GNN 2 – A TRAINABLE GRAPH CONVOLUTIONAL NETWORK

Introduction

In the previous study, we have assembled the basic components of a GCN model, and demonstrated that even in the untrained state, the model is able to produce features useful for community detection. Nonetheless, those remarkable results are more of a coincidence than a norm, and a viable GCN model must necessarily be trained to optimize its performance. This session aims to use the basic principles of GCN model and mathematical optimization to handicraft a trainable GCN model without the use of any GCN packages.

Recapitulation

We shall recapitulate the conventional formulation of the GCN model, where

$$H_k = \varphi_k\left(\widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}H_{k-1}W_k\right).$$

Here, H_{k-1} is in the input to the kth layer of the network, and W_k is the weight matrix for the kth layer, and H_k is the output for the kth layer which would then become the input for the $(k+1)$th layer. For a network G, $\tilde{A}$ is the sum of the network's adjacency matrix and the identity matrix, and $\widetilde{D}$ is the degree matrix computed from $\tilde{A}$. In addition, φ_k is the nonlinear map for layer k. To make the conceptual relation simpler, we shall express output of the kth layer of the GCN model as a function F_k of the input, parameterized by the network G, and the weight matrix W_k, where

$$H_k = F_k(H_{k-1}; G, W_k).$$

Suppose that the model has l layers, the eventual output would be denoted simply as a function composition, where

$$H_l = [F_l(G, W_l) \circ F_{l-1}(G, W_{l-1}) \circ \cdots \circ F_1(G, W_1)](H_0)$$

$$= F(H_0; G, W_1, W_2, \cdots W_l).$$

F denotes the overall operation performed by the entire GCN model, which is a function of H_0 equivalent to the raw input X, which is an $n \times d$ matrix of n nodes and d covariates. The overall operation is parameterized by G, which in inherent to the network of interest, as well as the l weight matrices that are trainable.

Besides the GCN model, the method for optimization shall be outlined. For a vector-to-scalar map $f(|x\rangle)$, a minimization scheme would be based on the gradient vector $\nabla_f(|x\rangle)$, such that for an initialized value of $|x\rangle_0$, the minimizer is computed sequentially by

$$|x\rangle_{k+1} = |x\rangle_k - \lambda\nabla_f\left(|x\rangle_k\right).$$

The factor λ helps to control the step size of the minimization, such that a smaller λ results in more computational steps needed before convergence, whereas a larger λ reduces the computational steps needed, but may cause $|x\rangle$ to jump around the point of minimization.

Theoretical considerations for the network in GNN 1

We recall that GCN model experimented in the previous session eventually aims to classify all nodes into two communities based on the similarities in their three covariates. For this task, rather than the nonlinear map $ReLU$, a function more appropriate for classification should be used, of

which one potential option is the multinomial logit function, abbreviated MNL. For a vector input $|x\rangle$, the MNL states that

$$MNL_i(|x\rangle) = \frac{e^{x_i}}{\sum_j e^{x_j}}.$$

This ensures that each element of the function output is between 0 and 1, and all elements sum to one, which are amenable as probabilities. The MNL for a matrix simply operates on each row separately and returns a matrix. To measure the loss function $\mathcal{L}$, Kipf and Welling proposed that for MNL, the loss function should evaluate the cross-entropy error as

$$\mathcal{L} = -\sum_{l \in y_L} \sum_{k=1}^{K} Y_{lk} ln Z_{lk}.$$

Here, y_L is the set of nodes whose labels are known and are used to train the model, Z is the matrix output for MNL. In this context, Y should be a $n \times K$ matrix, where $Y_{ij} = 1$ if node i come from class j, and zero otherwise. Clearly, $\mathcal{L}$ approaches zero when $Y_{ij} = 1$ and Z_{ij} approaches one for all nodes, which indicates that the model unequivocally predicts the correct classes for the entire network.

