L. Stankovic and D. P. Mandic, "Understanding the Basis of Graph Convolutional Neural Networks via an Intuitive Matched Filtering Approach [Lecture Notes]," in *IEEE Signal Processing Magazine*, vol. 40, no. 2, pp. 155-165, March 2023, doi: 10.1109/ MSP.2022.3207304.

Understanding the Basis of Graph Convolutional Neural Networks via an Intuitive Matched Filtering Approach

Ljubiša Stanković, *Fellow, IEEE* and Danilo Mandic, *Fellow, IEEE*

I. Scope

Graph Convolutional Neural Networks (GCNN) are becoming a model of choice for learning on irregular domains; yet due to the black box nature of neural networks (NNs) their underlying principles are rarely examined in depth. To this end, we revisit the operation of GCNNs from first principles and show that their key component – the convolutional layer – effectively performs matched filtering of its inputs with a set of templates (filters, kernels) of interest. This serves as a vehicle to establish a compact matched filtering perspective of the whole convolution-activation-pooling chain, which allows for a theoretically well founded and physically meaningful insight into the overall operation of GCNNs. This is shown to help mitigate their interpretability and explainability issues, together with providing intuition for further developments, their applications, and education purposes. This Lecture Note is supported by an online Supplement Material which provides more detail on several aspects of GCNN operation and training.

II. Relevance

The success of deep learning (DL) and convolutional neural networks (CNN) has also highlighted that NN-based analysis of signals and images of large sizes poses a considerable challenge, as every input sample/pixel needs to be associated with one neuron at the input layer. A meaningful NN-based analysis requires at least one hidden layer; yet even for a simplest fully-connected (FC) hidden layer the number of weights increases exponentially with data volume – the so called Curse of Dimensionality. The dimensionality bottleneck may be partially mitigated by leveraging on the smooth nature of most physical data, whereby the neighboring signal samples (or image pixels) typically exhibit a degree of similarity. This allows us to describe signals/images in terms of their local characteristic features (patterns), and exploit such local information throughout the processing chain. In this way, a learning task boils down to a much more computationally efficient search for features (patterns) in data, in contrast to standard computationally hungry brute-force approaches. In the following, unless otherwise stated, we will use the term signal for both signals and images.

L. Stanković is with the University of Montenegro, Podgorica, Montenegro. D. P. Mandic is with Imperial College London, London, United Kingdom. This paper has downloadable Supplementary Material available at http://ieeexplore.ieee.org., which contains more details of the key underlying concepts, together with a step-by-step elaboration of back-propagation training of GCNNs. Contact ljubisa@ucg.ac.me for further questions about this work.

An important advantage of operating in the feature space is that this resolves the problem of position change of patterns in signals. Namely, if a certain signal feature is moved from its original temporal or spatial position, e.g. due to translation, a standard sample/pixel-based approach would assume that it is presented with a completely different set of samples/pixels; in contrast, a feature-based approach will look for this specific pattern of interest anywhere in the signal. This underpins the operation of CNNs which effectively perform a search for features in the analyzed signal; this is achieved in such a way that these features are invariant to changes in their positions [1], [2]. In this way, each "feature window" used in the convolution operation within CNNs (*convolution filter or convolution kernel*) becomes, through training, the best match to a specific feature within the analyzed signal. Such feature matching is performed over the whole signal, akin to a mathematical lens in search of some specific forms [3].

Irregular domains are conveniently modeled as graphs [4], [5], and for learning on such domains it is natural to consider graphs in conjunction with neural networks – the graph neural networks (GNNs) paradigm. Such an approach benefits from the universal approximation property of neural networks, pattern matching inherent to CNNs, and the ability of graphs to capture local relations and implicit complex couplings in data. Research on GNNs has emerged almost two decades ago [6], [7], with more recent developments centered around graph convolutional neural networks (GCNNs) [8], [9]. These have largely focused on learning aspects of CNNs [10], [11], and assume stationarity (via shift invariance of convolution) and compositionality (via downsampling or pooling) [5] – but without considering the wider context underpinning the operation of GNNs – a subject of this Lecture Note.

The aim of this Lecture Note is to revisit the operation of GCNNs in order to reveal the basic principles underpinning their functionality, and in this way help demystify their operation for a generally knowledgeable reader and for educational purposes. Upon establishing that the operation of convolutional layers in standard CNNs rests upon the classical signal processing concept of matched filtering, we introduce an analogous graph matched filtering framework for understanding the convolution-activation-pooling chain within GCNNs, which maintains intuition and physical interpretability through the whole GCNN operation. It is our hope that such a perspective will further help seamless migration of concepts and ideas from Signal Processing into the more empirical area of Neural Networks, as well as serving as a platform for extensions

and application opportunities in our data-hungry world. For enhanced intuition, physical insight into the operation of GNNs is provided through a step-by-step numerical example which also visualizes information propagation through GCNNs, thus illuminating all stages of GCNN operation and learning.

III. Prerequisites

This Lecture Note assumes a basic knowledge of Linear Algebra and Digital Signal Processing (DSP), and is supported by on-line Supplementary Material which contains more details of the key underlying concepts, together with a step-by-step elaboration of back-propagation training of GCNNs.

