministic readout function on graphs and* $\vec{s}^{(k)} = \mathcal{R}(\mathbf{X}^{(k)})$ *be the summary vector of the* k*-th graph, with marginal distribution* $p(\vec{s})$*. The optimal classifier between the joint distribution* $p(\mathbf{X}, \vec{s})$ *and the product of marginals* $p(\mathbf{X})p(\vec{s})$*, assuming class balance, has an error rate upper bounded by* $\mathrm{Err}^* = \frac{1}{2}\sum_{k=1}^{|\mathbf{X}|} p(\vec{s}^{(k)})^2$*. This upper bound is achieved if* $\mathcal{R}$ *is injective.*

$$\frac{p(\mathbf{X}^{(k)})}{\sum_{\mathbf{X}' \in \mathcal{Q}^{(k)}} p(\mathbf{X}')} p(\vec{s}^{(k)})^2 = \rho^{(k)} p(\vec{s}^{(k)})^2 \quad \text{if } \mathcal{R} \text{ is injective, } \rho^{(k)} = 1$$

Corollary 1. *From now on, assume that the readout function used,* $\mathcal{R}$*, is injective. Assume the number of allowable states in the space of* $\vec{s}$*,* $|\vec{s}|$*, is greater than or equal to* $|\mathbf{X}|$*. Then, for* $\vec{s}^\star$*, the optimal summary under the classification error of an optimal classifier between the joint and the product of marginals, it holds that* $|\vec{s}^\star| = |\mathbf{X}|$*.*

- ✓ The optimal way to summarize the graphs is to have a summary vector space that is at least as large as the number of unique graphs

2. DGI Methodology – Theoretical Motivation

Theorem 1. $\vec{s}^{\star} = \text{argmax}_{\vec{s}} \text{MI}(\mathbf{X}; \vec{s})$, *where* MI *is mutual information.*

Proof. This follows from the fact that the mutual information is invariant under invertible transforms. As $|\vec{s}^{\star}| = |\mathbf{X}|$ and $\mathcal{R}$ is injective, it has an inverse function, $\mathcal{R}^{-1}$. It follows then that, for any $\vec{s}$, $\text{MI}(\mathbf{X}; \vec{s}) \leq H(\mathbf{X}) = \text{MI}(\mathbf{X}; \mathbf{X}) = \text{MI}(\mathbf{X}; \mathcal{R}(\mathbf{X})) = \text{MI}(\mathbf{X}; \vec{s}^{\star})$, where H is entropy. □

Theorem 2. *Let* $\mathbf{X}_i^{(k)} = \{\vec{x}_j\}_{j \in n(\mathbf{X}^{(k)}, i)}$ *be the neighborhood of the node* i *in the* k*-th graph that collectively maps to its high-level features,* $\vec{h}_i = \mathcal{E}(\mathbf{X}_i^{(k)})$*, where* n *is the neighborhood function that returns the set of neighborhood indices of node* i *for graph* $\mathbf{X}^{(k)}$*, and* $\mathcal{E}$ *is a deterministic encoder function. Let us assume that* $|\mathbf{X}_i| = |\mathbf{X}| = |\vec{s}| \geq |\vec{h}_i|$*. Then, the* $\vec{h}_i$ *that minimizes the classification error between* $p(\vec{h}_i, \vec{s})$ *and* $p(\vec{h}_i)p(\vec{s})$ *also maximizes* $\text{MI}(\mathbf{X}_i^{(k)}; \vec{h}_i)$.

