Graph Convolutional Networks

Pulkit Joshi

Sahil Harish Batra

Topics

1. Introduction
2. Spectral Graph theory
3. Simple Graph Convolution
4. Graph Convolutional Network

Introduction

- Many real world datasets are in the form of graphs. For eg - In Social Networks like facebook network users can be considered as vertices of a graph and they have an edge between them if they are friends, chemical compounds can be represented as graphs, in world wide web, web pages can be considered as vertices of a graph and there is edge between u and v vertices if there is a link of page v on page u.

- There are many more real-life examples where graphs are used to represent data and this shows that graph is indeed a very important data structure which has many practical applications.

Graph and its properties

- Graph is a non-linear data structure consisting of finite number of nodes (or vertices) and edges that connect two nodes, where edges can be directed or undirected.
- Let us consider L as the finite set of labels for vertices and edges.
- A graph is usually represented by an adjacency matrix.
- Adjacency matrix A is a N x N matrix, where N is the number of nodes in graph, with A(i, j) = 1 if there is an edge from vertex i to j, and A(i, j) = 0 if there is no edge from vertex i to j. For weighted graph, A(i, j) = weight of edge from i to j.

Images vs Graphs

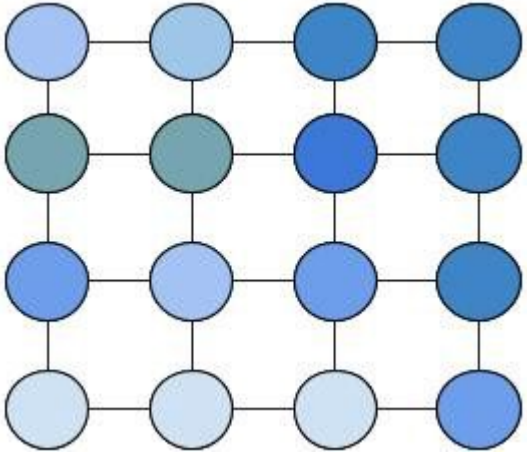
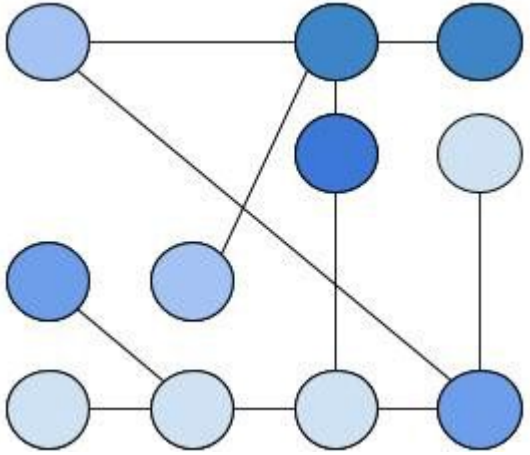


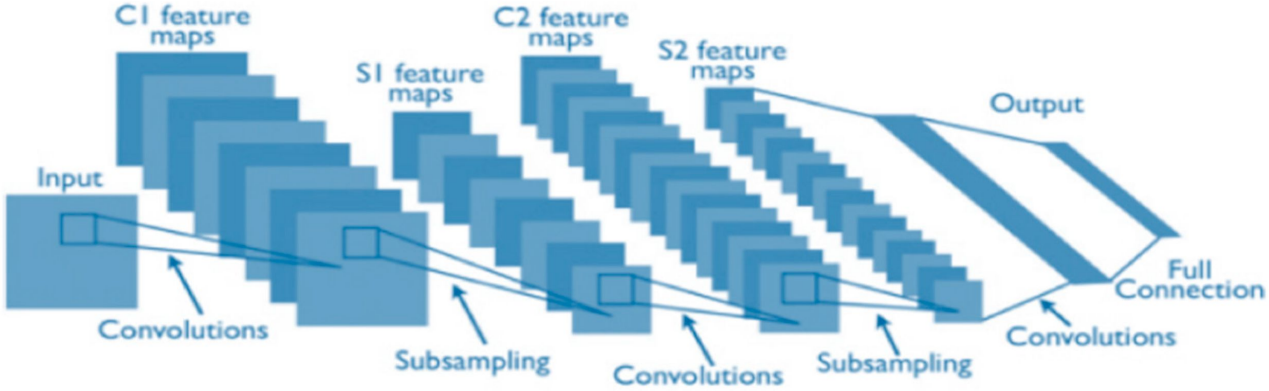
Image looks something like this (node are pixels)



Graphs look like this

Convolutional Neural Networks

To understand GCN, we first need to understand neural networks and convolutional neural networks.