IV. Problem statement and solution

A. The matched filter and convolution

The use of a convolution filter/kernel underpins the operation of CNNs, yet the key question of how to justify the use of convolution to detect features in a signal remains largely unanswered – a subject of this work. To address this issue, recall that matched filtering is a widely used technique for the detection of the presence of a known template (feature), w, in an observed unknown noisy signal, x, and is achieved by cross-correlating the signal, x, with the template w, that is

$$y(n) = \sum_m x(n+m)w(m). \tag{1}$$

The maximum of this cross-correlation, $y(n)$, will occur at the time instance when the template, $w(n)$, is present in the observed signal, $x(n)$, that is, when the pattern in $x(n)$ matches the feature (pattern), $w(n)$. This technique also operates in the presence of noise, with the matched filter aiming to maximize the signal to noise ratio. However, the implementation of the cross-correlation in (1) is awkward for streaming data, where it is customary to use a standard digital filter, given by

$$y(n) = \sum_m x(n-m)w(m) = x(n) * w(n). \tag{2}$$

This filter performs the convolution between $x(n)$ and $w(n)$, denoted by $y(n) = x(n) * w(n)$. A comparison between the cross-correlation in (1) and the convolution in (2) immediately suggests a way to implement the matched filter through convolution, namely by time-reversing the template, $w(m) \to w(-m)$, in the convolution sum in (2), to arrive at

$$\begin{aligned} y(n) = x(n) * w(-n) &= \sum_m x(n-m)w(-m) \\ &= \sum_m x(n+m)w(m) = x(n) *_c w(n). \end{aligned} \tag{3}$$

The symbol $*_c$ denotes the convolution with the time-reversed feature/template vector, $w(-n)$, which also serves as the impulse response of this filter. Figure 1 illustrates the operation of a matched filter which searches for a bipolar squarewave template in the observed signal, and is implemented through convolution and thresholding.

Remark 1: The convolution operation in convolutional neural networks is applied after one of the signals is time-reversed, that is, $x(n) * w(-n) = x(n) *_c w(n)$, which is implicitly indicated in CNN implementations in the literature, for example, $\mathbf{x} * rot180^0(\mathbf{w})$ or $\text{conv}(\mathbf{x}, reverse(\mathbf{w}))$. Therefore, the convolutional layer in CNNs performs precisely the matched filtering operation, described in (3) and Figure 1, yet NNs equipped with this pattern matching operation are referred to as *convolutional neural networks* (CNNs), and not *correlational*.

Remark 2: Convolution-based feature detection is independent of the feature position within the considered signal, $x(n)$, since $y(n) = x(n) * w(-n)$ is calculated by sliding the window (filter/kernel), $w(-n)$, of length M which multiplies the corresponding segments of the signal, $x(n)$, for all n.

B. Interpretation of the building blocks within CNNs

Consider the problem of determining the best match between a received signal, $x(n)$, and one of the template waveforms, $w_k(n)$, from a predefined set of K such template waveforms (dictionary). For template waveforms with equal normalized energies, the maximum of the cross-correlation between $x(n)$ and the elements of $\{w_k(n),\ k = 1,2,\ldots,K\}$, will indicate the best match between the signal, $x(n)$, and the features, $w_k(n)$. In light of Remark 1, this can be performed by passing the received waveform through a bank of digital filters, each having as the impulse response one of the time-reversed template waveforms from the dictionary, $w_k(-n)$, that is, through the convolution

$$y_k(n) = x(n) * w_k(-n) = x(n) *_c w_k(n), \quad k = 1,\ldots,K. \tag{4}$$

Finally, the decision on which feature, $w_k(n)$, is contained in the input signal is based on a simple thresholding operation

$$k = \arg\{\max_k \{A_k,\ k = 1,2,\ldots,K\}\} \tag{5}$$

where $A_k = \max_n\{x(n) *_c w_k(n)\}$.

Remark 3: **(ReLU)** The decision in (5), on whether the input, $x(n)$, contains a feature, $w_k(n)$, is based on the maximum value of the output of the bank of matched filters and will not be affected if the negative values in the output in (4) are not considered, that is, upon the application of the mapping

$$o_k(n) = \text{ReLU}\{y_k(n)\} = \text{ReLU}\{x(n) *_c w_k(n)\},\ k = 1,\ldots,K \tag{6}$$

where ReLU denotes the *Rectified Linear Unit*, a common nonlinear activation function in CNNs, defined by

$$\text{ReLU}(x) = \max\{0, x\}. \tag{7}$$

The decision in (5) is also not affected by the scaling the amplitudes of all kernels, $w_k(n)$, by the same positive factor, even at each iteration. The same reasoning presented for the ReLU applies to the unipolar sigmoid (tanh, logistic) and unipolar binary nonlinearities.

