Graph Convolutional Networks (GCNs) are to learn a function of signals/features on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, which takes as input:

- A feature description xi 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).

Every neural network layer can then be written as a non-linear function

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

with $H^0 = X$ and $H^l = Z$ (or z for graph-level outputs), L being being the number of layers. The specific models then differ only in how $f(\cdot, \cdot)$ is chosen and parameterized.

1. **Spacial Graph ConvNets**

As an example, let's consider the following very simple form of a layer-wise propagation rule:

$$f(H^l, A) = \sigma(AH^lW^l), \tag{2}$$

where W^l is a weight matrix for the l-th neural network layer and $\sigma(\cdot)$ is a non-linear activation function like the ReLU.
Multiplication with A means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself (unless there are self-loops in the graph). We can "fix" this by enforcing self-loops in the graph: we simply add the identity matrix to A.
Another limitation of Eq. (2) is that A is typically not normalized and therefore the multiplication with A will completely change the scale of the feature vectors (we can understand that by looking at the eigenvalues of A). Normalizing A such that all rows sum to one, i.e. $D^{-1}A$, where D is the diagonal node degree matrix, gets rid of this problem. Multiplying with $D^{-1}A$ now corresponds to taking the average of neighboring node features. In practice, dynamics get more interesting when we use a symmetric normalization, i.e. $D^{-1/2}AD^{-1/2}$ (as this no longer amounts to mere averaging of neighboring nodes). Combining these two tricks, we essentially arrive at the propagation rule introduced in [1]:

$$f(H^l, A) = \sigma(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H^lW^l), \tag{3}$$

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

2. **Spectral Graph ConvNets**

We define the graph (normalized) Laplacian as

$$\Delta = I - D^{-1/2}AD^{-1/2}. \tag{4}$$

The Laplacian is interpreted as the measurement of smoothness of graph, in other words, the difference between the local value node h_i and its neighborhood average value of node h_j's.
The following is the eigen-decomposition of graph Laplacian,

$$\Delta = \Phi^T\Lambda\Phi, \tag{5}$$

where Φ contains column vectors, or Lap eigenvectors ϕ_i to ϕ_n, each of size $n \times 1$, and those are also called **Fourier functions**. And Fourier functions form an orthonormal basis, $\Phi = [\phi_1, \cdots, \phi_n]$ and $\Phi^T\Phi = I$. Λ is a diagonal matrix with Laplacian eigenvalues, and on the diagonal are λ_1 to λ_n.

The Fourier transform is basically projecting a function h on the Fourier functions, and the result are the coefficients of the Fourier series,

$$\mathcal{F}(h) = \Phi^T h = \hat{h}. \tag{6}$$

Inverse fourier transform gives

$$\mathcal{F}^{-1}(\hat{h}) = \Phi\hat{h} = \Phi\Phi^T h = h \tag{7}$$

Fourier transform of the convolution of two functions is the pointwise product of their Fourier transforms.

We define a graph spectral convolutional layer such that given layer h^l, the activation of the next layer is:

$$h^{l+1} = \sigma(w^l * h^l), \tag{8}$$

where σ represents a nonlinear activation and w^l is a spatial filter. $w^l * h^l$ is equivalent to $\hat{w}^l(\Delta)h^l$, where $\hat{w}^l$ represents a spectral filter and Δ is the Laplacian. We can further decompose it into $\Phi\hat{w}^l(\Lambda)\Phi^T h^l$, where Φ is the eigenvector matrix and Λ is the eigenvalues. This yields the final activation equation as below.

$$h^{l+1} = \sigma(\Phi\hat{w}^l(\Lambda)\Phi^T h^l) \tag{9}$$

The objective is to learn the spectral filter $\hat{w}^l$ using backpropagation instead of hand crafting.

References

[1] Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks[J]. arXiv preprint arXiv:1609.02907, 2016.