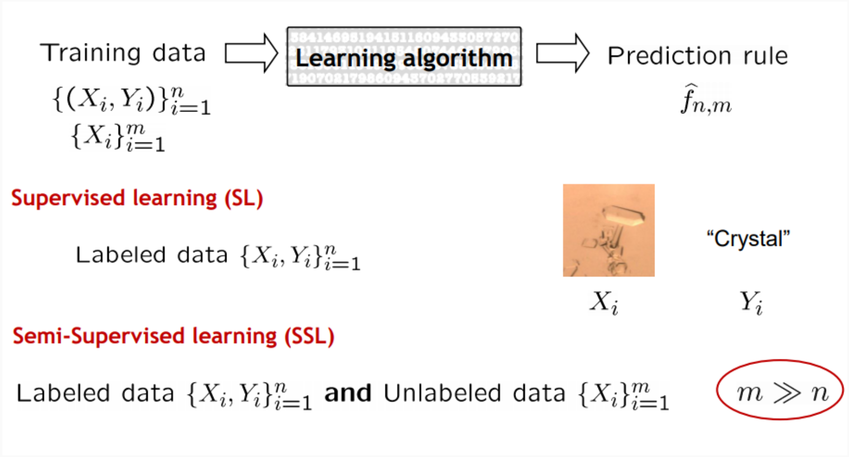
SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

Thomas N. Kipf, Max Welling

ICLR 2017

Presented by Devansh Shah

Semi-Supervised Learning



Training data $\{(X_i, Y_i)\}_{i=1}^n$ $\{X_i\}_{i=1}^m$ $\Rightarrow$ **Learning algorithm** $\Rightarrow$ Prediction rule $\widehat{f}_{n,m}$

Supervised learning (SL)

Labeled data $\{X_i, Y_i\}_{i=1}^n$

X_i "Crystal" Y_i

Semi-Supervised learning (SSL)

Labeled data $\{X_i, Y_i\}_{i=1}^n$ **and** Unlabeled data $\{X_i\}_{i=1}^m$

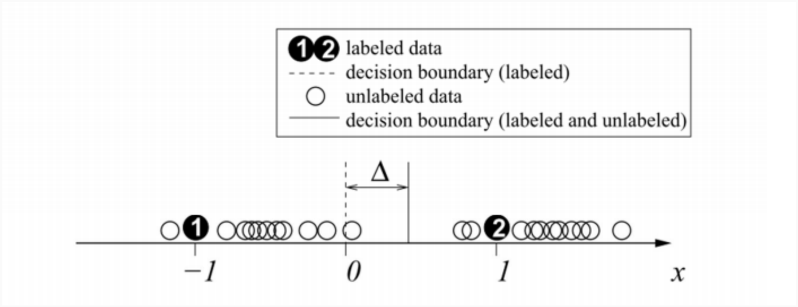
$m \gg n$

$$\boxed{\text{Goal: Learn a better prediction rule than based on labeled data alone}}$$

Why bother?

- Unlabeled data is cheap
- Labeled data can be hard to get
    - human annotation is boring
    - labels may require experts

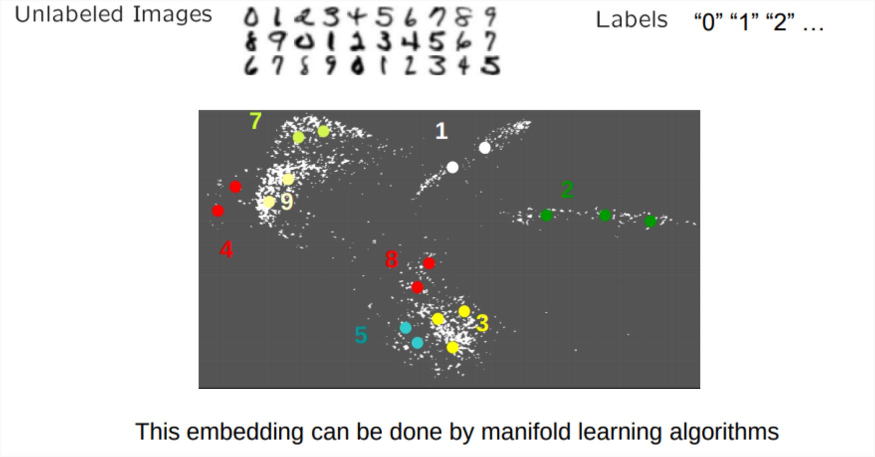
Can Unlabeled data help?



