A unified deep learning formalism for processing graph signals

Myriam Bontonou* Carlos Lassance* Jean-Charles Vialatte† Vincent Gripon*

1 Introduction

Deep learning networks outperform any other algorithms in many machine learning challenges. These networks rely on an assembly of layers. Each layer composes a linear function, whose coefficients are parameters to be learned, and a non linear function. The network is trained end-to-end, using a variant of gradient descent on a criterion to be optimized. In images, a major factor explaining their performances is the ability of a layer to learn which shape features should be extracted in images. In such layers, a convolutional operator is implemented as a learnable linear function.

Two features distinguish the convolutional layer from a fully connected layer: a- the same parameters are used on all local portions of the input, b- the outputs of the network depend only on a local portion of the input. From these two characteristics arises an essential property of the convolutional layers: the number of parameters is independent of the size of the input.

This convolutional layer only makes sense when it is applied on inputs supported on a discrete Euclidean space (e.g: images, sounds). However, so many real-world signals are defined on more complex topologies (point clouds, geo-localized sensor networks, or functional activity in the brain). Thus, in recent years, a variety of works have emerged to extend the performances of convolutional layers to more diverse signals, and in particular, to signals defined on graphs.

In this study, we review some of the major deep learning models designed to exploit the underlying graph structure of signals. We express them in a unified formalism, giving them a new and comparative reading.

2 Unified formalism

Many deep learning approaches have been proposed to process graph signals. Each approach proposes a new linear function. As authors introduce their own formalism, it can be difficult to compare models. We propose a unified formalism to express them. Indices are written in lower case, and the size of their sets in upper case. In deep learning, data is divided into B batches. Given I input neurons and P channels, the input of a layer is a tensor $\mathbf{X} \in \mathbb{R}^{\mathrm{B}\times\mathrm{P}\times\mathrm{I}}$. Similarly, given J output neurons and Q features, the output $g(\mathbf{X}) \in \mathbb{R}^{\mathrm{B}\times\mathrm{Q}\times\mathrm{J}}$. Let's introduce a first simple example. Considering only one batch, $\mathbf{X} \in \mathbb{R}^{\mathrm{P}\times\mathrm{I}}$, a feature q and an output neuron j, we store the coefficients of the linear function g in $\mathbf{W} \in \mathbb{R}^{\mathrm{Q}\times\mathrm{P}\times\mathrm{J}\times\mathrm{I}}$:

$$g(\mathbf{X})_{qj} = \sum_{p=1}^{\mathrm{P}} \sum_{i=1}^{\mathrm{I}} \mathbf{W}_{qpji}\mathbf{X}_{pi} \ . \tag{2.1}$$

The same coefficients may be used in multiple connections. It is therefore inefficient to store them all in $\mathbf{W}$. Subsequently, given K unique parameters, we propose to store them in $\theta \in \mathbb{R}^{\mathrm{K}}$. Now, we define an allocation tensor $\mathbf{S} \in \mathbb{R}^{\mathrm{K}\times\mathrm{J}\times\mathrm{I}}$ such that $\mathbf{W}_{qpji} = \sum_{k=1}^{\mathrm{K}} \theta_{qpk}\mathbf{S}_{kji}$. To simplify the notations, we omit the sums and by convention, we consider that there is a sum on an index when it is no longer present in the output variable. The formula above becomes: $\mathbf{W}_{qpji} = \theta_{qpk}\mathbf{S}_{kji}$. By reintroducing the batch index, we obtain the following formalism:

$$g(\mathbf{X}) = \widehat{\mathbf{\Theta}\mathbf{S}\mathbf{X}} \text{ where } \left\{ \begin{array}{c} \mathbf{W}_{qpji} = \mathbf{\Theta}_{qpk}\mathbf{S}_{kji} \\ g(\mathbf{X})_{bqj} = \mathbf{W}_{qpji}\mathbf{X}_{bpi} \end{array} \right\} \tag{2.2}$$

