MTQ Graph Theory

Social Influence Prediction with Deep Learning
Graph Convolutional Networks

Reference paper - DeepInf: Social Influence Prediction with Deep Learning
Jiezhong Qiu , Jian Tang, Hao Ma, Yuxiao Dong , Kuansan Wang, and Jie Tang
Link - https://arxiv.org/pdf/1807.05560

Professor: B.S. Panda
Ayush Chaurasia
Siddhant

What is a Graph Convolutional Network?

GCNs are a very powerful neural network architecture for machine learning on graphs.

More formally, a *graph convolutional network (GCN)* is a neural network that operates on graphs. Given a graph $G = (V, E)$, a GCN takes as input:

- an input feature matrix $N \times F^0$ feature matrix, $\boldsymbol{X}$, where N is the number of nodes and F^0 is the number of input features for each node, and
- an $N \times N$ matrix representation of the graph structure such as the adjacency matrix $\boldsymbol{A}$ of G.

A hidden layer in the GCN can thus be written as $\boldsymbol{H}^i = f(\boldsymbol{H}^{i-1}, \boldsymbol{A}))$ where $\boldsymbol{H}^0 = \boldsymbol{X}$ and f is a propagation. Each layer $\boldsymbol{H}^i$ corresponds to an $N \times F^i$ feature matrix where each row is a feature representation of a node. At each layer, these features are aggregated to form the next layer's features using the propagation rule f. In this way, features become increasingly more abstract at each consecutive layer. In this framework, variants of GCN differ only in the choice of propagation rule f.

Hidden Layer

A hidden layer in the GCN can thus be written as $\boldsymbol{H}^{l} = f(\boldsymbol{H}^{i-1}, \boldsymbol{A}))$ where $\boldsymbol{H}^{0} = \boldsymbol{X}$ and f is a propagation. Each layer $\boldsymbol{H}^{l}$ corresponds to an $N \times F^{l}$ feature matrix where each row is a feature representation of a node. At each layer, these features are aggregated to form the next layer's features using the propagation rule f. In this way, features become increasingly more abstract at each consecutive layer. In this framework, variants of GCN differ only in the choice of propagation rule f.

Propagation Rule Example

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

where $\boldsymbol{W}^{l}$ is the weight matrix for layer i and σ is a non-linear activation function such as the ReLU function. The weight matrix has dimensions $F^{l} \times F^{i+1}$; in other words the size of the second dimension of the weight matrix determines the number of features at the next layer.

Problems

1. The aggregated representation of a node does not include its own features. The representation is an aggregate of the features of neighbor nodes, so only nodes that has a self-loop will include their own features in the aggregate.
2. Nodes with large degrees will have large values in their feature representation while nodes with small degrees will have small values. This can cause vanishing or exploding gradients, but is also problematic for stochastic gradient descent algorithms which are typically used to train such networks and are sensitive to the scale (or range of values) of each of the input features.

1. **Self-Loops**

```
In [4]: I = np.matrix(np.eye(A.shape[0]))
        I


Out[4]: matrix([
            [1., 0., 0., 0.],
            [0., 1., 0., 0.],
            [0., 0., 1., 0.],
            [0., 0., 0., 1.]
        ])


In [8]: A_hat = A + I
        A_hat * X
Out[8]: matrix([
            [ 1., -1.],
            [ 6., -6.],
            [ 3., -3.],
            [ 5., -5.]])
```

2. **Normalizing the Feature Representation**

```
In [9]: D = np.array(np.sum(A, axis=0))[0]
        D = np.matrix(np.diag(D))
        D
Out[9]: matrix([
            [1., 0., 0., 0.],
            [0., 2., 0., 0.],
            [0., 0., 2., 0.],
            [0., 0., 0., 1.]
        ])
```

Final Output

- Self-Loops
- Normalizing the Feature Representation
- Activation Function
- Weight matrix

```
In [51]: W = np.matrix([
             [1, -1],
             [-1, 1]
         ])
         relu(D_hat**-1 * A_hat * X * W)
Out[51]: matrix([[1., 0.],
         [4., 0.],
         [2., 0.],
         [5., 0.]])
```

Spectral Graph Convolutions

A recent paper by Kipf and Welling proposes fast approximate spectral graph convolutions using a spectral propagation rule:

$$f(X, A) = \sigma(\mathbf{D}^{-0.5}\hat{\mathbf{A}}\mathbf{D}^{-0.5}\mathbf{X}\mathbf{W})$$

The spectral rule differs only in the choice of aggregate function. Although it is somewhat similar to the mean rule in that it normalizes the aggregate using the degree matrix ***D*** raised to a negative power, the normalization is asymmetric.

$$f(X, A) = \sigma(\mathbf{D}^{-0.5}\hat{\mathbf{A}}\mathbf{D}^{-0.5}\mathbf{X}\mathbf{W})$$

$$\begin{aligned}
aggregate(\mathbf{A}, \mathbf{X})_i &= \mathbf{D}^{-0.5}\mathbf{A}_i\mathbf{D}^{-0.5}\mathbf{X} \\
&= \sum_{k=1}^{N} D_{i,k}^{-0.5} \sum_{j=1}^{N} A_{i,j} \sum_{l=1}^{N} D_{j,l}^{-0.5}\mathbf{X}_j \\
&= \sum_{j=1}^{N} D_{i,i}^{-0.5} A_{i,j} D_{j,j}^{-0.5}\mathbf{X}_j \\
&= \sum_{j=1}^{N} \frac{1}{D_{i,i}^{0.5}} A_{i,j} \frac{1}{D_{j,j}^{0.5}}\mathbf{X}_j
\end{aligned}$$

Training Procedure

1. Perform forward propagation through the GCN.
2. Apply the sigmoid function row-wise on the last layer in the GCN.
3. Compute the cross entropy loss on known node labels.
4. Backpropagate the loss and update the weight matrices W in each layer.

https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-62acf5b143d0