Remark 4: **(Max–pooling)** The maximum in (6) is calculated over all available samples of the signal, $x(n)$, which is computationally inefficient. This operation will not be compromised if the output domain is split into sub-domains of P time instants, since after a local maximum is detected we are not interested in the neighboring values of the matched filter output which correspond to the same feature. The presence of local features is

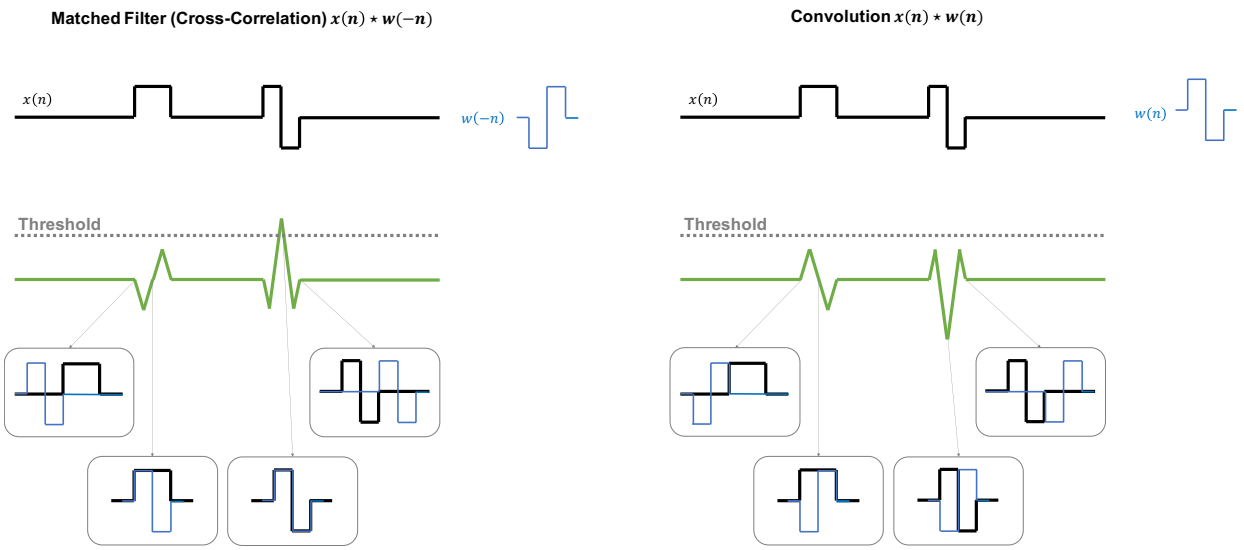

Fig. 1. Operation of a matched filter, which confirms the presence and position of a known squarewave template (thin blue line), w, in the observed signal, x (thick black line). The matched filter essentially performs the cross-correlation between a known template of interest, $w(n)$, and the unknown input, $x(n)$. In practical applications, this cross-correlation is implemented through a convolution between $x(n)$ and a time-reversed template, $w(-n)$, as in the left panel.

then examined within these sub-domains, which underpins the so called *max-pooling* operation in CNNs and their efficiency over standard NNs. For more detail, see Supplement B.

We have shown that the convolutional layer within a CNN, which consists of a set of convolutional filters, effectively operates as a bank of matched filters. Then, hierarchical features of varying degrees of complexity can be catered for by employing several convolutional layers in a CNN. In addition, to learn various aspects of the feature space, different convolution filters can be applied at different convolutional layers, each aiming to "zoom into" a different feature within the analyzed signal. Such flexibility makes convolutional networks suitable for robust and efficient analysis and classification of signals and images.

Remark 5: In a special case when all convolution filter waveforms are symmetric, that is, $w_k(n) = w_k(-n)$, the convolution filter and the matched filter produce the same result, $x(n) * w_k(n) = x(n) * w_k(-n) = x(n) *_c w_k(n)$.

C. Graph filtering

Consider a graph signal which is observed at N vertices of a graph, and given by

$$\mathbf{x} = [x(1),\ x(2),\ \ldots, x(N)]^T. \tag{8}$$

In order to extend the matched filtering perspective of CNNs to GCNNs, we shall first revisit the notion of the system on a graph. More detail can be found in Supplement A.

Graph filtering. The standard finite impulse response (FIR) system in the discrete-time domain, with the coefficients h_0, h_1, ..., h_{M-1}, is given by

$$y(n) = h_0 x(n) + h_1 x(n-1) + \cdots + h_{M-1} x(n-M+1).$$

In a direct analogy, a system on a graph is defined using a graph shift operator, $\mathbf{S}$, to give [12], [13]

$$\mathbf{y} = h_0 \mathbf{x} + h_1 \mathbf{S}\mathbf{x} + \cdots + h_{M-1} \mathbf{S}^{M-1}\mathbf{x}. \tag{9}$$

First-order system on a graph. This concept is central to our perspective on GCNNs, and may assume different forms depending on the choice of the graph shift operator, $\mathbf{S}$, in (9). Commonly used first-order systems on a graph include those:

1) With the graph Laplacian, $\mathbf{L}$, as the graph shift operator, that is $\mathbf{S} = \mathbf{L}$, the first order system assumes the form

$$\mathbf{y} = h_0 \mathbf{x} + h_1 \mathbf{L}\mathbf{x}. \tag{10}$$

Notice that the calculation of output signal requires only samples from the one-neighborhood of every vertex.