❶❷ labeled data
decision boundary (labeled)
○ unlabeled data
decision boundary (labeled and unlabeled)
Δ
−1
0
1
x

- Assuming each class is a coherent group (e.g. Gaussian)
- With and without unlabeled data: decision boundary shift

Can Unlabeled data help?



This embedding can be done by manifold learning algorithms

"Similar" data points have "similar" labels

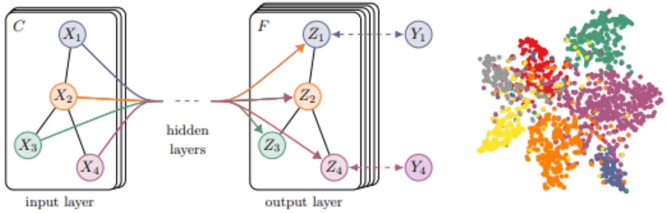
Semi-supervised vs transductive learning

- labeled data $(X_l, Y_l) = \{(x_{1:l}, y_{1:l})\}$
- unlabeled data $X_u = \{x_{l+1:n}\}$, available during training
- test data $X_{test} = \{x_{n+1:}\}$, not available during training

Inductive learning is ultimately applied to the test data.

Transductive learning is only concerned with the unlabeled data.

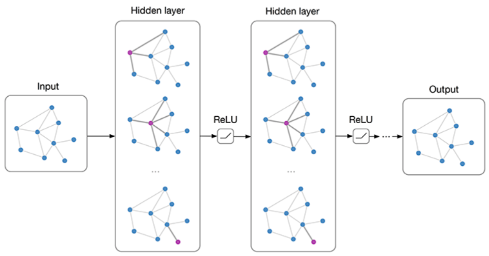
Graph Convolutional Networks



(a) Graph Convolutional Network (b) Hidden layer activations

Applications

- Social Networks
- Protein-Protein Interaction
- 3D Meshes
- Clustering
- Scene Graphs



Multi-layer Graph Convolutional Network (GCN) with first-order filters.

Graph Learning Problem

Inputs:

- graph $G = (V, E)$
- A feature description x_i for every node i; summarized in a $N \times D$ feature matrix X (N: number of nodes, D: number of input features)
- Adjacency matrix A

Outputs:

- node-level output Z (an N$\times$F feature matrix, where F is the number of output features per node)

Understanding Graph Neural Networks

Every neural network layer can be written as a non-linear function $H^{l+1} = f(H^l, A)$ with

- $H^0 = X$
- $H^L = Z$ where L is number of layers

Understanding Graph Neural Networks

$f(H^l, A) = \sigma(AH^l W^l)$ where

- W^l is weight matrix for the l-th layer
- $\sigma(.)$ is a non-linear activation function like the ReLU

Understanding Graph Neural Networks

Limitation I:

- Multiplication with A means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself

Fix:

- Enforce self-loop in the graph by adding identity matrix to A

Understanding Graph Neural Networks

Limitation II:

- A is typically not normalized and therefore the multiplication with A will completely change the scale of the feature vectors

Fix:

- Normalize A such that all rows sum to one, i.e. $D^{-1}A$, where D is the diagonal node degree matrix. Multiplying with $D^{-1}A$ now corresponds to taking the average of neighboring node features

Understanding Graph Neural Networks

Propagation Rule: $f(H^l, A) = \sigma(\hat{D}^{-0.5}\hat{A}\hat{D}^{-0.5}H^l W^l)$

- $\hat{A} = A + I$, where I is the identity matrix
- $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$