For the network discussed in GNN 1, we only have two classes to predict. In addition, to make the model more parsimonious, it would be judicious to commence with one layer only, such that the overall model is

$$Z = MNL\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\tilde{X}W\right).$$

In this case, $\tilde{X}$ is the version of the input matrix X where all covariates are centered and standardized, to prevent explosive results upon exponential operations. In addition, we shall treat the labels for node 30 and node 90 as known, to explore how class labels from these two nodes could be used to train the model. Since only two nodes are known, the loss function reduces to

$$\mathcal{L}_1 = -(Y_{30,1} ln Z_{30,1} + Y_{30,2} ln Z_{30,2} + Y_{90,1} ln Z_{90,1} + Y_{90,2} ln Z_{90,2}).$$

Clearly, when the network structure and their covariates are known, $\mathcal{L}_1$ is a function of W only, which can be denoted as $\mathcal{L}_1(W)$. Here, W is a matrix instead of a vector, which appears inconsistent with the formulations of minimization in the previous section. Nonetheless, by the foundational postulates for partial derivatives, it is not hard to conceive of a matrix of derivatives Ω such that

$$\Omega_{ij} = \frac{\partial \mathcal{L}_1}{\partial W_{ij}} = \lim_{h \to 0} \frac{\mathcal{L}_1\left(W + hI_{\{ij\}}\right) - \mathcal{L}_1(W)}{h}.$$

Here $I_{\{ij\}}$ is the matrix with element (i, j) as one and all other elements zero. Generally, $\mathcal{L}_1$ is too complicated for the explicit derivation of the derivatives. Nonetheless, in practice, we can always use an h as small as permissible by the precision of the computer to obtain a reasonable approximation of Ω evaluated at W.

Therefore, the procedures to train the GCN model is as follows:

1. Initialize all elements of the weight matrix $W_{(0)}$.
2. Approximate the derivative matrix $\Omega\left(W_{(0)}\right)$ evaluated at $W_{(0)}$, by selection of a small h.
3. Update the weights as $W_{(1)} = W_{(0)} - \lambda\Omega\left(W_{(0)}\right)$, where λ is the step size.
4. Repeat procedures 2 and 3, until the derivatives are very close to zero.

Experimentation with one-layered GCN

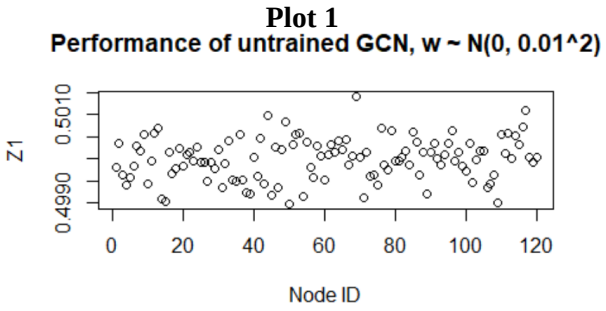
We shall then proceed to train the aforementioned GCN model on network samples from the stochastic block model. Nonetheless, to evaluate if the trained model offers superior performance over networks with less distinct demarcation of communities and increased noise, we shall set the probability of interaction within each community as 0.8, and that between two communities as 0.6. The distributions to sample the covariates are stated in Table 1.

Table 1: Distributions from which covariates are sampled

	x_1	x_2	x_3
Community 1	$N(20, 8^2)$	$N(40, 8^2)$	$N(22, 6^2)$
Community 2	$N(30, 8^2)$	$N(30, 8^2)$	$N(25, 4^2)$

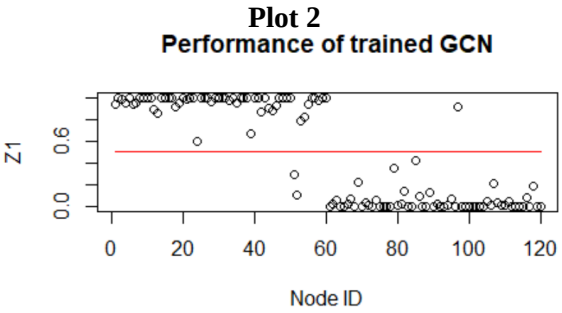
This scheme should create substantial overlap between the values of covariates between community 1 and 2. Indeed, this time, the initial weights sampled from a normal distribution of $N(0, 0.01^2)$ do not produce any useful classifications, as shown in Plot 1.

