# Semi-Supervised Classification with Graph Convolutional Networks

## -Previous works

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}}, \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2 = f(X)^\top \Delta f(X). \qquad (1)$$

Here, $L_0$ denotes the supervised loss, $\lambda$ is a weighing factor and $X$ is a matrix of node feature vectors, and $\Delta = D - A$ denotes the unnormalized graph Laplacian of an undirected graph $G$, where $A$ is an adjacency matrix and $D$ is a degree matrix. There was a problem of classifying nodes when using a graph Laplacian regularization term in the loss function ($L_{\text{reg}}$ in equaiton1) If we use equation1, then we need to assume that adjacent nodes are likely to share the same label. And it restricts modeling capacity, such as every edge should contain similarity between adjacent nodes. To deal with the problem, this work introduced the neural network model $f(X, A)$ and train on a supervised target $L_0$ for all nodes with labels. Using adjacency matrix allows the model to use information that which connectivity of nodes.

## -Motivation

### -How can we use graphs as an input of Convolutional Network?

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right). \qquad (2)$$

Equation2 shows the propagation rule of Graph Convolutional Network(GCN). Here, $\tilde{A} = A + I_N$ is the adjacency matrix with added self-connections, $\tilde{D}$ is a degree matrix with added self-connections, $W^{(l)}$ is a layer-specific trainable weight matrix, $\sigma(\cdot)$ denotes an activation function, and $H^{(l)}$ is the matrix of activations in the $l^{th}$ layer.

#### -Spectral Graph Convolutions

In signal processing, we use frequency domain to interpret the signal, such as using Fourier transform. Similarly, this work considers spectral convolutions on graphs defined as the multiplication of signal with a filter $g_\theta$ parametrized by $\theta$ in the Fourier domain.

$$g_\theta \star x = U g_\theta U^\top x, \qquad (3)$$

Here, $U$ is the matrix of eigenvectors of the normalized Laplacian. However, evaluating equation3 and computing eigendecomposition of Laplacian is computiationally expensive. Therefore, this work approximated to Chebyshev polynomials as follows.

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) , \tag{4}$$

Then, we have

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x , \tag{5}$$

The complexity of equation5 is $O(|\epsilon|)$ We further approximate $\lambda_{max} = 2$ in equation5, then it simplifies to

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 \left(L - I_N\right) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x , \tag{6}$$

by taking first two terms of equation5. In practice, it is more beneficial to constrain the number of parameters to prevent overfitting and to minimize the number of operations per layer. Therefore, we use a single parameter as below.

$$g_{\theta} \star x \approx \theta \left( I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x , \tag{7}$$

However, using deep neural network could lead to exploding or vanishing gradients. To alleviate this problem, this work introduce the renormalization trick; adding self-connections to adjacency matrix and a degree matrix. In conclusion, we get the following, where $Z$ is the convolved signal matrix.

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta , \tag{8}$$

If we consider a two-layer GCN, then forward model takes the simple form:

$$Z = f(X, A) = \mathrm{softmax}\Big( \hat{A} \ \mathrm{ReLU}\Big( \hat{A} X W^{(0)} \Big) W^{(1)} \Big) . \tag{9}$$

For semi-supervised classification, we evaluate the cross-entropy error over all labeled examples:

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf} \,, \tag{10}$$

## -Experiments

### -Datasets

Citation networks : Citeseer, Cora, Pubmed

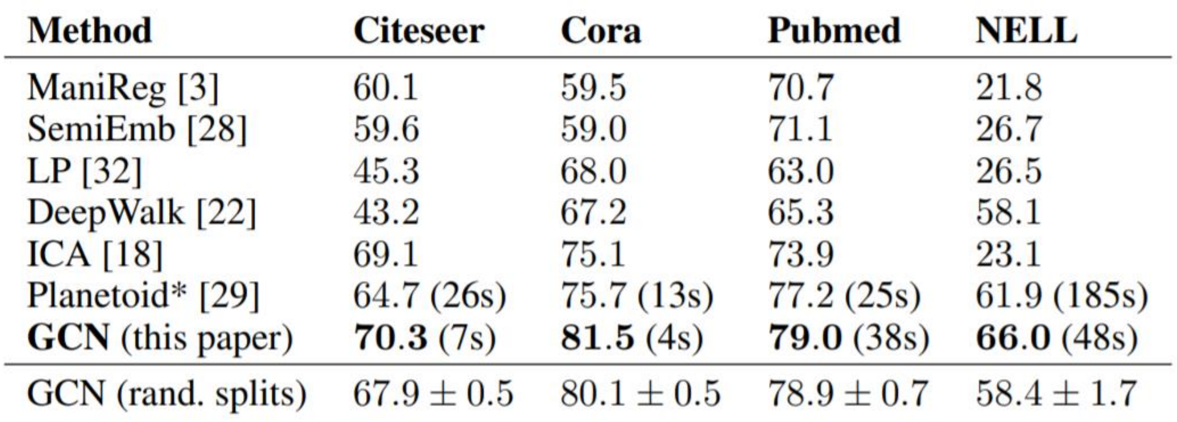
Treat the citation links as edges and each document has a class label.