2) Employing the normalized graph Laplacian, $\mathbf{L}_N$, shift

$$\begin{aligned}\mathbf{L}_N &= \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{D}^{-1/2}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-1/2} \\ &= \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{W}_N,\end{aligned}$$

where $\mathbf{W}_N = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ is also commonly used as a shift operator in convolutional layers of GCNNs [6], [14]. When $\mathbf{S} = \mathbf{L}_N$, the first-order graph system has the form

$$\mathbf{y} = (h_0 \mathbf{L}_N^0 + h_1 \mathbf{L}_N^1)\,\mathbf{x} = (h_0 + h_1)\mathbf{x} - h_1 \mathbf{W}_N \mathbf{x}. \tag{11}$$

3) For **multichannel systems**, relation (11) should involve the channel index, k. Then, for the k-th channel, with the input signal, $\mathbf{x}$, and the output, $\mathbf{y}_k$, we have

$$\begin{aligned}\mathbf{y}_k &= w_k(0)\mathbf{x} + w_k(1)\mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}\mathbf{x} \\ &= w_k(0)\mathbf{x} + w_k(1)\mathbf{W}_N\mathbf{x}, \quad k = 1,2,\ldots,K\end{aligned} \tag{12}$$

where the weights $w_k(0)$ and $w_k(1)$ correspond respectively to the weights $(h_0 + h_1)$ and $(-h_1)$ in (11). A simplification which uses only one parameter, $w_k(0) = w_k(1) = \theta_k$, was originally proposed for the

convolutional layer in GCNNs in [6], [15], in the form $\mathbf{y}_k = \theta_k(\mathbf{x} + \mathbf{W}_N\mathbf{x})$. However, this reduces the parameter space (over-simplifies the feature form) of GCNNs and we resort to (12) – see Supplement A for more detail.

Remark 6: The common choice of $w_k(0) = w_k(1)$ forces all learned kernels to perform low-pass filtering (see Supplement), leading to the problem of over–smoothing in GCNN. Observe that this problem is readily rectified by simply not imposing the (unnecessary) constraint $w_k(0) = w_k(1)$, as in the matched filtering interpretation in (11) and (12). Such an intuitive and elegant solution has been overlooked by many attempts trying to resolve the over–smoothing problem in GCNNs.

4) Based on the *random walk graph shift operator*, $\mathbf{S} = \mathbf{D}^{-1}\mathbf{W}$, which produces the first-order multichannel system

$$\mathbf{y}_k = w_k(0)\mathbf{x} + w_k(1)\mathbf{D}^{-1}\mathbf{W}\mathbf{x}, \quad k = 1,2,\ldots,K. \quad (13)$$

Notice that this graph shift operator does not, generally, preserve the symmetry property of the shift matrix, so that the eigenvectors may assume complex values.

5) For directed unweighted graphs, we may use the adjacency matrix, $\mathbf{A}$, as a (non-symmetric) graph shift, to yield

$$\mathbf{y}_k = w_k(0)\mathbf{x} + w_k(1)\mathbf{A}\mathbf{x} + w_k(2)\mathbf{A}^T\mathbf{x}, \quad (14)$$

where $\mathbf{A}\mathbf{x}$ denotes the backward shift and $\mathbf{A}^T\mathbf{x}$ the forward shift on a graph. Note that we can equally use the normalized adjacency matrix, $\mathbf{A}/\lambda_{\max}$ as a graph shift.

Spectral description of a system on a graph is obtained from the eigendecomposition of the graph shift operator $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}$, where $\mathbf{U}$ is the transformation matrix with the eigenvectors of $\mathbf{S}$ as its columns, and $\mathbf{\Lambda}$ is a diagonal matrix of the eigenvalues of $\mathbf{S}$. Upon pre-multiplying the vertex domain relation in (9) by the inverse transformation, $\mathbf{U}^{-1}$, we obtain

$$\mathbf{Y} = h_0\mathbf{X} + h_1\mathbf{\Lambda}\mathbf{X} + \cdots + h_{M-1}\mathbf{\Lambda}^{M-1}\mathbf{X}, \quad (15)$$

where $\mathbf{X} = \mathbf{U}^{-1}\mathbf{x}$ and $\mathbf{Y} = \mathbf{U}^{-1}\mathbf{y}$ are the graph Fourier transforms (GFT) of the graph signals $\mathbf{x}$ and $\mathbf{y}$, with the corresponding inverse graph Fourier transforms (IGFT), $\mathbf{x} = \mathbf{U}\mathbf{X}$ and $\mathbf{y} = \mathbf{U}\mathbf{Y}$. Notice that the transfer function of this system is a diagonal matrix, $H(\mathbf{\Lambda})$, defined by

$$\mathbf{Y} = H(\mathbf{\Lambda})\mathbf{X} = (h_0 + h_1\mathbf{\Lambda} + \cdots + h_{M-1}\mathbf{\Lambda}^{M-1})\mathbf{X}. \quad (16)$$

Remark 7: **(Graph convolution)** The graph convolution operator follows directly from the element-wise form of the transfer function of the system on a graph in (16), given by $Y(k) = H(\lambda_k)X(k)$. Physically, graph convolution is the inverse GFT of $Y(k)$, that is $y(n) = x(n) \star h(n) = \text{IGFT}\{X(k)H(\lambda_k)\}$, which follows from classical analysis on regular domains [16].

In order to distinguish between the classical convolution and graph convolution we will use slightly different symbols $*$ and $\star$, respectively. To distinguish between the GFT of a signal, $X(k)$, and the transfer function calculated based on the eigenvalues, $H(\lambda_k)$, we use different notations (positions of index k) for these two functions. This topic is studied in detail in [13], [17] and in Supplement A.