Semi-Supervised Node Classification

Cross-Entropy error over all labeled examples

$$Z = softmax(H^L)$$

$$Loss = -\sum_{l \in Y_L} \sum_{f=1}^{F} Y_{lf} \, ln Z_{lf}$$

- H_L is the output of the last layer
- Y_L is the set of node indices that have labels
- F is the number of distinct output classes

Experiments

Datasets

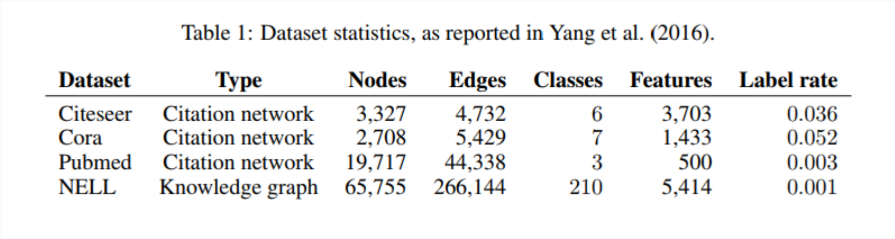


Table 1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Experiments

Baselines

- Label Propagation (LP)
- Semi-Supervised embedding (SemiEmb)
- Manifold regularization (ManiReg)
- skip-gram based graph embeddings (DeepWalk)
- Iterative classification algorithm (ICA)

Experiments

Results

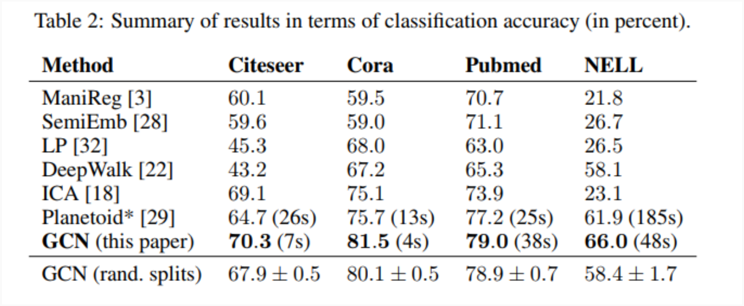


Table 2: Summary of results in terms of classification accuracy (in percent).

Method	**Citeseer**	**Cora**	**Pubmed**	**NELL**
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

Robust Graph Convolutional Networks Against Adversarial Attacks

Dingyuan Zhu, Ziwei Zhang, Peng Cui, Wenwu Zhu

ACM SIGKDD 2019

Presented by Devansh Shah

Adversarial Attacks on Graphs

RELATED WORK

- **Adversarial Attack on Graph Structured Data**
- **Adversarial Attacks on Neural Networks for Graph Data**

Graph adversarial attack

Transductive Node Classification Setting

- A single graph $G_0 = (V_0, E_0)$ is considered in the entire dataset
- A target node $c_i \in V_i$ of graph G_i is associated with a corresponding node label $y_i \in Y$
- Test nodes (but not their labels) are also observed during training
- $D^{(tra)} = \{(G_0, c_i, y_i)\}_{i=1}^{N}$

Graph adversarial attack

Problem Definition
Given:

- A learned classifier f
- An instance from the dataset $(G, c, y) \in D$

The graph adversarial attacker $g(\cdot, \cdot) : G \times D \to G$ modifies the graph $G = (V, E)$ into $\tilde{G} = (\tilde{V}, \tilde{E})$ such that,

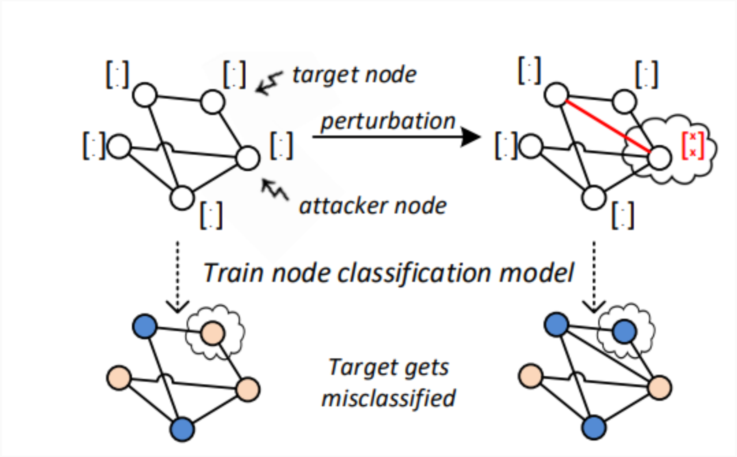
$$\max_{\tilde{G}} \mathbb{1}(f(\tilde{G}, c) \neq y)$$

