Semi-Supervised Classification with Graph Convolutional Networks

Thomas N. Kipf, Max Welling

UAmsterdam

Presenter: Jack Lanchantin
`https://qdata.github.io/deep2Read`

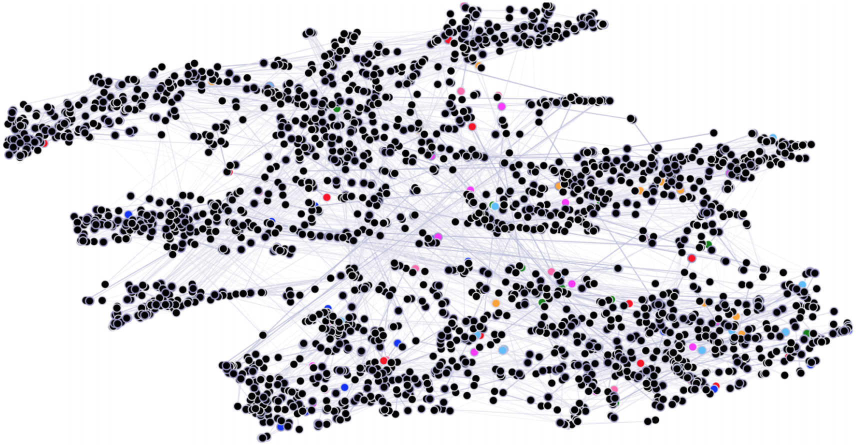
Outline

Outline

Definitions

- Undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$
- Adjacency matrix $A \in \mathbb{R}^{N \times N}$ (binary or weighted)
- Degree matrix $D_{ii} = \sum_j A_{ij}$
- Laplacian: $L = D - A$
- Normalized Laplacian: $L = I - D^{-1/2} A D^{-1/2}$

Task: Semi-Supervised Classification



Previous methods

Label information is smoothed over the graph via some form of explicit graph-based regularization (e.g. graph Laplacian regularization in the loss):

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}} \,, \tag{1}$$

$$\mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2 = f(X)^\top L f(X) \,. \tag{2}$$

- $\mathcal{L}_0$: supervised loss w.r.t. the labeled part of the graph
- $f(\cdot)$: neural network
- X is a matrix of node feature vectors X_i

Previous Work Drawbacks

- The formulation of Eq. 1 relies on the assumption that connected nodes in the graph are likely to share the same label.
- This assumption might restrict modeling capacity, as graph edges need not necessarily encode node similarity, but could contain additional information.

This Paper

- Encode the graph structure directly using a neural network model $f(X, A)$ and train on a supervised target $\mathcal{L}_0$ for all nodes with labels, thereby avoiding explicit graph-based regularization in the loss
- Conditioning $f(\cdot)$ on A will allow the model to distribute gradient information from the supervised loss $\mathcal{L}_0$ and will enable it to learn representations all nodes (with and without labels)

Outline

Graph Convolutional Network (GCN)

GCNs consist of the following layer-wise propagation rule

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \tag{3}$$

- $\tilde{A} = A + I$ (adjacency matrix with added self-connections)
- $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
- $W^{(l)}$ is a layer-specific trainable weight matrix.
- $H^{(l)} \in \mathbb{R}^{N \times D}$ is the matrix of activations in the l^{th} layer; $H^{(0)} = X$.

Graph Convolutional Network (GCN)

In the following, we show that the form of this propagation is motivated via a first-order approximation of localized spectral filters on graphs

Spectral Graph Convolutions

Spectral convolutions on graphs are defined as multiplication of a signal $x \in \mathbb{R}^N$ (scalar for every node) with a filter $g_\theta = \mathrm{diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^N$ in the Fourier domain:

$$g_\theta \star x = U g_\theta U^\top x \,, \tag{4}$$

- U is the matrix of eigenvectors of the normalized graph Laplacian $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^\top$, with a diagonal matrix of its eigenvalues Λ, and $U^\top x$ being the graph Fourier transform of x
- We can view g_θ as a function of the eigenvalues of L, i.e. $g_\theta(\Lambda)$.

