SMAI Project

Semi-Supervised Classification With Graph Convolutional Networks

Team 35
Chirag Shameek Sahu (2019102006)
Isha Gupta (2019101111)
Ashish Gupta (2019101016)
Dinesh Garg (2019101085)

Repository Link

[Team 35- Semi-Supervised Classification With Graph Convolution Network]

Introduction

Many critical real-world datasets come in the form of graphs or networks such as the social network, knowledge graphs, protein-interaction networks, the World Wide Web, etc. Yet, until recently, very little attention has been devoted to the generalisation of neural network models to such structured datasets.

This project implementation can be considered as a problem of classifying nodes (such as documents) in a graph (such as a citation network), where labels are only available for a small subset of nodes. A scalable approach for semi-supervised learning on graph-structured data is presented that is based on an efficient variant of convolutional neural networks which operate directly on graphs.

This problem can be framed as graph-based semi-supervised learning, where label information is smoothed over the graph via some form of explicit graph-based regularization.

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

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

- $\mathcal{L}_0$ the supervised loss w.r.t. the labeled part of the graph
- $f(.)$ a neural network like differentiable function
- λ weighing factor
- X matrix of node feature vectors X_i

Here, Δ=D-A denotes the unnormalized graph Laplacian of an undirected graph G = (V, ε), with N nodes v_i ε V, edges (v_i, v_j) ∈ ε an adjacency matrix A ε R^{NxN} (binary or weighted) and a degree matrix $D_{ii} = \Sigma_j A_{ij}$.

The above formulation relies on the assumption that connected nodes in the graph are likely to share the same label. Here, we will 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 function. Further, conditioning *f(.)* on the adjacency matrix of the graph will allow the model to distribute gradient information from the supervised loss $\mathcal{L}_0$ and will enable it to learn representations of nodes both with and without labels.

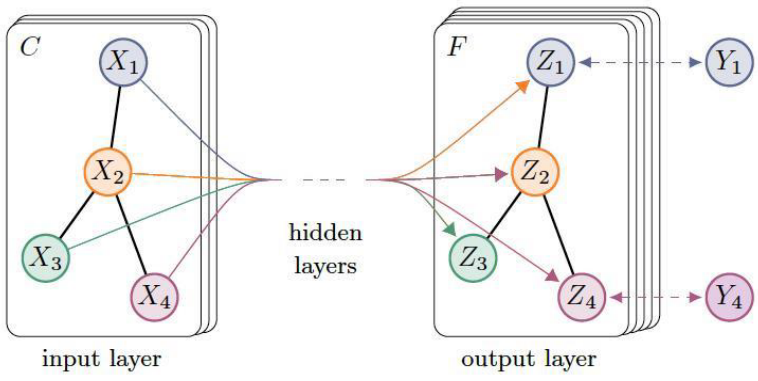
Key Implementation

Hence, there is a two step process for the implementation of this procedure.

- Introduce a simple and well-behaved layer-wise propagation rule for neural network models which operate directly on graphs and show how it can be motivated from a first-order approximation of spectral graph convolutions.
- Demonstrate how this form of a graph-based neural network model can be used for fast and scalable semi-supervised classification of nodes in a graph.

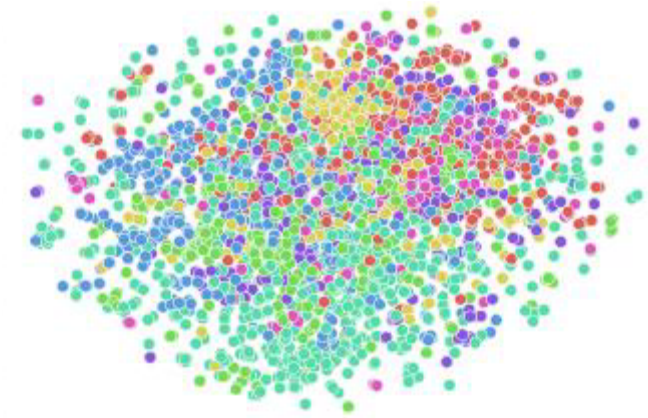
Graph Convolution Network

A Graph Convolutional Network, or GCN, is an approach for semi-supervised learning on graph-structured data. It is based on an efficient variant of convolutional neural networks which operate directly on graphs. The choice of convolutional architecture is motivated via a localized first-order approximation of spectral graph convolutions. The model scales linearly in the number of graph edges and learns hidden layer representations that encode both local graph structure and features of nodes.



(a) Graph Convolutional Network

The goal here is to learn a function of signals/features on a graph $G=(V,E)$. Here, the inputs are a feature description x_i for every node i, summarized in a $N\times D$ feature matrix X (where N is the number of nodes and D is the number of input feature) and a representative description of the graph structure in matrix form, typically in the form of an adjacency matrix A (or some function thereof). The output is a node level Z (an $N\times F$ feature matrix, where F is the number of output features per node).