$$s.t. \quad \tilde{G} = g(f, (G, c, y))$$

$$Eq(G, \tilde{G}, c) = 1$$

Here $Eq(\cdot, \cdot, \cdot) : G \times G \times V \to \{0, 1\}$ is an equivalency indicator that tells whether two graphs G and $\tilde{G}$ are semantically equivalent

Graph adversarial attack



Robust Graph Convolutional Network (RGCN)

Crux of the paper

- Instead of representing nodes as vectors, they are represented as Gaussian distributions in each convolutional layer
- When the graph is attacked, the model can automatically absorb the effects of adversarial changes in the variances of the Gaussian distributions
- To remedy the propagation of adversarial attacks in GCNs, variance-based attention mechanism is used when performing convolutions

Gaussian-based Graph Convolution Layer

Latent representation of node v_i in layer l

$$h_i^l = \mathcal{N}(\mu_i^l, diag(\sigma_i^l))$$

$$\mu_i^l \in \mathbb{R}^{f_l} \textit{ is the mean vector}$$

$$diag(\sigma_i^l)) \in \mathbb{R}^{f_l \times f_l} \textit{ is the diagonal variance matrix}$$

Notation:

$$M^l = [\mu_1^l, ..., \mu_1^N] \in \mathbb{R}^{N \times f_l} \textit{ is the mean matrix}$$

$$\mathrm{Cov}^l = [\sigma_1^l, ..., \sigma_1^N] \in \mathbb{R}^{N \times f_l} \textit{ is the variance matrix}$$

RGCN

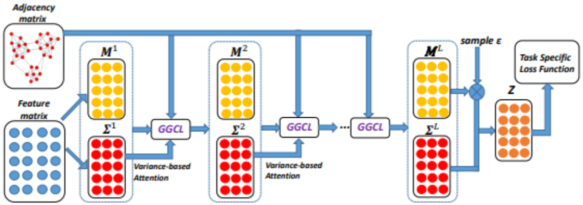


Figure 1: The framework of our proposed RGCN. GGCL represents Gaussian-based Graph Convolution Layer introduced in Section 4.2.

RGCN

Theorem
If $x_i \sim \mathcal{N}(\mu_i, diag(\sigma_i))$ $i = 1, ...n$ and they are independent, then for any fixed weights w_i, we have:

$$\sum_{i=1}^{n} w_i x_i \sim \mathcal{N}(\sum_{i=1}^{n} w_i \mu_i,\ diag(\sum_{i=1}^{n} w_i^2 \sigma_i))$$

RGCN Node Aggregation

To prevent the propagation of adversarial attacks in GCNs, we propose an attention mechanism to assign different weights to neighbors based on their variances since **larger variances indicate more uncertainties** in the latent representations and larger probability of having been attacked

$$\alpha_j^l = exp(-\gamma\sigma_j^l)$$

Here α_j^l are the attention weights of node v_j in the layer l and γ is a hyper-parameter

RGCN Node Aggregation

$$\mu_i^{l+1} = ReLU(\sum_{j \in ne(i)} \frac{1}{\sqrt{\tilde{D}_{i,i}\tilde{D}_{j,j}}}(\mu_j^l \odot \alpha_j^l)W_\mu^l)$$

$$\sigma_i^{l+1} = ReLU(\sum_{j \in ne(i)} \frac{1}{\tilde{D}_{i,i}\tilde{D}_{j,j}}(\sigma_j^l \odot \alpha_j^l \odot \alpha_j^l)W_\sigma^l)$$

