॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

GRAPH THEORY

PROJECT REPORT

Authored by :

Pulkit Joshi

Sahil Harish Batra

Summary

This report refers to work completed during our coursework project under the guidance of *Dr Anand Mishra of the Department of Computer Science, Indian Institute of Technology, Jodhpur. This is a short report (of 6 pages) that will summarize about the Graph Convolutional Networks. Further references have been added at the end.*

Acknowledgement

The project opportunity we had during my coursework was a great chance for learning and development. Bearing in mind previous, I am using this opportunity to express my deepest gratitude and special thanks to *Dr Anand Mishra* who in spite of being extraordinarily busy with his duties, took time out to guide us during this course project and gave us a chance to explore more into this field.

Graph Convolutional Networks

Index

Introduction

Graphs are a very important data-structure and are used in many important real-world datasets like social networks, chemical compounds, etc which have many practical applications. But Convolutional Neural Networks, which are an integral part of deep learning applications, do not work well on such structured and complex datasets made up of graphs, and this led to the development of the new architecture of neural networks Graph Convolutional Networks (GCNs) to work directly on graphs and leverage their structural information. Each layer of GCN has a non-linearising function, which does not add anything to the receptivity of the network and thus can be removed to simplify the model, and removing it introduces Simple Graph Convolution(SGC). Both GCNs and SGC are explained in detail below.

Graph Spectral Theory

Graphs are a generic way of representing many problems. Graphs are found in many places, in Kirchoff's Laws, In Signal processing etc. Thus knowing about basic signal processing on graphs is a must at this stage to learn about Graph Convolutional Networks. Hence we introduce basic graph terminologies in here.

1. Adjacency matrix and Normalisation

Suppose we have a network of sensors that are connected in some order, which sense the temperature of a region. Now suppose you need to give the temperature as output (y(n)) at an n^{th} position which is equal to the temperature at the n^{th} position as well as the sensors connected to the sensor. Then you can write y(n) as :

$$y(n) = x(n) + \sum_j A_{ij}\, x(j) \qquad (1)$$

Where x is the signal (temperature) at the n^{th} position and A_{ij} is 1 if i connected to j, else zero. Thus we can more elegantly write the above equation in terms of **A** the adjacency matrix. Now it can be written as:

$$Y = x + Ax$$

Similarly, A can be replaced with W for some weighted matrix. For the case of normalisation, W can be normalised in the form

$$y = 1/2(x + D^{-1} Wx),$$

where D is a diagonal matrix with entries $D_{nn} = \sum W_{nm}$

2. Graph Laplacian

Graph laplacian is an important operator for graph signals. Graph Laplacian **L** is given as :

$$\mathbf{L = D - W}$$

We defined A, W and D through our sensor network example to give a glimpse of signal, and graphs. Another thing to note is when the relationship between the nodes is symmetric then these matrices are also symmetric giving and such graphs are called undirected, while those which are not having such symmetric relationship are called directed graphs.

3. Shift operator on graphs

A time signal can be seen as a line graph. There we could easily shift the signal back and forth. However, in case of a not so line graph, this is not the case. Thus here shifting means that the signal at a node x moves to all its neighbouring nodes in a single shift. Thus we defined shift as:

$$\mathrm{x_{shifted}} = \mathrm{Ax}$$

However, we defined various matrices as A, W and D which can be used as the shift operator. Thus the shift operation is defined using **S**. Hence we can write

$$\mathrm{x_{shifted}} = \mathrm{Sx}$$

Now we can define a system on the graph as the input-output relation using S as:

$$y = H(S)x$$

Also, now a linear system can be defined in terms of the graph as :

a. Linear if the following holds: $H(s)(a_1x + a_2x) = a_1y_1 + a_2y_2$

b. Shift Invariant if the following holds: $H(s)(Sx) = S(H(s)x)$

4. Graph Fourier Transform

Graph signals employ adjacency/weighting matrix or the graph laplacian eigenvalue decomposition. We can decompose the laplacian into the orthonormal matrix of eigenvectors and a diagonal matrix of eigenvalues. Mathematically,