D. Matched Filter on a Graph (GMF)

Consider a graph signal, $x(n)$, and assume that it is processed by a system on graph whose transfer function is $G(\lambda_k)$, with $g(n) = \text{IGFT}\{G(\lambda_k)\}$. The output of this system is then

$$\begin{aligned} y(n) = x(n) \star g(n) &= \text{IGFT}\{X(k)G(\lambda_k)\} \\ &= \sum_{k=1}^{N} X(k)G(\lambda_k)u_k(n), \end{aligned}$$

with $u_k(n)$ as elements of the k-th column (k-th eigenvector) of the eigenvector matrix, $\mathbf{U}$.

A (graph) matched filter design problem consists of finding the transfer function, $G(\lambda_k)$, of the system that maximizes the power of the output signal, $y(n)$, for an input signal $x(n)$, at a vertex $n = n_0$. Then, the system output at n_0, $y(n_0)$, is

$$\begin{aligned} |y(n_0)|^2 &= \Big| \sum_{k=1}^{N} X(k)G(\lambda_k)u_k(n_0) \Big|^2 \\ &\le \sum_{k=1}^{N} |X(k)|^2 \sum_{k=1}^{N} |G(\lambda_k)u_k(n_0)|^2. \end{aligned}$$

According to the Schwartz inequality above, the maximum is achieved when the equality holds which, in general, yields the transfer function of the graph matched filter in the form (with $(\cdot)^*$ as the complex conjugate) given by

$$G(\lambda_k)u_k(n_0) = X^*(k), \quad (17)$$

up to a possible scaling constant. In our context, we are not interested in the minimum value, which occurs for a negative scaling constant, that is, $G(\lambda_k)u_k(n_0) = -X^*(k)$. From (17), the maximum value of the output is given by

$$y(n_0) = \sum_{k=1}^{N} |X(k)|^2 = E_x,$$

where E_x is the input signal energy.

Remark 8: In classical analysis (where the GFT is complex-valued and corresponds to the standard DFT, it is well-known that

$$G(\lambda_k) = X^*(k)/u_k(n_0) = X^*(k)e^{-j2\pi n_0 k/N}/\sqrt{N}.$$

For $n_0 = 0$, the matched filter impulse response becomes

$$g(n) = \text{IDFT}\{G(\lambda_k)\} = x^*(-n).$$

However, in the case of general graphs and graph signals, the vertex domain form of (17) is much more complicated due to the fact that $G(\lambda_k)$ is calculated based on the eigenvalues, while $X(k)$ is the GFT of the signal, see Supplement A.

We now proceed to analyze more general forms of graph matched filters, starting with a special case when the input signal can be considered as a result of a *diffusion process on a graph*.

Graph matched filter. The intuition and implementation of the graph matched filter can be significantly simplified if the considered graph signal, $x(n)$, is a result of an $(M-1)$-step diffusion process, with the unit delta pulse at an arbitrary vertex n_0 as the initial signal, $\mathbf{x}_0$, in the diffusion.

Consider a graph signal arising from an $(M-1)$-step diffusion starting from a delta pulse, $\mathbf{x}_0$, at a vertex n_0, that is, $x_0(n)=\delta(n-n_0)$. By repeatedly applying the graph shift operator $(M-1)$ times, the resulting graph signal, $\mathbf{x}$, is given by

$$\mathbf{x}=a_0\mathbf{x}_0+a_1\mathbf{L}_N\mathbf{x}_0+\cdots+a_{M-1}\mathbf{L}_N^{M-1}\mathbf{x}_0, \qquad (18)$$

where a_ν, $\nu=0,1,\ldots,M-1$, are the diffusion constants, while the GFT of $\mathbf{x}_0$ is defined as

$$X_0(k)=\sum_{n=1}^{N}x_0(n)u_k(n)=u_k(n_0).$$

The GFT of the graph signal, $\mathbf{x}$, follows from (18) and (15) as

$$\mathbf{X}=\Big(a_0+a_1\mathbf{\Lambda}_N+\cdots+a_{M-1}\mathbf{\Lambda}_N^{M-1}\Big)\mathbf{X}_0, \qquad (19)$$

or in an element-wise form

$$X(k)=\Big(a_0+a_1\lambda_k+\cdots+a_{M-1}\lambda_k^{M-1}\Big)u_k(n_0). \qquad (20)$$

The spectral domain solution of (17) follows immediately as

$$G(\lambda_k)=a_0+a_1\lambda_k+\cdots+a_{M-1}\lambda_k^{M-1} \qquad (21)$$

by having in mind that $X(k)$ is real-valued for a symmetric shift operator like $\mathbf{L}_N$ in (18). Although the vertex domain form of $g(n)$ in the graph convolution $y(n)=x(n)*g(n)$ may be quite complex, the vertex domain implementation of such a matched filter is rather simple and follows from $Y(k)=G(\lambda_k)X(k)$ to yield

$$\mathbf{y}=\Big(a_0+a_1\mathbf{L}_N+\cdots+a_{M-1}\mathbf{L}_N^{M-1}\Big)\mathbf{x}. \qquad (22)$$

For $\mathbf{x}=\mathbf{x}_0$, we obtain the relation between the matched filter output and the initial delta pulse signal in the diffusion system.