Loss Functions

Considering that the hidden representations of our method are Gaussian distributions, we first adopt a sampling process in the last hidden layer

$$z_i \sim \mathcal{N}(\mu_i^L, diag(\sigma_i^L))$$

Next z_i is passed to a softmax function to get the predicted labels:

$$\tilde{Y} = softmax(Z), Z = [z1, ..., zn]$$

L_{cls} is the cross-entropy loss between the actual labels and the predicted probabilities for the labelled nodes

Loss Functions

To ensure that the learned representations are indeed Gaussian distributions, we use an explicit regularization to constrain the latent representations in the first layer as follows

$$L_{reg1} = \sum_{i=1}^{n} KL(\mathcal{N}(\mu_i, diag(\sigma_i))||\mathcal{N}(0, I))$$

where $KL(\cdot||\cdot)$ is the KL-divergence between two distributions

We also impose L_2 regularization on parameters of the first layer as follows:

$$L_{reg2} = \left\| W_\mu^{(0)} \right\|_2^2 + \left\| W_\sigma^{(0)} \right\|_2^2$$

Loss Functions

$$L = L_{cls} + \beta_1 L_{reg1} + \beta_2 L_{reg2}$$

where β_1 and β_2 are hyper-parameters that control the impact of different regularizations

Results

Node Classification on Clean Datasets

RGCN slightly outperforms the baseline methods on Pubmed, while having comparable performance on Cora and Citeseer

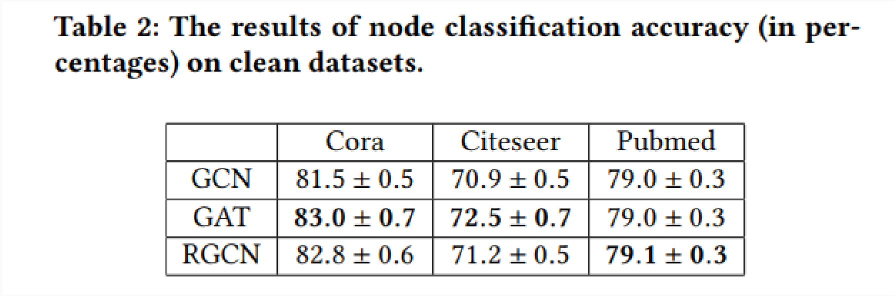


Table 2: The results of node classification accuracy (in percentages) on clean datasets.

	Cora	Citeseer	Pubmed
GCN	81.5 ± 0.5	70.9 ± 0.5	79.0 ± 0.3
GAT	$\mathbf{83.0 \pm 0.7}$	$\mathbf{72.5 \pm 0.7}$	79.0 ± 0.3
RGCN	82.8 ± 0.6	71.2 ± 0.5	$\mathbf{79.1 \pm 0.3}$

Results

Against Non-targeted Adversarial Attacks

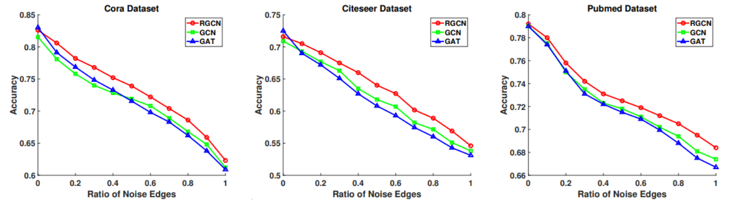


Figure 2: Results of different methods when adopting Random Attack as the attack method.

Results

Against Targeted Adversarial Attacks

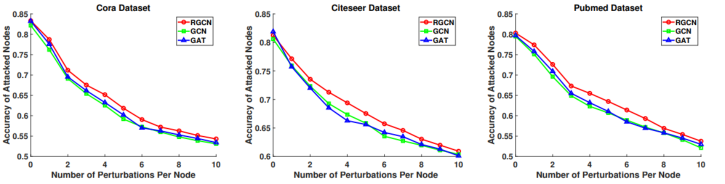


Figure 3: Results of different methods when adopting RL-S2V as the attack method.

Thank You!