Plot 1



The model is then trained stepwise with a constant $\lambda = 1$, which stops when the Frobenius norm of Ω is less than 10^{-3}. The trained model is shown in Plot 2.

Plot 2



As seen from the plot, the trained model offers a much better performance in classifying the communities. For nodes 1 to 60, the vast majority of nodes have $Z_{\cdot,1}$ close to 1, which indicates that the model places them to community 1 almost unequivocally. Similarly, for nodes 61 to 120, most are placed in community 2 almost unequivocally. While some nodes have intermediate values of $Z_{\cdot,1}$, if we use $Z_{\cdot,1} = 0.5$ as the standard cutoff for classification, the model only makes 3 errors over 120 nodes.

Extension to multilayered GCN

Conceptually, the GCN model that involves multiple layers is not much different from that with only one layer. The main difference that for a l-layered model, we have l weight matrices, where we denote the weight matrix for layer k as W^k. This means that a loss function $\mathcal{L}_2$ would be the function of all l weight matrices, denoted as $\mathcal{L}_2(W^1, W^2, \cdots, W^l)$. Therefore, at each step when

we train the model, we calculate l matrices of derivatives, such that element (i, j) for the kth derivative matrix would be

$$\Omega_{ij}^{k} = \frac{\partial \mathcal{L}_2}{\partial W_{ij}^{k}} = \lim_{h \to 0} \frac{\mathcal{L}_2\left(W^1, \cdots, W^k + hI_{\{ij\}}, \cdots, W^l\right) - \mathcal{L}_2\left(W^1, \cdots, W^k, \cdots, W^l\right)}{h}.$$

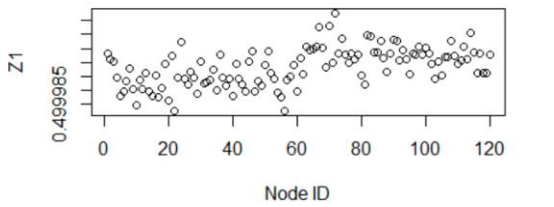
Aside from that, all other steps to train the model remain the same.

To demonstrate, we shall add another layer to the GCN model in the previous session, where we use a cubic nonlinear map B, which maps each element of the matrix to its cubic. The overall model is therefore denoted as

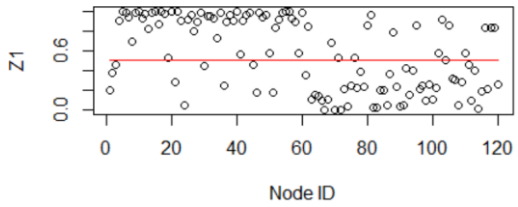
$$H_1 = B\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\tilde{X}W\right),$$
$$Z = MNL\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H_1 W\right).$$

The intermediate result, H_1, comprises two features. The weights are initialized by the normal distribution $N \sim (2, 0.01^2)$, and the model is trained with $\lambda = 1000$. The training terminates when the square root of the sum of the Frobenius norms squared for both derivative matrices is less than 10^{-5}. Plots 3 and 4 shows the untrained and trained models respectively.

Plot 3



Plot 4



Similar to Plot 2, the trained model would push many points to values of $Z_{\cdot,1}$ close to either zero or one. Nonetheless, it is clear that the overall performance of the two-layered GCN model is less ideal than the previously trained one-layered model. This shows that a more sophisticated model structure does not necessarily produce better results.

Discussions

The previous sessions have demonstrated the basic procedures to train GCN models in general. Nonetheless, the implementations of the procedures are far from ideal. Firstly, we did not adhere to any theoretical foundations to determine an appropriate step size, such that the choice for λ is overall by trial and error. This also means that we did not implement any scheme to adapt λ to the local structures of the derivatives. In addition, the update rules for the weight matrices were the simplest form of gradient descent, in spite of the existence of many other procedures that are more robust. The lack of these considerations contributed to a set of models with inadequate theoretical rigor and slow convergence rate, which would be addressed in later studies.

References

[1] TN Kipf & M Welling 2017, Semi-supervised Classification with Graph Convolutional Networks.