Remark 9: **(Graph Matched Filter (GMF))** The vertex domain form of GMF represents the IGFT of the graph filter transfer function, and is given by

$$g(n)=\sum_{k=1}^{N}G(\lambda_k)u_k(n). \qquad (23)$$

To arrive at the vector-matrix form of the above GMF, denote

$$g_0(n)=\sum_{k=1}^{N}u_k(n). \qquad (24)$$

With $\mathbf{g}_0$ as a vector whose elements are sums of all eigenvector elements, $u_k(n)$, at a given vertex n, we have

$$\mathbf{g}_0=\mathbf{U}\mathbf{1}, \qquad \mathbf{1}=[1,1,\ldots,1]^T.$$

In general, the elements $g_0(n)$ are nonzero for all n, making it quite different from $x_0(n)$ in (18). The matrix form of the vertex domain matched filter then becomes

$$\begin{aligned}\mathbf{g}=\mathbf{U}G(\mathbf{\Lambda})\mathbf{1}&=\mathbf{U}\Big(a_0+a_1\mathbf{\Lambda}_N+\cdots+a_{M-1}\mathbf{\Lambda}_N^{M-1}\Big)\mathbf{1}\\&=a_0\mathbf{g}_0+a_1\mathbf{L}_N\mathbf{g}_0+\cdots+a_{M-1}\mathbf{L}_N^{M-1}\mathbf{g}_0. \qquad (25)\end{aligned}$$

Unlike in the standard time domain where $x(n)=g(-n)$, although the signals $\mathbf{x}$ in (18) and the matched filter response $\mathbf{g}$ in (25) have similar forms in the spectral domain (given by (17)), they significantly differ due to the different forms of $\mathbf{x}_0$ and $\mathbf{g}_0$, defined respectively by $x_0(n)=\delta(n-n_0)$ and (24).

In classical analysis, with the adjacency matrix on a directed circular graph serving as a shift operator [13], we immediately arrive at the impulse response of the classical matched filter

$$g_0(n)=\sum_{k=1}^{N}u_k(n)=\sum_{k=1}^{N}e^{j2\pi nk/N}/\sqrt{N}=\delta(n)\sqrt{N},$$
$$\lambda_k=e^{-j2\pi k/N}/\sqrt{N}, \quad \text{and}$$
$$g(n)=a_0\delta(n)+a_1\delta(n-1)+\cdots+a_{M-1}\delta(n-M+1).$$

The calculation of the responses of the graph matched filter is much more involved, and will be elucidated through an intuitive example. We considered a simple undirected 8-vertex graph with unit edge weights, shown in Fig. 2(a), and a unit pulse graph signal, shown in Fig. 2(b). This graph was used as an irregular signal domain for the considered analysis (the exact values of the weight matrix, $\mathbf{W}$, elements for this graph is given in Supplement B by relation (SM-**??**)).

Example 1. **(Matched filter on a graph)** *To illustrate the operation of the matched filter on a graph, consider two signals (features) on the graph from Fig. 2(a), which are created through diffusion by graph shifting a delta pulse, $x_0(n)=\delta(n-n_0)$, from a vertex n_0 to its one-neighborhood, as in the first order system on a graph in (11).*

(i) The first signal (feature), $\mathbf{x}_1$, is produced by a graph shift

$$\mathbf{x}_1=\mathbf{x}_0+3\mathbf{W}_N\mathbf{x}_0=4\mathbf{x}_0-3\mathbf{L}_N\mathbf{x}_0,$$

of a pulse input $x_0(n)=\delta(n-3)$, shown in Fig. 2(b).

(ii) The second signal (feature), $\mathbf{x}_2$, employs a different shift weight

$$\mathbf{x}_2=\mathbf{x}_0-2.5\mathbf{W}_N\mathbf{x}_0=-1.5\mathbf{x}_0+2.5\mathbf{L}_N\mathbf{x}_0,$$

and a different pulse input, $x_0(n)=\delta(n-4)$.

Both features are generated based on same $\mathbf{L}_N=\mathbf{I}-\mathbf{W}_N$, where

$$\mathbf{W}_N=\begin{bmatrix}0&0.26&\frac{1}{3}&0&0&0&0&\frac{1}{3}\\0.26&0&0.26&0.22&\frac{1}{5}&0&0&0.26\\\frac{1}{3}&0.26&0&0.29&0&0&0&0\\0&0.22&0.29&0&0.22&0.29&0&0\\0&\frac{1}{5}&0&0.22&0&0.26&0.32&0.26\\0&0&0&0.29&0.26&0&0.41&0\\0&0&0&0&0.32&0.41&0&0\\\frac{1}{3}&0.26&0&0&0.26&0&0&0\end{bmatrix}. \qquad (26)$$

Therefore, the features $\mathbf{x}_1$ and $\mathbf{x}_2$ are generated by diffusion from a pulse located at different vertices, and based on a positive weight $+3$ for $\mathbf{x}_1$ and a negative weight -2.5 for $\mathbf{x}_2$. Consequently, the shapes of $\mathbf{x}_1$ and $\mathbf{x}_2$ are different, as shown in Fig. 2(c) and (d), with their vertex-index axis representations given in Fig. 2(e) and (f).

