Semi-Supervised classification with graph convolutional networks

MLRG March 31st , 2021

Lironne Kurzman

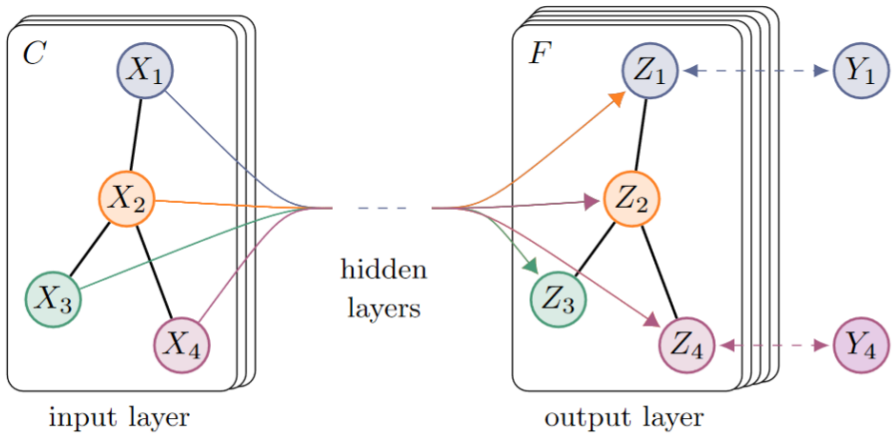
Outline

Graph Convolutional Networks

Spectral Graph Convolutions

Semi-Supervised Classification with Graph Convolutional Networks by Thomas N.Kipf and Max Welling

Graph Convolutional Networks



(a) Graph Convolutional Network

Graph Convolutional Networks

The goal learn a function of signals/features on a graph $G = (V, E)$

Input:

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)

A representative description of the graph structure in matrix form; typically, in the form of an adjacency matrix A (or some function thereof)

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

Graph Convolutional Networks

Every neural network layer can then 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), L being the number of layers. The specific models then differ only in how $f(\cdot,\cdot)$ is chosen and parameterized.

For further reading and a more in-depth tutorial

https://tkipf.github.io/graph-convolutional-networks/ by Thomas N.Kipf

Spectral Graph Convolutions

Spectral Graph Convolutions

decomposing a graph into a combination (usually, a sum) of simple elements (wavelets, graphlets). To have some nice properties of such a *decomposition*, these simple elements are usually *orthogonal*, i.e. mutually linearly independent, and therefore form a *basis*.

But when we talk about graphs and graph neural networks (GNNs), "spectral" implies *eigen-decomposition* of the graph Laplacian *L*

Eigen Decomposition

Let $\boldsymbol{A}$ be a square $n \times n$ matrix with n linearly independent eigenvectors q_i (where $i = 1, \ldots, n$). Then $\boldsymbol{A}$ can be factorized as

where $\boldsymbol{Q}$ is the square $n \times n$ matrix whose ith column is the eigenvector q_i of $\boldsymbol{A}$, and $\boldsymbol{\Lambda}$ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, $\Lambda_{ii} = \lambda_i$. Note that only diagonalizable matrices can be factorized in this way.

Graph Laplacian

- Intuitively, the graph Laplacian shows in what directions and how *smoothly* the "energy" will diffuse over a graph if we put some "potential" in node i.
- A typical use-case of Laplacian in mathematics and physics is to solve how a signal (wave) propagates in a dynamic system.

Given a simple graph (no self loops / no more than 1 edge between 2 nodes) G with n vertices, its Laplacian matrix $L_{n\times n}$ is defined as:

$$\mathrm{L} = D - A$$

where D is the degree matrix and A is the adjacency matrix of the graph. Since G is a simple graph, A only contains 1s or 0s and its diagonal elements are all 0s.

Spectral Graph Convolutions

For further reading and implementation:

https://towardsdatascience.com/spectral-graph-convolution-explained-and-implemented-step-by-step-2e495b57f801 by Boris Knyazev

Semi-Supervised Classification with Graph Convolutional Networks

by Thomas N.Kipf and Max Welling

The problem

- classifying nodes (such as documents) in a graph (such as a citation network), where labels are only available for a small subset of nodes. 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

- Key Assumption they made that connected nodes in the graph are likely to share the same label

The Model + Theory

Loss Function

- using a graph Laplacian regularization term in the loss function

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

- $\mathcal{L}_0$ denotes the supervised loss w.r.t. the labeled part of the graph
- $f(\cdot)$ can be a neural network-like differentiable function,
- λ is a weighing factor
- X is a matrix of node feature vectors Xi.
- $\Delta = D - A$ denotes the unnormalized graph Laplacian of an undirected graph

$G = (V, E)$ with N nodes $vi \in V$, edges(vi,vj)∈ E,

- adjacency matrix A∈R^ N×N(binary or weighted)
- degree matrix $Dii = \sum jAij$.

Fast Approximate Convolutions on Graphs

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

- $\tilde{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections.
- I_N is the identity matrix
- Dii=∑j~ Aij
- W(l)is a layer-specific trainable weight matrix
- H(l)∈R^N×D is the matrix of activations in the lth layer
- H(0)=X

Spectral Graph Convolutions

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

- Filter $g\theta = diag(\theta)$ parameterized by $\theta \in R^N$
- U is the matrix of eigenvectors of the normalized graph Laplacian

$L = I_N - D^{-1/2} A D^{-1/2} = U \Lambda U^T$, with a diagonal matrix of its eigenvalues Λ

- $U^T\, x$ being the graph Fourier transform of 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

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

Rescaled $\widetilde{\Lambda} = \frac{2}{\lambda max}\Lambda - I_N$.

