Machine Learning with Graphs: Representation learning 2/3 - Graph Neural Networks

Arlei Silva

Spring 2022

Graph Convolution

A *convolution* is an operation that returns a function $g * f$, given input functions f and g. In the case of graph convolutions, one of the functions is a graph signal $\mathbf{f} \in \mathbb{R}^n$—we will write as a vector. A key tool in defining graph convolutions is the *Graph Fourier Transform (GFT)* $\hat{\mathbf{f}} = U^T\mathbf{f}$, where $L = I - D^{-1/2}AD^{-1/2} = U\Lambda U^T$. As we have seen earlier, the eigenvectors of the Laplacian form a basis for graph signals and the value of the associated eigenvalue provides a notion of frequency for a basis vector.

The *spectral convolution* on a graph is defined as follows:

$$g_\theta * \mathbf{f} = U g_\theta(\Lambda) U^T \mathbf{f}$$

where $g_\theta(\Lambda) = \mathbf{diag}(g_\theta(\lambda_1), g_\theta(\lambda_2), \dots g_\theta(\lambda_n))$

Let $\hat{\mathbf{f}} = [\hat{\mathbf{f}}(\lambda_1), \dots \hat{\mathbf{f}}(\lambda_n)]$. Then, we can write each entry of $g_\theta * \mathbf{f}$ as:

$$g_\theta * \mathbf{f}[v] = \sum_{\ell=1}^{n} \hat{\mathbf{f}}(\lambda_\ell) g_\theta(\lambda_\ell) \mathbf{u}_\ell[v]$$

We call g_θ a *filtering function*. As an example, consider the following optimization problem with the goal of learning a vector $\mathbf{f}$ such that it approximates another vector $\mathbf{y}$ while also being smooth over the graph:

$$\mathbf{f}* = \arg\min_{\mathbf{f}} ||\mathbf{y} - \mathbf{f}||_2^2 + c\mathbf{f}^T L \mathbf{f}$$

where c is a constant. We can think of $\mathbf{y}$ and $\mathbf{f}$ as noisy and de-noised labels based on the graph topology, respectively. We minimize the objective by setting the derivative to zero:

$$\frac{\partial}{\partial \mathbf{f}} ||\mathbf{y} - \mathbf{f}||_2^2 + c\mathbf{f}^T L \mathbf{f} = 2(\mathbf{f} - \mathbf{y}) + 2cL\mathbf{f} = 0$$

It follows that:

$$
\begin{aligned}
(I + cL)\mathbf{f} &= \mathbf{y} \\
(UU^T + cU\Lambda U^T)\mathbf{f} &= \mathbf{y} \\
U(I + c\Lambda)U^T\mathbf{f} &= \mathbf{y} \\
\mathbf{f} = U(I + c\Lambda)^{-1}U^T\mathbf{y}
\end{aligned}
$$

Minimizing our objective is equivalent to applying a filter $g_\theta(\lambda_\ell) = 1/(1 + c\lambda_\ell)$ to the noisy signal $\mathbf{y}$. Intuitively, this filter reduces the importance high-frequency components—in the graph topology—from $\mathbf{y}$. If $c = 0$, then $\mathbf{f} = \mathbf{y}$. On the other hand, as $c \to \infty$, $\mathbf{f} = 0_n$ becomes a minima of the objective.

We will apply graph convolutions for learning problems by fitting the parameters of filtering function $g_\theta(\lambda_\ell) = \theta_\ell$. As an example, consider that our goal is to approximate a ground truth vector $\mathbf{y} \in \mathbb{R}^n$:

$$
\arg\min_\theta ||\mathbf{y} - g_\theta * \mathbf{f}||_2^2
$$

Then, by letting $\hat{\mathbf{y}} = U^T\mathbf{y}$, we get the optimal filter:

$$
g_\theta(\lambda_\ell) = \frac{\hat{\mathbf{y}}(\lambda_\ell)}{\hat{\mathbf{f}}(\lambda_\ell)} = \theta_\ell
$$

The above example shows that graph filters are quite flexible. However, notice that g_θ has n parameters to be learned, which might be infeasible. Instead, we can fix the number of parameters to $k + 1$ by assuming that $g_\theta(\lambda_\ell) = \sum_{k=0}^{K} \theta_k \lambda_\ell^k$—i.e. it is a polynomial of λ_ℓ. As result, we get a different form for the convolution:

$$
g_\theta * \mathbf{f} = U(\sum_{k=0}^{K} \theta_k \Lambda^k)U^t\mathbf{f} = \sum_{k=0}^{K} \theta_k L^k \mathbf{f}
$$