Based on the general form of vertex domain GMF in (25), the impulse responses of the GMFs, $\mathbf{g}_1$ and $\mathbf{g}_2$, corresponding to the features $\mathbf{x}_1$ and $\mathbf{x}_2$ assume the form

$$\mathbf{g}_1=4\mathbf{g}_0-3\mathbf{L}_N\mathbf{g}_0, \qquad (27)$$
$$\mathbf{g}_2=-1.5\mathbf{g}_0+2.5\mathbf{L}_N\mathbf{g}_0 \qquad (28)$$

and are depicted in Fig. 2(g) and (h), with $g_0(n)=\sum_{k=1}^{N}u_k(n)$.

The respective matched filter responses (outputs) are next calculated using the vertex domain graph filters, $\mathbf{g}_1$ and $\mathbf{g}_2$, to yield

$$\mathbf{y}_{11}=4\mathbf{x}_1-3\mathbf{L}_N\mathbf{x}_1, \quad \mathbf{y}_{21}=4\mathbf{x}_2-3\mathbf{L}_N\mathbf{x}_2$$
$$\mathbf{y}_{12}=-1.5\mathbf{x}_1+2.5\mathbf{L}_N\mathbf{x}_1, \quad \mathbf{y}_{22}=-1.5\mathbf{x}_2+2.5\mathbf{L}_N\mathbf{x}_2 \qquad (29)$$

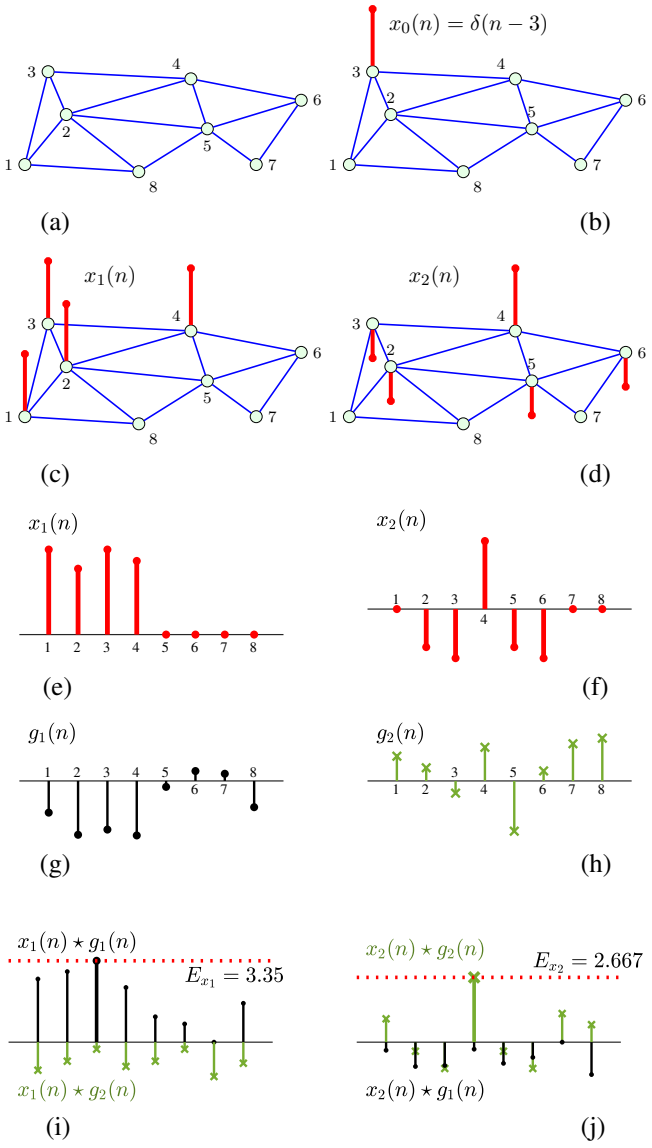

Fig. 2. Example of matched filtering on an undirected unweighted graph, with two input signals, $\mathbf{x}_1$ and $\mathbf{x}_2$, (containing different features) observed in two different scenarios. (a) Considered graph topology. (b) Unit pulse graph signal, $x_0(n) = \delta(n-3)$, at a vertex $n = 3$. (c) The input signal with the first feature, $\mathbf{x}_1$, is obtained by shifting the unit pulse $x_0(n) = \delta(n-3)$ from the vertex $n = 3$ to its one-neighborhood, with the graph shift weight of 3, that is, $\mathbf{x}_1 = \mathbf{x}_0 + 3\mathbf{W}_N\mathbf{x}_0$, with $\mathbf{W}_N$ given in (26). (d) The second input signal (second feature) is obtained by shifting the unit pulse, $x_0(n) = \delta(n-4)$, to its one-neighborhood vertices, with the graph shift weight of -2.5, that is, $\mathbf{x}_2 = \mathbf{x}_0 - 2.5\mathbf{W}_N\mathbf{x}_0$. The graph signals from panels (c) and (d) are respectively given on a linear vertex-index axis in (e) and (f). The impulse responses of the corresponding matched filters, $\mathbf{g}_1 = 4\mathbf{g}_0 - 3\mathbf{L}_N\mathbf{g}_0$ and $\mathbf{g}_2 = -1.5\mathbf{g}_0 + 2.5\mathbf{L}_N\mathbf{g}_0$, are shown respectively in panels (g) and (h). The outputs of the graph matched filters, with the impulse response $g_1(n)$ corresponding to the feature $x_1(n)$ and $g_2(n)$ corresponding to $x_2(n)$, are given respectively in panels (i) and (j). Observe from panel (i) that, as desired, the matched filter $\mathbf{g}_1$ correctly detected the presence of the feature $\mathbf{x}_1$ (black line), with the maximum output at $n = 3$, while the matched filter $\mathbf{g}_2$ failed to detect the feature $\mathbf{x}_1$ as it was not designed for this purpose (green line). Panel (j) depicts an analogous scenario for the detection of the feature $\mathbf{x}_2$, with the matched filter $\mathbf{g}_2$ performing a correct detection with the maximum output at $n = 4$. The maximum values of the outputs of the two matched filters (energies of the corresponding input features) are marked by larger symbols and a dotted red line, while their respective values are denoted by E_{x_1} and E_{x_2}.