Spectral Graph Theory

1. Matrices of Graph (Adjacency matrix, weight matrix and graph laplacian)
2. Shift operator on Graphs
3. Graph Fourier Transform

Spectral Graph Theory

In mathematics, **spectral graph theory** is the study of the properties of a **graph** using characteristic polynomial, eigenvalues, and eigenvectors of matrices associated with the **graph**, such as its adjacency matrix or Laplacian matrix.

- Adjacency matrix and normalization
- Graph laplacian
- Shift operator on graphs
- Graph fourier transform
- Output in fourier domain

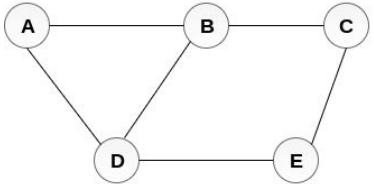
Adjacency matrix

- You have sensor network as shown in **undirected graph**
- Suppose you want output at a and its neighbours:

$$y = x_a + x_b + x_d$$

- This can be written as with the help of Adjacency matrix

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



Undirected Graph

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

Adjacency Matrix

॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

Weighted and Normalised Matrix

- Previously we saw:
  $Y = x + Ax$

- If graph is weighted, we get:

  $Y = x + Wx$
  where W is weight matrix of our graph

- Further we can have normalised matrix, which gives

  $y = ½(x + D^{-1} Wx)$
  where D is the diagonal degree matrix of graph.

Graph laplacian

- Graph laplacian operator is represented as L and its equation is defined below -

$$L = D - W$$

  Where D is the diagonal degree matrix of graph and W is the adjacency matrix of graph.

- One important thing to note here that for undirected graphs, the relations between nodes is symmetric and thus these matrices are also symmetric while for directed graphs there is no symmetric relationship.

Shift operator on Graphs

- Contrary to time signals, for graphs shifting would mean that a signal at node x would move to all its neighbouring nodes on a single shift.
- Thus shift for a graph is defined as -

$$X_{shifted} = Ax, \quad \text{where A is the adjacency matrix.}$$

- Instead of using A, W or any other matrix, we define S as our shift operator.

$$X_{shifted} = SX$$

Defining a system using shift operator

We define a system on the graph as input-output relation using shift operator S as -

$$\mathbf{y = H(S)\, x}$$

We can define following two properties of this system -

- Linear - If $\mathbf{H(s)(a_1x_1 + a_2x_2) = a_1y_1 + a_2y_2}$, where $y_1 = H(S)x_1$ and $y_2 = H(s)x_2$
- Shift invariant - if **H(s)(Sx) = S(H(s)x)**

Graph Fourier Transform (GFT)

- Laplacian can be decomposed into **orthonormal matrix of eigenvectors** and **diagonal matrix of eigenvalues**, i.e.

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

- Thus we can define Graph Fourier Transform (GFT) as -

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

- Similarly, inverse GFT can be defined as -

$$x = UX$$

Output in Fourier domain

- We can rewrite previously defined equation, y = H(S) x, as below -
- The above equation can be further $y = \sum_{m=0}^{M-1} h_m L^m x$ inverse GFT as below -