$$L = U \Lambda U^{-1}$$

The graph Fourier transform is thus defined as:

$$X = U^{-1}x$$

Now, we have $U^{-1} = U^T$, the k^{th} element of X i.e X_k is clearly a projection of the graph signal on the eigenvector. Similarly, inverse GFT is defined as:

$$x = UX$$

5. Output in Fourier domain

We previously defined, $y = H(S)x$, this can be written as:

$$y = \sum_{m=0}^{M-1} h_m L^m x$$

Now this can be further reduced using Graph Fourier Transform (and its inverse) to:

$$y = UH(\wedge)U^{-1}x$$

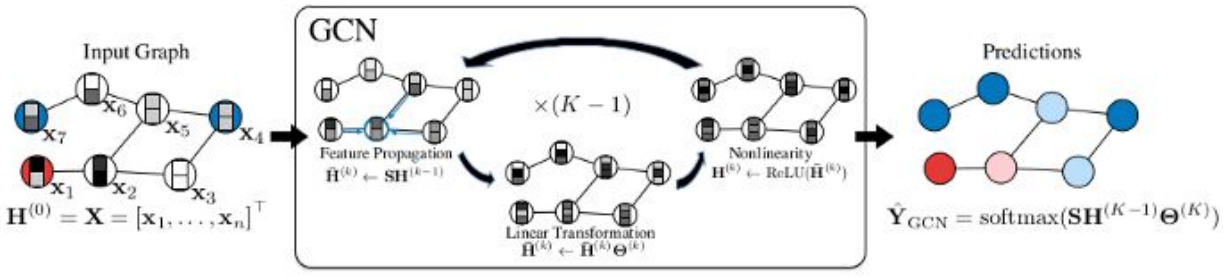
Finally, taking GFT both sides gives:

$$Y = H(\wedge)X$$

Simple Graph Convolution

Before going into Graph Convolutional Networks, we will look into the expected predecessor of Graph convolutional network, which is a simple linear model called Simple Graph Convolution. We will see that simple graph convolution is nothing but a feature engineering step that gives almost similar results as GCNs and thus can be regarded as a predecessor of GCNs. Then we would compare the results of GCN and SGC on "*Cora dataset*", and see that they give the same results on it. Our implementation of SGC can be found on Github. Further, we would use the spectral theory that we introduced above to give brief mathematics behind the GCNs.

1. Preview of Graph Convolution



Before getting into the Simple graph convolution model, we will preview how Graph convolution works in steps for a GCN classification. The significance of this part is merely to introduce the Simple graph

convolution and also how a basic GCN works. This part will be further brought up in upcoming pages.

a. **Feature Propagation**
We defined the shift operator in our graph spectral theory portion. Here we will use the same. During feature propagation in GCN, we have $H^{(k)}$ as our k^{th} layer which is obtained using the previous layer using the rule:

We say the same rule in spectral theory as (1), here we have normalised it further. This propagation can be written as :

$$H^k = SH^{(k-1)}$$

wheres is our shift operator S

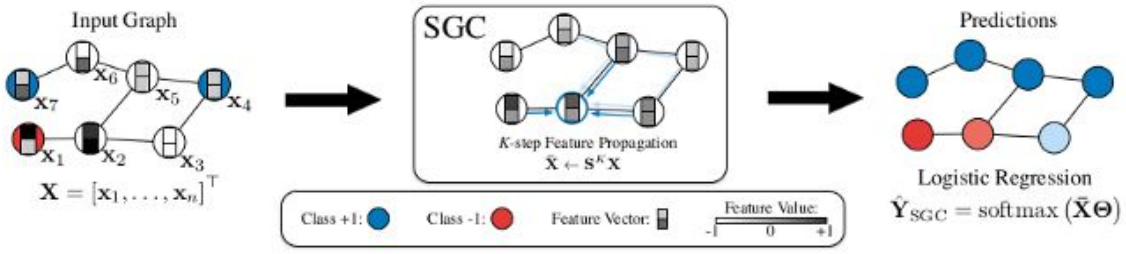
b. **Feature Transformation and nonlinear transition**
Each step in GCN has a learnable weight matrix $\Theta^{(k)}$ and is passed through an activation function(ReLU). Thus we can write the second step as:

$$H'^{(k)} = ReLU(H^{(k)}\Theta^{(k)})$$

c. **Classification**
For node classification, the last layer of GCN is passed through a softmax classifier as:

$$\mathbf{Y}_{gcn} = \text{softmax} (SH^{(k-1)} \Theta^{(k-1)})$$

2. Simple Graph Convolution



In the last part, we saw how GCN propagates and learns in a basic way, that is,

Feature propagation -> activation -> classification

Now, here we would simplify the GCN into a simple model by converting it into a linear model and which will convert the shifting step into nothing more than a feature engineering step that we see in machine learning problem

a. **Linearization**
We saw that GCN aggregates the labels/signal from one hop, sends it into a non-linear layer and gives us the output. However, let us hypothesize that real power that GCN get from is the signal propagation and not the non-linear step. After removing the non-linear part we can write:

$$H'^{(k)} = H^{(k)} \Theta^{(k)}$$

And we know,

$$H^k = SH^{(k-1)}$$

Thus using the base condition of $H^1 = SX\Theta^{(1)}$, we can write

$$H^k = S...SSX\Theta^{(1)}\Theta^{(2)}\Theta^{(3)}....\Theta^{(k)}$$

b. Classification

After linearising out our layers we feed the final step into a softmax classifier as:

$$Y_{gcn} = \text{softmax}\ (\ S...SSX\Theta^{(1)}\Theta^{(2)}\Theta^{(3)}....\Theta^{(k)}\ )$$

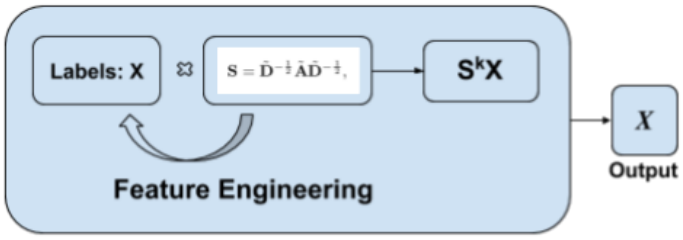
$$Y_{gcn} = \text{softmax}\ (\ S^kX\boldsymbol{\Theta}\ )$$

c. Feature engineering

Let us look at the equation of Y_{gcn}, we see our input is multiplied by a matrix S^k, thus we can replace S^kX by $\boldsymbol{X}$ to give:

$$Y_{gcn} = \text{softmax}\ (\ \boldsymbol{X\Theta}\ )$$

Thus the model is reduced to a simple linear classification with feature engineering step.



Feature Engineering in SGC

3. Implementation and Results of SGC

We implemented SGC using pytorch and further tested the results on cora dataset. SGC is not only faster than GCN and fastGCN but gives comparable results. For GCN, the literature says that the % accuracy is **81.5%** while SGC gives the accuracy as **81%.**