✓ Theorem 2 provides justification for using the following Loss function

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

2. DGI Methodology – Overview of DGI

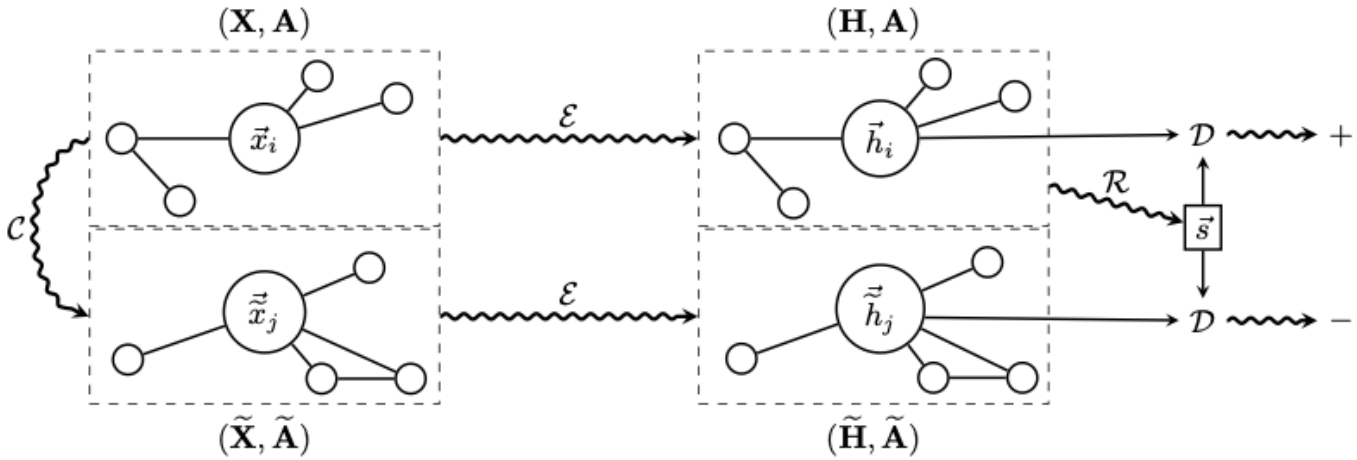


Figure 1: A high-level overview of Deep Graph Infomax. Refer to Section 3.4 for more details.

3.4 Overview of DGI

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, $\vec{\tilde{h}}_j$ for the negative example by passing it through the encoder: $\widetilde{\mathbf{H}} = \mathcal{E}(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}}) = \{\vec{\tilde{h}}_1, \vec{\tilde{h}}_2, \ldots, \vec{\tilde{h}}_M\}$.
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.

3. Classification Performance

Table 1: Summary of the datasets used in our experiments.

Dataset	Task	Nodes	Edges	Features	Classes	Train/Val/Test Nodes
Cora	Transductive	2,708	5,429	1,433	7	140/500/1,000
Citeseer	Transductive	3,327	4,732	3,703	6	120/500/1,000
Pubmed	Transductive	19,717	44,338	500	3	60/500/1,000
Reddit	Inductive	231,443	11,606,919	602	41	151,708/23,699/55,334
PPI	Inductive	56,944 (24 graphs)	818,716	50	121 (multilbl.)	44,906/6,514/5,524 (20/2/2 graphs)