3 Results and Discussion

The formalism $g(\mathbf{X}) = \widehat{\mathbf{\Theta}\mathbf{S}\mathbf{X}}$ distinguishes the roles of $\mathbf{\Theta}$ and $\mathbf{S}$. $\mathbf{\Theta}$ stores the parameters. $\mathbf{S}$ contains the parameter sharing scheme. In this section, we express four major deep learning models in graph signal processing as well as a fully connected (FC) layer and a conventional convolutional layer. Consider a graph $\mathcal{G} = \langle V, \mathbf{A} \rangle$, where $V = \{1 \dots N\}$ is a set of vertices, $\mathbf{A} \in \mathbb{R}^{N\times N}$ an adjacency matrix, $\mathbf{D}$ a degree matrix and $\mathbf{L} = \mathbf{D} - \mathbf{A}$ a Laplacian matrix. A signal with P features, supported on a graph, is represented by $\mathbf{X} \in \mathbb{R}^{N\times\mathrm{P}}$. This signal can be an input of the linear function described in the unified formalism.

Fully connected layer: A FC layer is represented by I input neurons, J output neurons, 1 channel and 1 feature. The number of parameters K is equal to the number of possible connections between the input neurons and the output neurons, $\mathrm{K} = \mathrm{I}\times\mathrm{J}$. In that case, $g(\mathbf{X})_{b1j} = \mathbf{\Theta}_{11k}\mathbf{S}_{kji}\mathbf{X}_{b1i}$. $\mathbf{\Theta}_{11k} \in \mathbb{R}^{1\times1\times\mathrm{IJ}}$ is indeed a vector. Given neurons i and j, $\mathbf{S}[:, j, i]$ is a one-hot

*IMT Atlantique, MILA
†IMT Atlantique

vector, which selects the parameter associated to the connection between i and j. By squeezing the shape of tensors, we get the usual formula describing a FC layer: $g(\mathbf{X})_{bj} = \mathbf{W}_{ji}\mathbf{X}_{bi}$.

Convolutional layer: Parameters depend on the inputs channel p and feature map q. $\mathbf{\Theta}[q, p, :] \in \mathbb{R}^{\mathrm{K}}$ contains the kernel weights for a given input channel p and feature map q. In a convolutional layer, the convolution matrix is Toeplitz. Each diagonal, from the top left to the bottom right, contains either the same parameter or a zero. In the ternary representation, $\mathbf{S}$ maintains the same structure. Indeed, in the 1D case, $\mathbf{S}[k, :, :] \in \mathbb{R}^{\mathrm{I}\times\mathrm{J}}$ is full of zeros, except on one diagonal where the coefficient is 1. For each parameter $\mathbf{\Theta}[q, p, k]$, a different diagonal of $\mathbf{S}$ would contain 1.

ChebNet ChebNet [1] is inspired from spectral graph theory, where the Discrete Fourier Transform (DFT) is expanded to graph signals. Let c be the order of a Chebyshev polynomial T_c. The linear function g becomes:

$$g(\mathbf{X})_q = \sum_{p=1}^{\mathrm{P}} \mathbf{W}_{qp}(\mathbf{L})\mathbf{x}_p \ . \tag{3.3}$$

Given a constant C, $\lambda_{\max}$ the largest eigenvalue of $\mathbf{L}$, the identity matrix $\mathbf{I}_N$, $T_c(\frac{\lambda_{\max}\mathbf{L}}{2} - \mathbf{I}_N) \in \mathbb{R}^{N\times N}$ and $\theta \in \mathbb{R}^{\mathrm{C}}$ the learnable parameters, $\mathbf{W}_{qp} \in \mathbb{R}^{N\times N}$ is defined as: $\mathbf{W}_{qp}(\mathbf{L}) = \sum_{c=0}^{\mathrm{C}-1} \theta_c \, T_c(\frac{\lambda_{\max}\mathbf{L}}{2} - \mathbf{I}_N)$.

Unified formalism: There are as many input neurons I as output neurons J as vertices in the graph I = J = N. Only one weight is computed for a couple (p,q), therefore K = 1. When $\mathbf{S} = \sum_{c=0}^{C-1} T_c(\frac{\lambda_{\max}(\mathbf{L})\mathbf{L}}{2} - \mathbf{I}_N)$, a Chebyshev filter is computed.