$$y = U H \Lambda U^{-1} x$$

- Then taking GFT on both sides, we get -

$$Y = H \Lambda X$$

Simple Graph Convolution

1. Graph Convolution Network: A Glimpse
2. Simple Graph Convolution
3. Implementation and results of SGC

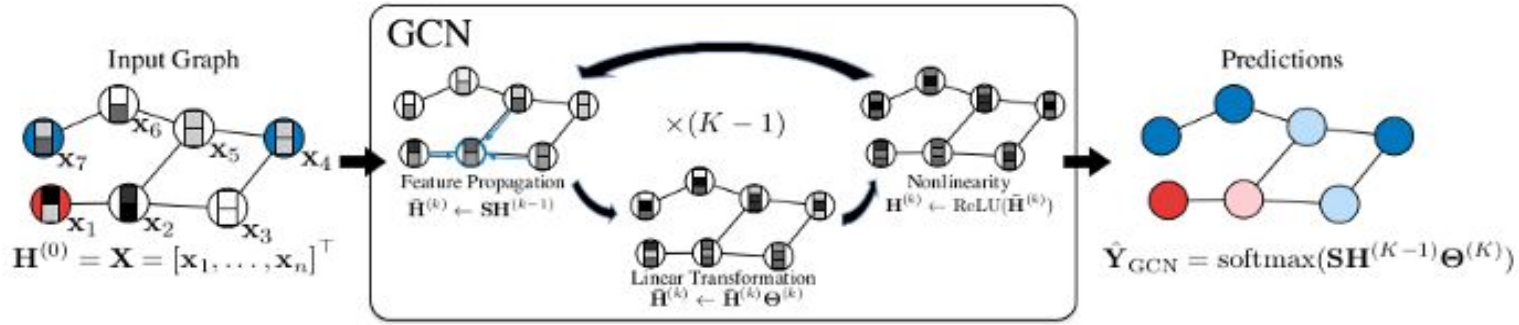
Simple Graph Convolution

Before diving deep into GCNs, we would first start with a simple linear model of GCNs known as Simple Graph Convolution (SGC), which gives us almost **similar results** as GCN and is faster than **FastGCN**.

We will go through the following topics to understand SGC -

- Graph Convolution Network
- Simple Graph Convolution
- Implementation and Results of SGC

Graph Convolution Network



Graph Convolution Network : A Glimpse

1. Feature Propagation

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

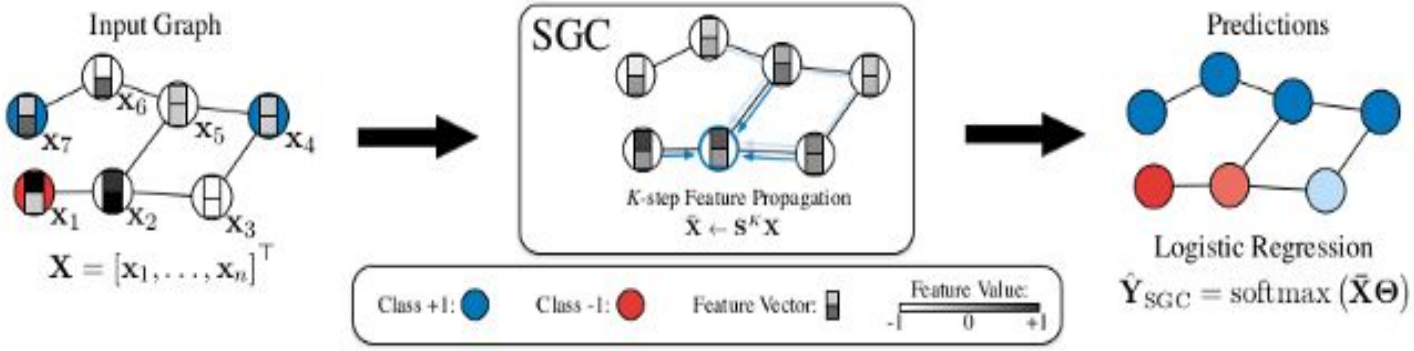
2. Feature Transformation and non linear transition

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

3. Classification

$$Y_{GCN} = \text{softmax}(SH^{(k-1)}\theta^{(k-1)})$$

Simple Graph Convolution



Simple Graph Convolution

Now we will linearise the GCN model to convert it into Simple Graph Convolution and it involves three steps -

1. Linearization
2. Classification
3. Feature Engineering

Linearization

- Removing the Non linear part,

$$H'^{(k)} = ReLU(H^{(k)}\theta^{(k)}) \text{ gets converted to } H'^{(k)} = H^{(k)}\theta^{(k)}$$

- As we know $H^k = SH^{(k-1)}$ and $H^1 = SX\theta^1$, we get the below equation -

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

Classification

Now that we have linearised the layers, we would feed the output of final layer into a softmax classifier.