*Transductive learning

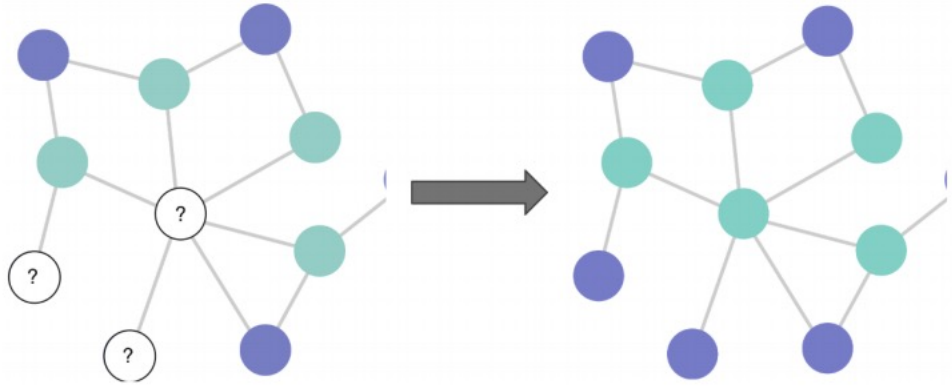
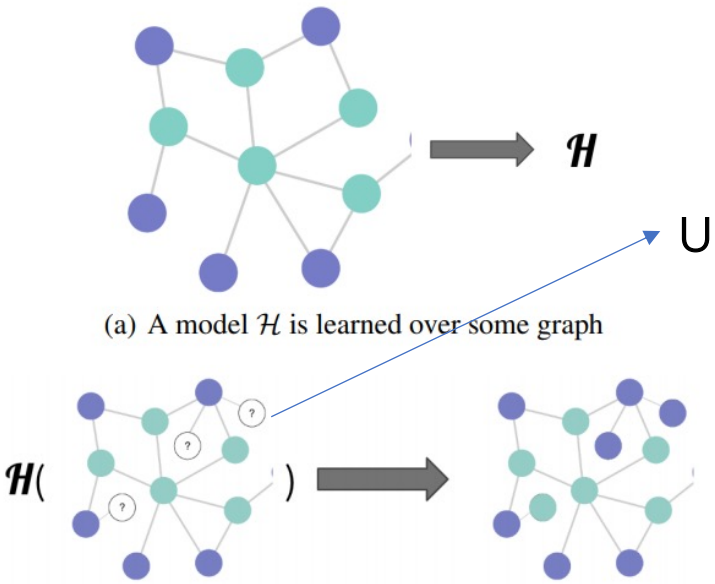


Figure 1. Node classification in transductive setting. At training time, the learning algorithm has access to all the nodes and edges including nodes for which labels are to be predicted.

*Inductive learning



3. Classification Performance

*Transductive learning

- Dataset: Cora, Citeseer, and Pubmed
- Task: classifying research papers into topics
- Encoder: One-layer GCN

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

- $\hat{A} = A + I_N$
- $\hat{D}$: Degree matrix
- σ : $PReLU$
- $\mathbf{\Theta} \in \mathbb{R}^{F \times F'}$: learnable linear transformation
- corrupted adjacency matrix $\tilde{A}$: A
- corrupted features $\tilde{X}$: row-wise shuffling of X

3. Classification Performance

*Inductive learning on large graphs

- Dataset: Reddit
- Task: predicting the community structure of a social network modeled with Reddit posts
- Encoder: three-layer mean-pooling model with skip connections
  - mean-pooling propagation rule in GraphSAGE-GCN

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

$$\widetilde{\mathbf{MP}}(\mathbf{X}, \mathbf{A}) = \sigma\left(\mathbf{X}\mathbf{\Theta}' \| \mathbf{MP}(\mathbf{X}, \mathbf{A})\right)$$

$$\mathcal{E}(\mathbf{X}, \mathbf{A}) = \widetilde{\mathbf{MP}}_3(\widetilde{\mathbf{MP}}_2(\widetilde{\mathbf{MP}}_1(\mathbf{X}, \mathbf{A}), \mathbf{A}), \mathbf{A})$$

- $||$: Featurewise concatenation
- σ : $PReLU$
- corruption function: similar approach as in the transductive task (row-wise shuffle)

3. Classification Performance

*Inductive learning on multiple graphs

- Dataset: PPI
- Task: classifying protein roles within protein-protein interaction network
- Encoder: three-layer mean-pooling model with dense skip connections

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

- W_{skip} :learnable projection matrix
- σ : $PReLU$
- for negative sample, randomly sample a different graph from the training set

3. Classification Performance

***Readout function**

$$\mathcal{R}(\mathbf{H}) = \sigma\left(\frac{1}{N}\sum_{i=1}^{N}\vec{h}_i\right)$$

- ✓ simple averaging of all the nodes' features + Sigmoid (for non-linearity)

***Discriminator**

$$\mathcal{D}(\vec{h}_i, \vec{s}) = \sigma\left(\vec{h}_i^T \mathbf{W} \vec{s}\right)$$

- ✓ Bilinear scoring function + Sigmoid (Converting score into probability)
- ✓ W: learnable scoring matrix

***Additional training details**

- ✓ Glorot initialization, Adam SGD (lr = 1e-3)
- ✓ Transductive dataset : early stopping with a patience of 20 epochs
- ✓ Inductive dataset: 150 epochs on Reddit & 20 epochs on PPI