NELL : dataset extracted from the knowledge graph(Carlson et al. 2010)

Random graphs : For a dataset with `N` nodes, create a random graph assigning `2N` edges uniformly at random. Also, take the identity matrix `X`. Add dummy labels for every node.

### -Results

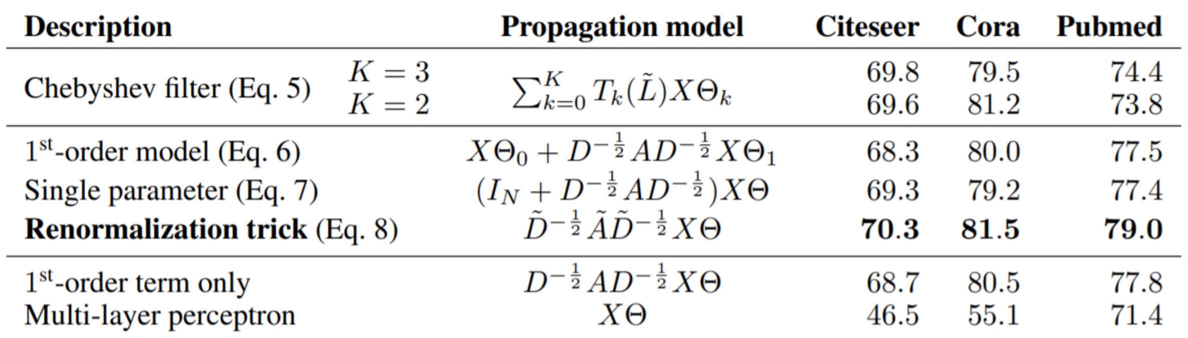
Table 2: Summary of results in terms of classification accuracy (in percent).



| 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) | $67.9 \pm 0.5$ | $80.1 \pm 0.5$ | $78.9 \pm 0.7$ | $58.4 \pm 1.7$ |

We could see that this work achieves the highest accuracy and training time compared to the other baseline.

Table 3: Comparison of propagation models.



| Description | | Propagation model | Citeseer | Cora | Pubmed |
|---|---|---|---|---|---|
| Chebyshev filter (Eq. 5) | $K = 3$ | $\sum_{k=0}^{K} T_k(\tilde{L})X\Theta_k$ | 69.8 | 79.5 | 74.4 |
| | $K = 2$ | | 69.6 | 81.2 | 73.8 |
| 1st-order model (Eq. 6) | | $X\Theta_0 + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta_1$ | 68.3 | 80.0 | 77.5 |
| Single parameter (Eq. 7) | | $(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})X\Theta$ | 69.3 | 79.2 | 77.4 |
| **Renormalization trick** (Eq. 8) | | $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$ | **70.3** | **81.5** | **79.0** |
| 1st-order term only | | $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta$ | 68.7 | 80.5 | 77.8 |
| Multi-layer perceptron | | $X\Theta$ | 46.5 | 55.1 | 71.4 |

Renormalization trick shown in equation 8 shows the best performance compared to the other propagation models as described in equation 5, 6, 7, 8.

## -Limitations

1. Memory requirement

Memory requirement grows linearly in the size of the dataset. Therefore, large graphs could not fit in GPU memory. To address this problem, we could use CPU instead or mini-batch stochastic gradient descent. However, for large and densely connected graph datasets, we need further approximations to use mini-batch stochastic gradient descent.

2. Directed edges and edge features

This work does not support edge features and is limited to undirected graphs.

3. Limiting assumptions

We assumed that self-connections and edges to neighboring nodes have the same contribution. However, it would not be beneficial to some datasets.