Cora t-SNE visualization

Every neural network layer can there be written as a non linear function $H^{(l+1)}=f(H^{(l)},A)$ with $H^{(0)}=X$ and $H^{(L)}=Z$ (or z for graph level outputs) and L being the number of layers. The specific models then differ only in how $f(\cdot,\cdot)$ is chosen and parameterized.

Fast Approximate Covolutions on Graphs

We consider a multi-layer Graph Convolutional Network (GCN) for a specific graph-based neural network model $f(X,A)$ with the following layer-wise propagation rule:

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

- $A=A+I_N$ is the adjacency matrix of the undirected graph G with added self connections.
- I_N is the identity matrix
- $D_{ii} = \Sigma_j A'_{ij}$
- $W^{(l)}$ is a layer specific trainable weight matrix
- $H^{(l)} \in R^{N\times D}$ is the matrix of activations in the l^{th} layer
- $H^{(0)} = X$

Spectral Graph Convolutions

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

$$g_\theta \star x = U g_\theta U^\top x$$

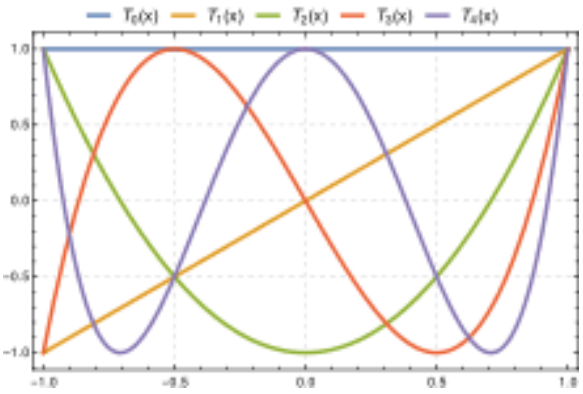
- U is the matrix of eigenvectors of the normalized graph Laplacian
- $U^T x$ being the graph Fourier transform x

However! Evaluating this is computationally expensive, multiplication with the eigenvector matrix U is $O(N^2)$ and computing the eigen decomposition of L in the first place can be expensive for large graphs.

Chebyshev Polynomial Approximation

Furthermore, computing the eigen decomposition of L in the first place might be prohibitively expensive for large graphs. To circumvent this problem, it was suggested in that $g_\theta(\Lambda)$ can be well-approximated by a truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to K^{th} order

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



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

where $L' = 2/\lambda_{max}\,(L - I_N)$ can be verified by $(U\Lambda U^T)^k = U\Lambda^k U^T$.

Note that 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 evaluating this is $O|E|$, i.e., linear in the number of edges.

Layer-Wise Linear Model

A neural network model based on graph convolutions can therefore be built by stacking multiple convolutional layers of this form, each layer followed by a point wise non linearity. We limit the layer wise convolution operation to K=1, i.e., a function that is linear with respect to L and therefore a linear function on the graph Laplacian spectrum.

We intuitively expect that such a model can alleviate the problem of overfitting on local neighborhood structures for graphs with very wide node degree distributions, such as social networks, citation networks, knowledge graphs and many other real-world graph datasets. Additionally, for a fixed computational budget, this layer-wise linear formulation allows us to build deeper models, a practice that is known to improve modeling capacity on a number of domains.

Further approximate λ_{max}=2, since neural network parameters will adapt to this change in scale during training, the above equation simplifies to

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

with two free parameters $\theta_0'$and θ_1'. The filter parameters can be shared over the whole graph. Successive application of filters of this form then effectively convolve the kth-order neighborhood of a node, where *k* is the number of successive filtering operations or convolutional layers in the neural network model.

It can be beneficial to constrain the number of parameters further to address overfitting and to minimize the number of operations (such as matrix multiplications) per layer. This can be done as follows

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

with a single parameter $\theta=\theta_0'=-\theta_1'$. $L=I_N-D^{-1/2}AD^{-1/2}$ now has eigenvalues in the range [0,2] that suggest a renormalization trick, to address numerical instabilities, and/or vanishing/exploding gradients $I_N+D^{-1/2}AD^{-1/2}\rightarrow D'^{-1/2}A'D'^{-1/2}$, with $A'=A+I_N$ and $D'_{ii}=\Sigma_j A_{ij}$.

Output

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

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

- $\Theta \in R^{C\times F}$ is now a matrix of filter parameters
- $Z \in R^{(N\times F)}$ is the convolved signal matrix

This filtering operation has complexity $O(|E|FC)$.

Semi-Supervised Node Classification

We can relax certain assumptions typically made in graph-based semi-supervised learning by conditioning our model f(X;A) both on the data X and on the adjacency matrix A of the underlying graph structure. We expect this setting to be especially powerful in scenarios where the adjacency matrix contains information not present in the data X, such as citation links between documents in a citation network or relations in a knowledge graph.

We consider a two-layer GCN for semi-supervised node classification on a graph with a symmetric adjacency matrix A (binary or weighted).

The forward model is then $Z=f(X,A) = softmax\,(A'\,ReLU(A'XW^{(0)})\,W^{(1)})$, where they first compute in pre processing: $A= D'^{-1/2}\,A'D'^{-1/2}$.