3. Classification Performance

Accuracy in transductive

Transductive

Available data	Method	Cora	Citeseer	Pubmed
X	Raw features	47.9 ± 0.4%	49.3 ± 0.2%	69.1 ± 0.3%
A, **Y**	LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
A	DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
X, **A**	DeepWalk + features	70.7 ± 0.6%	51.4 ± 0.5%	74.3 ± 0.9%
X, **A**	Random-Init (ours)	69.3 ± 1.4%	61.9 ± 1.6%	69.6 ± 1.9%
X, **A**	**DGI** (ours)	**82.3** ± 0.6%	**71.8** ± 0.7%	**76.8** ± 0.6%
X, **A**, **Y**	GCN (Kipf & Welling, 2016a)	81.5%	70.3%	79.0%
X, **A**, **Y**	Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%

Micro-F1 in inductive

$$Precision_{mi} = \frac{\sum_i TP_i}{\sum_i [TP_i + FP_i]}$$

$$Recall_{mi} = \frac{\sum_i TP_i}{\sum_i [TP_i + FN_i]}$$

$$F1_{mi} = 2\frac{Precision_{mi} \times Recall_{mi}}{Precision_{mi} + Recall_{mi}}$$

Inductive

Available data	Method	Reddit	PPI
X	Raw features	0.585	0.422
A	DeepWalk (Perozzi et al., 2014)	0.324	—
X, **A**	DeepWalk + features	0.691	—
X, **A**	GraphSAGE-GCN (Hamilton et al., 2017a)	0.908	0.465
X, **A**	GraphSAGE-mean (Hamilton et al., 2017a)	0.897	0.486
X, **A**	GraphSAGE-LSTM (Hamilton et al., 2017a)	0.907	0.482
X, **A**	GraphSAGE-pool (Hamilton et al., 2017a)	0.892	0.502
X, **A**	Random-Init (ours)	0.933 ± 0.001	0.626 ± 0.002
X, **A**	**DGI** (ours)	**0.940** ± 0.001	**0.638** ± 0.002
X, **A**, **Y**	FastGCN (Chen et al., 2018)	0.937	—
X, **A**, **Y**	Avg. pooling (Zhang et al., 2018)	0.958 ± 0.001	0.969 ± 0.002

4. Qualitative Analysis

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**). The clusters of the learned DGI model's embeddings are clearly defined, with a Silhouette score of 0.234.

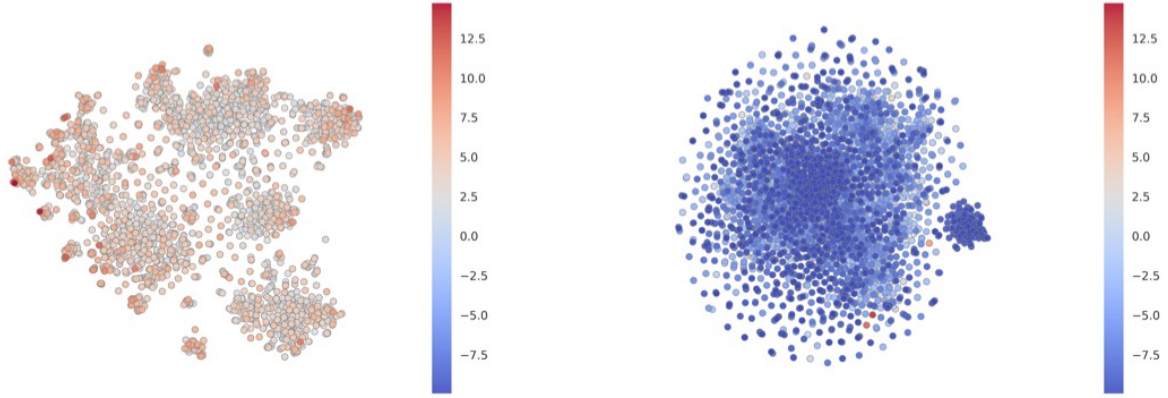


Figure 4: Discriminator scores, $\mathcal{D}\left(\vec{h}_i, \vec{s}\right)$, attributed to each node in the Cora dataset shown over a t-SNE of the DGI algorithm. Shown for both the original graph (**left**) and a negative sample (**right**).