Spectral Graph Convolutions

Spectral convolutions on graphs are defined as multiplication of a signal $x \in \mathbb{R}^N$ (scalar for every node) with a filter $g_\theta = \text{diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^N$ in the Fourier domain:

$$g_\theta \star x = U g_\theta U^\top x \,, \tag{4}$$

- U is the matrix of eigenvectors of the normalized graph Laplacian $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^\top$, with a diagonal matrix of its eigenvalues Λ, and $U^\top x$ being the graph Fourier transform of x
- We can view g_θ as a function of the eigenvalues of L, i.e. $g_\theta(\Lambda)$.

However, multiplication with the eigenvector matrix U is $\mathcal{O}(N^2)$

Spectral Graph Convolutions with Chebyshev polynomials

To circumvent this problem, $g_\theta(\Lambda)$ can be approximated by a truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to K^{th} order:

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

$\theta' \in \mathbb{R}^K$ is now a vector of Chebyshev coefficients. Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$.

Spectral Graph Convolutions with Chebyshev polynomials

Plugging $g_{\theta'}(\Lambda)$ back into our definition of a convolution of a signal x with a filter $g_{\theta'}$, and using the equality $(U\Lambda U^\top)^k = U\Lambda^k U^\top$, we now have:

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \,, \tag{6}$$

where $\tilde{L} = L - I$

Spectral Graph Convolutions with Chebyshev polynomials

Plugging $g_{\theta'}(\Lambda)$ back into our definition of a convolution of a signal x with a filter $g_{\theta'}$, and using the equality $(U\Lambda U^\top)^k = U\Lambda^k U^\top$, we now have:

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \,, \qquad (6)$$

where $\tilde{L} = L - I$

- This expression is now K-localized since it is a K^{th}-order polynomial in the Laplacian, i.e. it depends only on nodes that are at maximum K steps away from the central node (K^{th}-order neighborhood).
- The complexity of Eq. 6 is $\mathcal{O}(|\mathcal{E}|)$, i.e. linear in the number of edges

Approximated Spectral Graph Convolution

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,, \tag{6}$$

A neural network model based on graph convolutions can therefore be built by stacking multiple convolutional layers of the form of Eq. 6, each layer followed by a point-wise non-linearity.

Approximated Spectral Graph Convolution

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,, \tag{6}$$

- Further simplify this by limiting $K = 1$, i.e. a function that is linear w.r.t. L and therefore a linear function on the graph Laplacian spectrum.
- Successive application of filters of this form then effectively convolve the k^{th}-order neighborhood of a node, where k is the number of successive filtering operations or convolutional layers in the model.

Approximated Spectral Graph Convolution

Setting $K = 1$, Eq. 6 simplifies to:

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 (L - I)\, x \tag{7}$$
$$= \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x\,, \tag{8}$$

with two free parameters θ'_0 and θ'_1.

We further constrain $\theta = \theta'_0 = -\theta'_1$ to minimize the number of operations (such as matrix multiplications) per layer:

$$g_\theta \star x \approx \theta \left(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x\,, \tag{9}$$

Approximated Spectral Graph Convolution

- $I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ now has eigenvalues in the range $[0, 2]$.
- Repeated application of this operator can therefore lead to numerical instabilities and exploding/vanishing gradients
- To alleviate this problem, a *renormalization trick* is used:

$$I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$$

with $\tilde{A} = A + I$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

Approximated Spectral Graph Convolution

Final approximation for scalar inputs x and a vector of features θ:

$$g_\theta \star x \approx \theta \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right) x \tag{10}$$

Full Layer-Wise Linear Model

We can generalize this definition to a signal $X \in \mathbb{R}^{N \times C}$ with C input channels (i.e. a C-dimensional feature vector for every node) and F filters or feature maps as follows:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \,, \tag{11}$$

where $\Theta \in \mathbb{R}^{C \times F}$ is now a matrix of filter parameters and $Z \in \mathbb{R}^{N \times F}$ is the convolved signal matrix.