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

For semi-supervised multi class classification, they evaluate the cross entropy error over all labelled examples as:

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

where γ_L is the set of node indices that have labels.

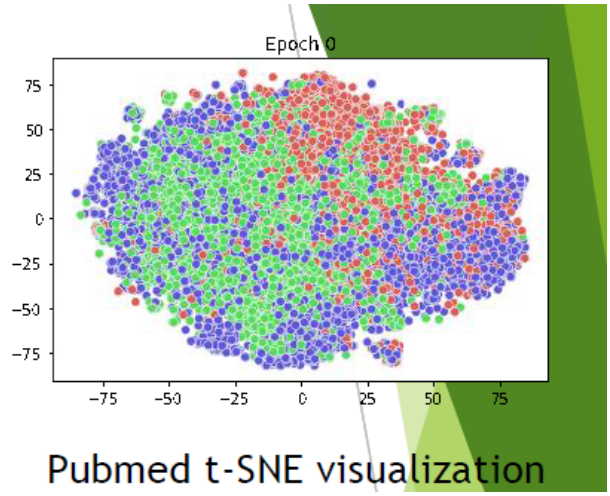
Experimentation

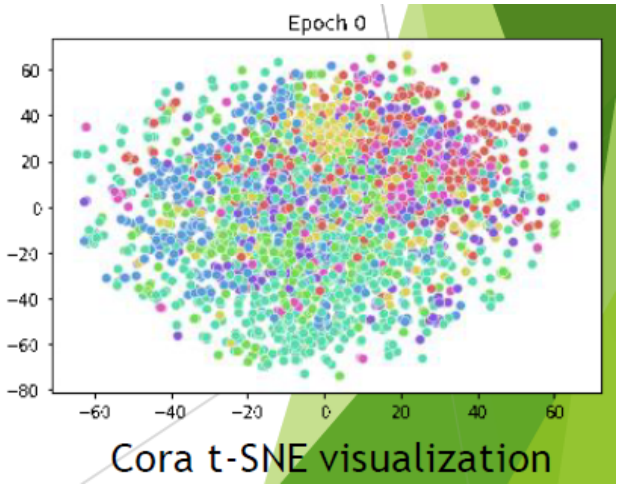
We test in a number of experiments: semi-supervised document classification in citation networks, semi-supervised entity classification in a bipartite graph extracted from a knowledge graph, an evaluation of various graph propagation models and a run-time analysis on random graphs.

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

Result: Semi-Supervised Node Embeddings

Consider the following semi-supervised learning setup: we add a softmax layer on top of our model and train using only a single labeled example per class (i.e. a total number of 4 labeled nodes). We train for 200 training iterations using Adam with a learning rate of 0.01 on a cross-entropy loss.

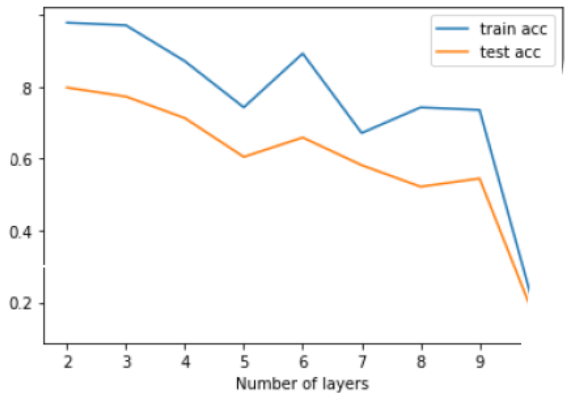




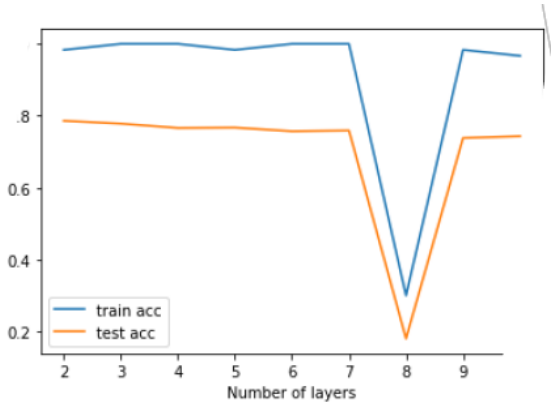
GIFs for the above are also submitted.

Test Accuracy	CORA	PUBMED
200 Iterations	80.30%	78.70%

Model Depth Experimentation



Cora



PubMed

In these experiments, we investigate the influence of model depth (number of layers) on classification performance. We report results on a 5-fold cross-validation experiment on the Cora, Citeseer and Pubmed datasets using all labels. In addition to the standard GCN model, we report results on a model variant where we use residual connections between hidden layers to facilitate training of deeper models by enabling the model to carry over information from the previous layer's input.

On each cross-validation split, we train for 200 epochs (without early stopping) using the Adam optimizer with a learning rate of 0.01.

Label Propagation