An interesting property of the above formulation is that powers of the Laplacian matrix are localized in the graph. The value of L_{ij}^k is zero if there is no path between nodes i and j in the graph. However, a downside is that the Laplacian powers are not orthogonal to each other. Instead, we can apply *Chebyshev polynomials* to describe our filtering function:

$$
g_\theta * \mathbf{f} = U(\sum_{k=0}^{K} \theta_k T_k(\Lambda^k))U^t\mathbf{f} = \sum_{k=0}^{K} \theta_k T_k(\tilde{L})\mathbf{f}
$$

Chebyshev polynomials (of the first kind) are defined as follows:

$$
\begin{aligned}
T_k(x) =& 2xT_{k-1}(x) - T_{k-2}(x) \\
T_0(x) =& 1 \\
T_1(x) =& x
\end{aligned}
$$

where we assume $x \in [-1, 1]$.

We approximate a function $f(x)$ using Chebyshev polynomials as follows:

$$f(x) \approx \sum_{k=0}^{\infty} c_k T_k(x)$$

In practice, we apply a small number of polynomials K to approximate $f(x)$. Our goal is to apply these polynomials to approximate $g_\theta(\lambda_\ell)$. First, we have to scale the entries of Λ within the range $[-1, 1]$:

$$\tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - I$$

Then, we get:

$$g_\theta(\tilde{\Lambda}) = \sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})$$

Now we can apply the approximation in the graph convolution:

$$g_\theta * \mathbf{f} = U(\sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda}))U^T \mathbf{f} = \sum_{k=0}^{K} \theta_k T_k(\tilde{L})\mathbf{f}$$

where:

$$\tilde{L} = \frac{2L}{\lambda_{max}} - I$$

The form of the Chebyshev polynomial applied to the Laplacian is as expected, $T_0(\tilde{L}) = I$, $T_1(\tilde{L}) = \tilde{L}$, and $T_k(\tilde{L}) = 2\tilde{L}T_{k-1}(\tilde{L}) - T_{k-2}(\tilde{L})$.

The *Graph Convolutional Network (GCN)* filter, applies the above formulation with $k = 1$:

$$g_\theta * \mathbf{f} \approx \theta_0 T_0(\tilde{L})\mathbf{f} + \theta_1 T_1(\tilde{L})\mathbf{f} \approx \theta_0 \mathbf{f} + \theta_1 (L - I)\mathbf{f} \approx \theta_0 \mathbf{f} - \theta_1 D^{-1/2} A D^{-1/2} \mathbf{f}$$

where we have assumed $\lambda_{max} = 2$ (upper bound) and thus $\tilde{L} = L - I$.

In fact, the number of parameters is reduced even further by setting $\theta = \theta_0 = -\theta_1$:

$$g_\theta * \mathbf{f} \approx \theta(I + D^{-1/2} A D^{-1/2})\mathbf{f}$$

There is still a minor issue with the above expression. In case we want to apply this convolution operator, repeatedly—as in multiple *layers*—the norm of the resulting vector might become a problem. More specifically, given a vector $\mathbf{x}$, we know that:

$$\max_{\mathbf{x}} \frac{||B\mathbf{x}||}{||\mathbf{x}||} = \lambda_{max}(B)$$

Thus, we can apply a *renormalization trick* to the matrix to keep the norm of the resulting vector constant. Let $\tilde{A} = A + I$ and $\tilde{D}$ be such that $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ and $\tilde{D}_{ij} = 0$ for $i \neq j$. We can show that $\lambda_{max}(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}) = 1$. So, we write:

$$g_\theta * \mathbf{f} \approx \theta \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} \mathbf{f}$$

We can generalize GCN filters to the case of D-dimensional channels (or signals) $X \in \mathbb{R}^{n \times D}$. Let $Z \in \mathbb{R}^{n \times h}$ be an h-dimensional output of the convolution. Then, we can define the graph convolution as:

$$Z = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X \Theta$$

where $\Theta \in \mathbb{R}^{D \times h}$ are parameters to be learned. It might be easier to look at each row of Z:

$$Z[i] = \sum_{v_j \in N(v_i) \cup \{v_i\}} [\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2}]_{ij} X[j] \Theta$$

References

[1] William L Hamilton. *Graph representation learning*. Morgan & Claypool, 2020.

[2] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[3] Yao Ma and Jiliang Tang. *Deep learning on graphs*. Cambridge University Press, 2021.