This filtering operation has complexity $\mathcal{O}(|\mathcal{E}|FC)$

Outline

Outline

GCN For Semi-supervised Classification

- We first calculate $\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ in a pre-processing step.
- 2 layer network is then represented by:

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A} X W^{(0)}\Big)\ W^{(1)}\Big)\ . \qquad (12)$$

Here, $W^{(0)} \in \mathbb{R}^{C \times H}$ is an input-to-hidden weight matrix for a hidden layer with H feature maps. $W^{(1)} \in \mathbb{R}^{H \times F}$ is a hidden-to-output weight matrix. The softmax is applied row-wise.

GCN For Semi-supervised Classification

We then evaluate the cross-entropy error over all labeled examples:

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf} \, , \tag{13}$$

where $\mathcal{Y}_L$ is the set of node indices that have labels.

GCN For Semi-supervised Classification

We then evaluate the cross-entropy error over all labeled examples:

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf} \, , \tag{13}$$

where $\mathcal{Y}_L$ is the set of node indices that have labels.

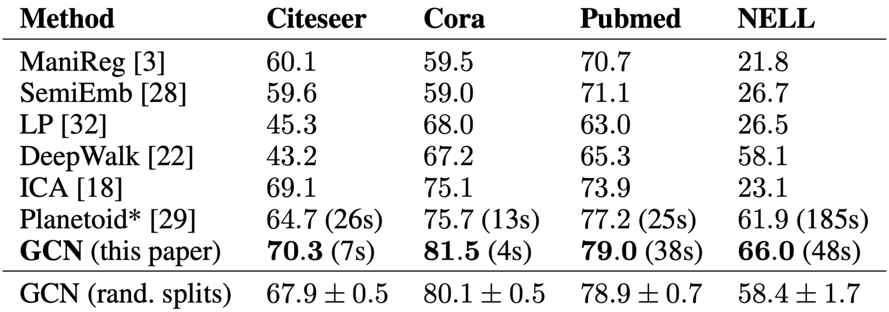
Batch gradient descent is used on the full dataset for every training iteration. Using a sparse representation for A, memory requirement is $\mathcal{O}(|\mathcal{E}|)$, i.e. linear in the number of edges.

Outline

Experiments

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

Results

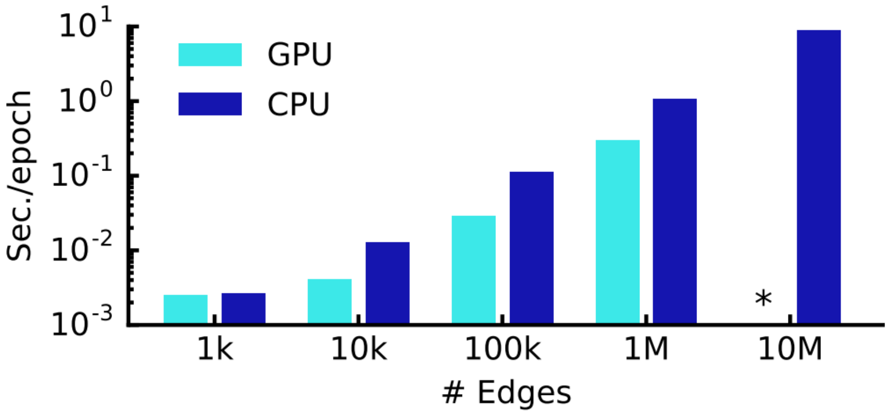


Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0**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

Results on Different Propagation Types

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)		$X\Theta_0 + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X\Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)		$(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) X\Theta$	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		$D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X\Theta$	68.7	80.5	77.8
Multi-layer perceptron		$X\Theta$	46.5	55.1	71.4

Runtimes on Random Graphs



Outline

Limitations

- **Memory**: For large graphs that do not fit in GPU memory, training on CPU can still be a viable option.
- **No Edge Features Allowed**: Limited to undirected graphs (weighted or unweighted).
- **Node Locality** Implicitly assumed locality (dependence on the K^{th}-order neighborhood for a GCN with K layers) and equal importance of self-connections vs edges to neighboring nodes.

Conclusion

- By their approximations, this work was able to achieve a rapid speedup in graph convolutions ($\mathcal{O}(N^2) \rightarrow \mathcal{O}(|\mathcal{E}|)$) with greater accuracy than previous methods
- The major drawbacks are the requirement to fit the entire graph into memory, working in the spectral domain, and only applicable to transductive tasks