5. Conclusion

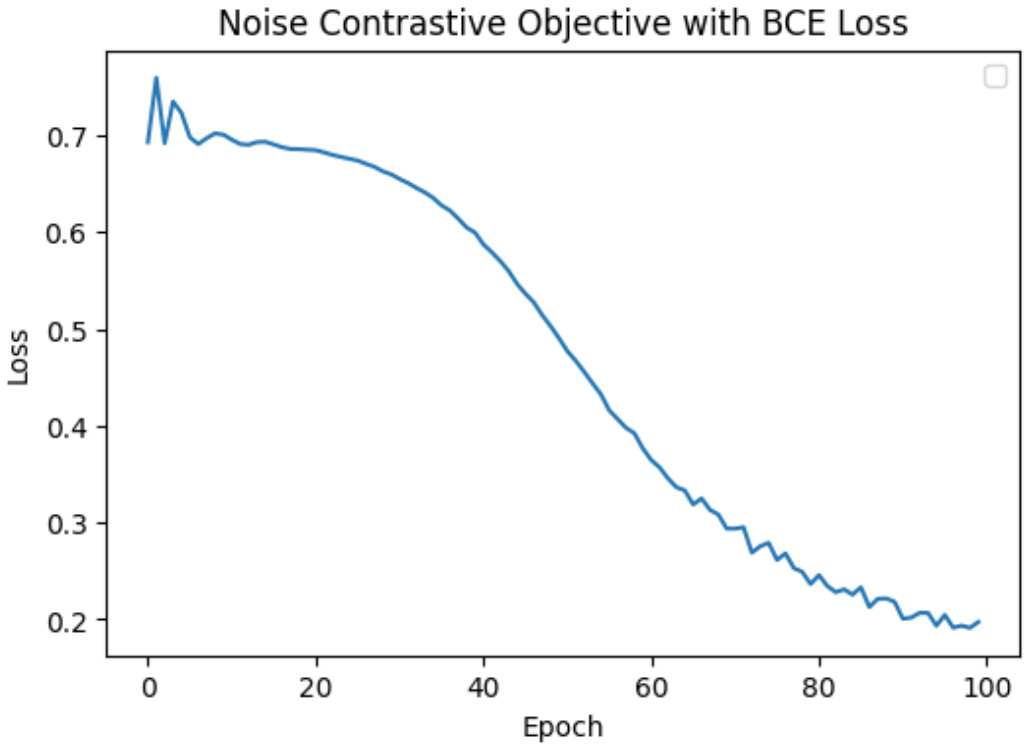
- ✓ New approach for learning unsupervised representation on graph-structured data
- ✓ Leveraging local mutual information maximization across the graph's patch representations
- ✓ Competitive performance across a variety of both transductive and inductive classification task
- ✓ Even outperforming relevant supervised architectures

6. Implementation – Model

```python
class Discriminator(nn.Module):
    def __init__(self, n_h):
        super(Discriminator, self).__init__()
        self.bilinear = nn.Bilinear(n_h, n_h, 1)

        for m in self.modules():
            self.weights_init(m)

    def weights_init(self, m):
        if isinstance(m, nn.Bilinear):
            torch.nn.init.xavier_uniform_(m.weight.data)


    def forward(self, s, pos_H, neg_H):
        summary = torch.unsqueeze(s, 1)
        summary = summary.expand_as(pos_H)

        pos_score = torch.squeeze(self.bilinear(pos_H, summary), 2)
        neg_score = torch.squeeze(self.bilinear(neg_H, summary), 2)

        logits = torch.cat((pos_score, neg_score), 1)

        return logits
class GCN(nn.Module):
    def __init__(self, in_ft, out_ft):
        super(GCN, self).__init__()
        self.fc = nn.Linear(in_ft, out_ft)
        self.prelu = nn.PReLU()

        for m in self.modules():
            self.weights_init(m)

    def weights_init(self, m):
        if isinstance(m, nn.Linear):
            torch.nn.init.xavier_uniform_(m.weight.data)

    def forward(self, X, adj):
        X_fts = self.fc(X)
        out = torch.unsqueeze(torch.spmm(adj, torch.squeeze(X_fts, 0)), 0)

        return self.prelu(out)
```