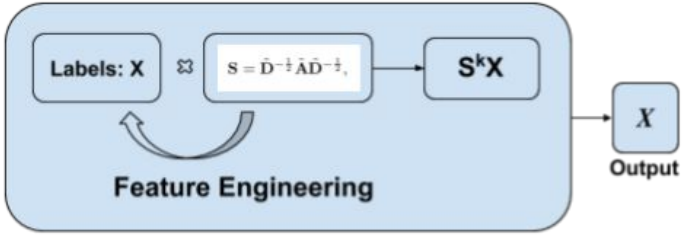
$$Y_{gcn} = \text{softmax}\left(S...SSX\Theta^{(1)}\Theta^{(2)}\Theta^{(3)}....\Theta^{(k)} \right)$$

$$Y_{gcn} = \text{softmax}\left(S^{k}X\Theta \right)$$

Feature Engineering

From the equation of Y_{gcn}, we can see that our input is being multiplied by matrix S^k and thus we can replace S^kX by $\boldsymbol{X}$, which reduces our model to a simple linear classification with feature engineering step.

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



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%.**

Graph Convolution Network

1. Graph Convolutional Networks
2. Graph Convolutions
3. Propagation rule in GCN

Graph Convolutional Networks

Graph Convolutional Networks are more generalised form of CNNs that operate on graphs.

Graph convolutions

- Using the concepts of eigen decomposition of Laplacian, and GFT, we define graph convolution over a signal as below -

  $g * x = U ((U^T g).(U^T x)) = UGU^T x,$

  Where G is a diagonal matrix equivalent to transform of g.

- Further using approximations of kth order polynomial we get -

$$UGU^T x \approx U \left(\sum_{i=0}^{k} \theta_i \Lambda^i \right) U^i x$$

Graph convolutions

- Now after using affine approximation(k = 1), $\theta_0 = 2\theta$ and $\theta_1 = -\theta$, graph convolution can be defined as -

  $g * x = \theta(I + D^{-1/2}AD^{-½}) x$

- Further normalization of $I + D^{-½}AD^{-½}$ gives $D'^{-½}A'D'^{-½}$ and thus finally graph convolution is defined as -

  $g * x = \theta(D'^{-½}A'D'^{-½}) x,$

  where A' = A + I and D' is diagonal matrix of graph represented by A' .

Semi-Supervised Learning Using Graph Convolutional Networks

Input and Output of GCN

- Input to any Graph Convolutional network is -
    - A NxD feature matrix X, where N is the number of nodes in graph and D is number of input features, in which i^{th} row describes the features of ith node of the graph
    - Adjacency matrix of graph A
- After going through the layers of GCN, a node level output matrix is produced which is a NxF matrix, where F is number of output features per node.
- It is important to note that GCN produces node-level output and to get a graph level output we would need to perform a global pooling like operation.

Propagation rule for GCN

Each layer of GCN can be written as below function, where $H^{(0)}$ is the feature matrix X as described in previous slide and $H^{(L)}$ is the node-level output Z , where L is number of layers in GCN and A is adjacency matrix of graph. The function *f* is chosen according to the requirement of our model.

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

Let us consider a simple layer-wise propagation rule by choosing function *f* as depicted below -

where $W^{(l)}$ is weight matrix fc $f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right)$ function.

The above discussed model has two 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.

The above discussed limitations can be resolved by doing the following modifications -

- 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 $D^{-½}A\,D^{-½}$.

Modified Propagation rule

After considering the above discussed limitations and applying symmetric normalization, i.e. using

$D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ instead of $D^{-1}A$, the final propagation rule is depicted below -

w

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

with $\hat{A} = A + I$, where I is the identity matrix and $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$.