Graph Convolutional Network GCN [3] is a neural network using the following linear function g:

$$g(\mathbf{X}) = \tilde{\mathbf{A}}\ \mathbf{X}\ \mathbf{\Theta}\ , \tag{3.4}$$

where $\tilde{\mathbf{A}} = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$, $\hat{\mathbf{D}}$ degree matrix of $\hat{\mathbf{A}}$, $\mathbf{\Theta} \in \mathbb{R}^{\mathrm{P}\times\mathrm{Q}}$. $\tilde{\mathbf{A}}\mathbf{X}$ diffuses the signal on the graph. Now, the values of the signal on a vertex depend on the values of the neighboring vertex signals. $\mathbf{\Theta}$ learns several representations of the diffused signal. This model is close to ChebNet if we reformulate it as follows: $g(\mathbf{X}) = \sum_{c=0}^{C-1} T_c\left(\frac{\lambda_{\max}\mathbf{L}}{2} - \mathbf{I}_N\right)\ \mathbf{X}\ \mathbf{\Theta}$.

Unified formalism: Consequently, its expression is similar as ChebNet. I = J = N. K = 1. $\mathbf{S}$ corresponds to the normalized adjacency matrix.

Graph ATtention network GAT [4] learns parameters which measure the importance of the neighbors considering all vertices independently. Given a matrix $\mathbf{\Theta} \in \mathbb{R}^{\mathrm{P}\times\mathrm{Q}}$ and two neighbor vertices (i, j), a fully connected layer, represented by the vector $\mathbf{a} \in \mathbb{R}^{2\mathrm{Q}}$, learns the attention α_{ji} that j deserves from i. $\alpha_{ji} = softmax(e_{ji})$, where $e_{ji} = LeakyReLU(\mathbf{a}^T\ [\mathbf{X}_j\mathbf{\Theta}\ ||\ \mathbf{X}_i\mathbf{\Theta}])$, $||$ meaning concatenation. $\mathbf{\Theta}$ transforms input features into higher-level features, and $\mathbf{a}$ diffuses the signal. To exploit the structure of the graph, a mechanism called attention mask is set up: e_{ji} is zero if the vertex i is not connected to the vertex j. Finally, noting $\mathcal{R}(j)$ the set of vertices connected to the vertex j, we get the following linear function g:

$$g(\mathbf{X})_j = \sum_{i\in\mathcal{R}(j)} \alpha_{ji}\mathbf{X}_i\mathbf{\Theta}\ . \tag{3.5}$$

To stabilize learning, several linear functions g are computed independently, then concatenated or averaged.

Unified formalism: Similarly to GCN, I = J = N. To express a concatenated multiple attention heads layer, A ternary layers are concatenated. For each layer, as only one weight is computed for a couple (p,q), K = 1, and $\mathbf{S}$ contains the attention coefficients. To express an averaged multiple attention heads layer, A weights are computed for a couple (p,q), K = A, and $\mathbf{S}[k, :, :]$ contains the attention coefficients divided by A.

Topology Adaptative GCN TAGCN [2] makes the signal of a vertex dependent to the signals of its neighbors located at most C connections. It introduces the powers of the normalized adjacency matrix $\tilde{\mathbf{A}}$ in GCN:

$$g(\mathbf{X}) = \sum_{c=1}^{C} \tilde{\mathbf{A}}^{\,c}\ \mathbf{X}\ \mathbf{\Theta}_c\ . \tag{3.6}$$

Unified formalism: Similar to GCN, excepted that the weight sharing $\mathbf{S}[k, :, :]$ contains the normalized adjacency matrix $\tilde{\mathbf{A}}^{\,k}$. Plus, if at most C powers are computed, given a couple (p,q), C parameters have to be learned. So, K = C.

References

[1] M. Defferrard, X. Bresson, and P. Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering*, in Advances in Neural Information Processing Systems, 2016, pp. 3844–3852.

[2] J. Du, S. Zhang, G. Wu, J. M. Moura, and S. Kar, *Topology adaptive graph convolutional networks*, arXiv preprint arXiv:1710.10370, (2017).

[3] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, arXiv preprint arXiv:1609.02907, (2016).

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, arXiv preprint arXiv:1710.10903, (2017).