Results from our implementation as available on github

Graph Convolutional Networks

1. Graph Convolution

In graph spectral theory, we defined laplacian **L** as **D - A**, and further learnt about the eigen decomposition of $L = U\Lambda U^{-1}$. Further, GFT was defined as $X = U^{-1}x$ and Inverse GFT as $x = UX$. Thus, now we can define graph convolution over a signal x as g * x. Now, we know that convolution in the time domain is equivalent to the product in the Fourier domain. Using this result we can write:

$$g * x = U\ ((\ U^Tg).(U^Tx)) = UGU^Tx$$

where G is a diagonal matrix equivalent to the transform of g. Further, using approximations of k-th order polynomial, we can write UGU^Tx as

$$UGU^Tx \approx U\left(\sum_{i=0}^{k} {}_i\Lambda^i\right) U^t x$$

Using affine approximation (k = 1), $_0 = 2$ and $_1 = -$ with we can define a GCN convolution as:

$$g * x = (I + D^{-1/2} A D^{-½}) x$$

Further, normalisation of $I + D^{-1/2} A D^{-½}$ to $\hat{D}^{-½} A \hat{D}^{-½}$. Hence our graph convolution is finally defined and is ready for use in GCNs !!

2. Semi-Supervised Learning Using Graph Convolutional Networks

Suppose there is a graph with few labelled nodes, while other nodes are not labelled and you need to classify these nodes. Such a problem is named as semi-supervised classification here. Such a problem is solved using the graph Laplacian regularization term in the loss function. However, this assumes that connected nodes in graphs share the same label which is not always true. Thus, we can encode the graph structure using a neural network model like f(X, A) that is trained on the supervised target for all nodes with labels. This avoids the explicit use of graph-based regularization in the loss function. Also considering the adjacency matrix in f, will enable the model to learn representations of nodes both with and without labels by allowing it to distribute gradient information from supervised loss function.

3. Introducing GCN

This section introduces the Graph Convolutional Network model f(X, A).
Input to GCN is -

- A N x D feature matrix X, with node i being summarized in i^{th} row of the matrix, where N = number of nodes in the graph and D = number of input features.
- Representation of graph, which is usually a N x N adjacency matrix A.

GCN then produces a node-level output matrix Z, which is a N x F matrix, where F = number of output features.
Thus every layer can be represented as a nonlinear function,

$$H^{(l+1)} = f(H^{(l)}, A)$$

With $H^{(0)} = X$, and $H^{(L)} = Z$, where L = total number of layers.

4. Propagation rule for GCN

GCN uses the following layer-wise propagation rule -

$$f(H^{(l)}, A) = \boldsymbol{\sigma}(A H^{(l)} W)$$

Where $W^{(l)}$ is the l^{th} layer weight matrix, A is the adjacency matrix and $\boldsymbol{\sigma}$ is a non-linear activation function.
But the above propagation rule has two major limitations -

- Matrix Multiplication with A means that feature vectors of all the neighboring nodes are added up for a node, but in this case, the node itself is not considered unless there is a self-loop.
- The matrix A is not normalized and thus the scale of feature vectors will completely change after multiplying with A.

To address the above-mentioned limitations, below mentioned modifications are needed-

- Adding Identity matrix to A, which will in-turn enforce self loops in the graph.
- Normalizing A by multiplying it with D^{-1}, where D is the diagonal degree matrix. Further using symmetric normalization is better, so A is replaced with $D^{-½} A D^{-½}$.8

The modified propagation rule is-

$$f(H^{(l)}, A) = \boldsymbol{\sigma}(\hat{D}^{-½} A \hat{D}^{-½} H^{(l)} W^{(l)})$$

Where $\hat{A} = A + I$ and $\hat{D}$ is diagonal degree matrix of $\hat{A}$.

References

1. Our Implementation of Simple Graph Convolution
2. B. Jiang, Z. Zhang, D. Lin, J. Tang and B. Luo, "Semi-supervised learning with graph learning-convolutional networks", *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, pp. 11 313-11 320, 2019.
3. Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr, Christopher Fifty, Tao Yu, and Kilian Q Weinberger. Simplifying graph convolutional networks. arXiv preprint arXiv:1902.07153, 2019.
4. https://tkipf.github.io/graph-convolutional-networks/
5. https://towardsdatascience.com/reverse-engineering-graph-convolutional-networks-dd78d4682ea1
6. https://arxiv.org/pdf/1903.11179.pdf