λ max denotes the largest eigenvalue of L. $\theta' \in R^K$ is now a vector of Chebyshev coefficients.

The Chebyshev polynomials are recursively defined as
$Tk(x) = 2xT\_k - 1(x) - T\_k - 2(x)$, with $T0(x) = 1$ and $T1(x) = x$.

Chebyshev Polynomial Approximation

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

Where $L = \frac{2}{\lambda max} L - I_N$

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 Kth-order polynomial in the Laplacian, i.e. it depends only on nodes that are at maximum K steps away from the central node (Kth-order neighborhood).

The complexity of evaluating this is O(|E|), i.e. linear in the number of edges.

The Big Picture

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

- stacking multiple convolutional layers of this form, each layer followed by a point-wise non-linearity.
- limit the layer-wise convolution operation to K= 1, i.e. a function that is linear w.r.t. L and therefore a linear function on the graph Laplacian spectrum.

Further Approx.

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

- further approximate λmax≈2, since neural network parameters will adapt to this change in scale during training
- two free parameters θ′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

Further Approx.

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

- single parameter θ= θ′0 =–θ′1
- $L = I_N - D^{-1/2} A D^{-1/2}$ now has eigenvalues in the range [0,2] the suggest a renormalization trick, to address numerical instabilities, and/or vanishing/exploding gradients
- $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$
- $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

Output

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

- 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
- $\Theta \in R^{C \times F}$ is now a matrix of filter parameters and $Z \in R^{N \times F}$ is the convolved signal matrix. This filtering operation has complexity O(|E|FC)

Semi-Supervised Node classification

- The forward model is then:

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

- Where they first compute in pre-processing :

$$\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$$

Semi-Supervised Node classification

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

- W(0)∈RC×H is an input-to-hidden weight matrix for a hidden layer with H feature maps. W(1)∈RH×F is a hidden-to-output weight matrix
- For semi-supervised multi-class classification, they evaluate the cross-entropy error over all labeled examples
- $$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}\,,$$

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

Experimental Set up

Implementation

- TensorFlow for an efficient GPU-based implementation using sparse-dense matrix multiplications. The computational complexity of evaluating is then O(|E|CHF), i.e. linear in the number of graph edges.
- trained using gradient descent. Perform batch gradient descent using the full dataset for every training iteration, which is a viable option if datasets fit in memory. (addressed later)

Datasets

- Citation Networks - three citation (Citeseer, Cora, Pubmed),

sparse bag-of-words feature vectors for each document and a list of citation links between documents.

treat the citation links as (undirected) edges and construct a binary, symmetric adjacency matrix A.

Each document has a class label

- NELL - dataset extracted from the knowledge graph,

assign separate relation nodes r1 and r2 for each entity pair (e1,r,e2) as (e1,r1) and (e2,r2).

Entity nodes are described by sparse feature vectors.

The semi-supervised task here considers the extreme case of only a single labeled example per class in the training set.

- Random graphs - For a dataset with N nodes create a random graph assigning 2N edges uniformly at random.

Assign identity matrix IN as input feature matrix X,

We add dummy labels Yi= 1 for every node.

Datasets - stats

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

Results

Semi-Supervised Node Classification

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

- GCN (rand. splits) - 10 randomly drawn dataset splits of the same size as in Yang et al. (2016).

Evaluation of Propagation Model

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

mean classification accuracy for 100 repeated runs with random weight matrix initializations.

Training Time Per Epoch

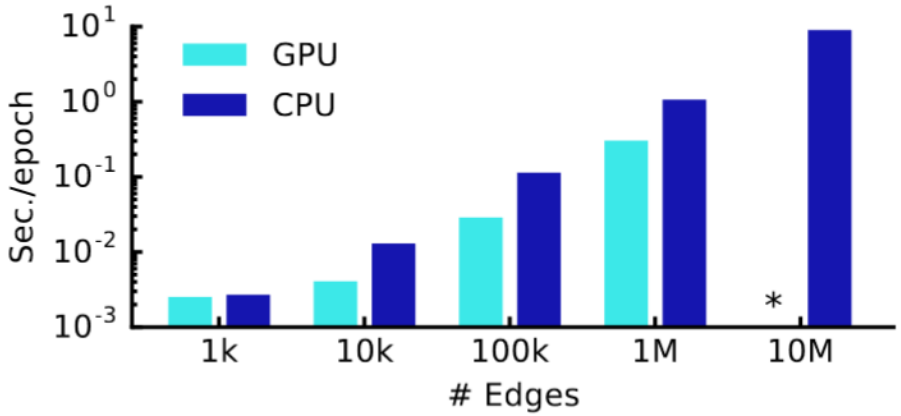


Figure 2: Wall-clock time per epoch for random graphs. (*) indicates out-of-memory error.

mean training time per epoch (forward pass, cross-entropy calculation, backward pass) for 100 epochs on simulated random graphs, measured in seconds wall-clock time.

Limitations

- Memory requirement : In the current setup with full-batch gradient descent, memory requirement grows linearly in the size of the dataset.
- Directed edges and edge features : The framework currently does not support edge features and is limited to undirected graphs
- Limiting Assumption: Through the approximations using the truncated expansion of Chebyshev polynomials , they implicitly assume locality (dependence on the Kth-order neighborhood for a GCN with K layers) and equal importance of self-connections vs. edges to neighboring nodes.

Questions?

Thank you