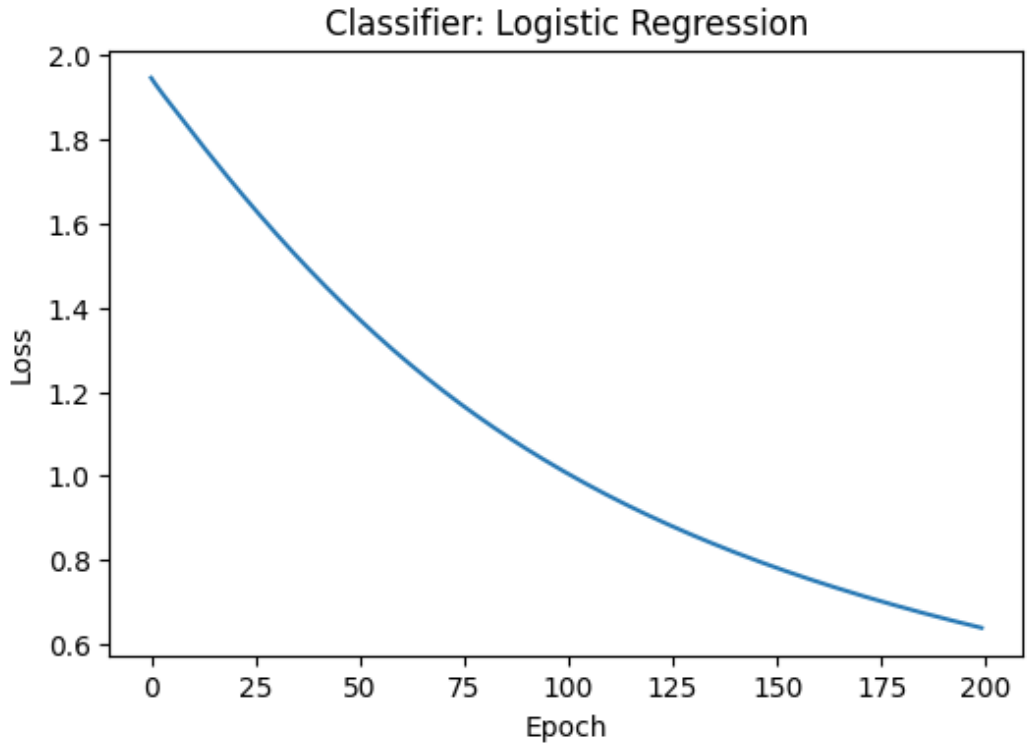
```python
class AvgReadout(nn.Module):
    def __init__(self):
        super(AvgReadout, self).__init__()

    def forward(self, pth_rep):
            '''pth_rep: patch_representation'''
            return torch.mean(pth_rep, 1)
class DGI(nn.Module):
    def __init__(self, n_in, n_h):
        super(DGI, self).__init__()
        self.gcn = GCN(n_in, n_h)
        self.read = AvgReadout()
        self.sigm = nn.Sigmoid()
        self.disc = Discriminator(n_h)

    def forward(self, pos, neg, adj):
        pos_H = self.gcn(pos, adj)
        s = self.read(pos_H)
        s = self.sigm(s)
        neg_H = self.gcn(neg, adj)
        logits = self.disc(s, pos_H, neg_H)

        return logits

    def embed(self, seq, adj):
        pos_H = self.gcn(seq, adj)
        s = self.read(pos_H)

        return pos_H.detach(), s.detach()
```

6. Implementation



Node Representation

Classifier: Logistic Regression



✓ Accuracy on Cora: 0.8120

Thank You