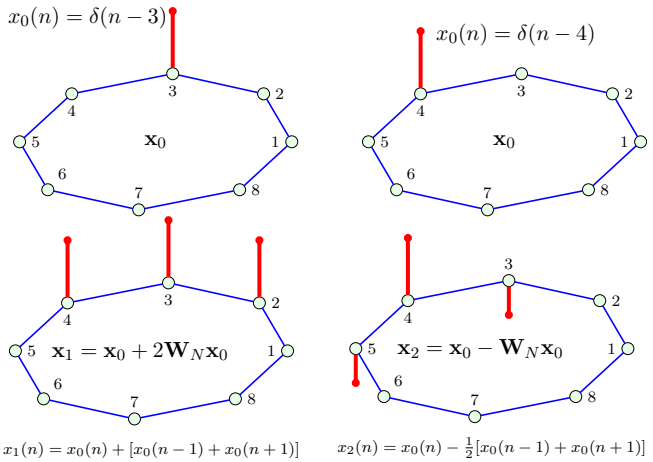

Fig. 3. An undirected unweighted circular graph with two features, $\mathbf{x}_1 = [0,1,1,1,0,0,0,0]^T$ and $\mathbf{x}_2 = [0,0,-0.5,1,-0.5,0,0,0]^T$. The feature $\mathbf{x}_1 = \mathbf{x} + 2\mathbf{W}_N\mathbf{x}$ is obtained by shifting the unit pulse, $x(n) = \delta(n-3)$, to both the left and right, with the shift weight of 2 and $\mathbf{W}_N$ as the adjacency (weight) matrix which was normalized by the degree matrix whose diagonal elements are equal to 2. The feature $\mathbf{x}_2 = \mathbf{x} - \mathbf{W}_N\mathbf{x}$ is produced by shifting the unit pulse, $x(n) = \delta(n-4)$, with the shift weight of -1. Notice a direct correspondence with regularly sampled time-domain analysis in standard CNNs, as the graph matched filter operating on a circular graph directly simplifies into a standard matched filter defined for the regular time-domain signal assumed by DFT.

For completeness, these results were verified through the corresponding spectral domain relations (see Supplement A)

$$y_{ij}(n) = x_i(n) * g_j(n) = \text{IGFT}\{GFT\{x_i(n)\}GFT\{g_j(n)\}\}. \quad (30)$$

While the vertex form in (29) and the spectral form in (30) produced identical results, the vertex domain relation is simpler for practical realization since it only requires local signal samples from the one-neighborhood of every vertex.

The outputs of the GMFs corresponding to the two features of interest are given in Fig. 2(i) and (j), and as desired, their maximum values are equal to the energies of the corresponding graph features, $\mathbf{x}_1$ and $\mathbf{x}_2$ (theoretical maximum output of a matched filter).
For enhanced intuition and direct comparison with the classical time domain matched filters, a similar scenario is presented on an undirected and unweighted circular graph in Fig. 3.

E. Forward Propagation

We now proceed to shed new light on the key steps of the operation of GCNNs from a matched filter perspective. For clarity, we assume that the weights of the convolution filters (*forward propagation*) are already initialized or calculated. In the following, we elaborate in detail upon the forward propagation path in GCNNs, while the adaptive learning procedure which includes backpropagation on graphs and graph coarsening is covered in the Supplement.

1) **Input.** The graph input signal, $\mathbf{x} = [x(1),\ldots,x(N)]^T$, is observed at N vertices of a graph. A common goal in GCNNs is to classify inputs into several non-overlapping categories which correspond to specific features in data.
2) **Convolutional layer.** This operation employs a convolution filter of M elements which corresponds to a GMF in (25), and is also called the *convolution kernel*. If we are

looking for K features in $\mathbf{x}$, then K GMFs are required, with the elements of the k-th GMF (graph convolutional layer) given by (see also Section C)

$$\mathbf{w}_k = [w_k(0),\, w_k(1),\, \ldots,\, w_k(M-1)]^T,\ \ k = 1,2,\ldots,K.$$

A common choice in GCNNs is the first-order system in (10), with a kernel/filter with $M = 2$ coefficients. With the normalized weight matrix as a shift operator, the $N \times 1$ output signals, $\mathbf{y}_k$, from the graph matched filters are

$$\begin{aligned}\mathbf{y}_k = \mathbf{w}_k \star \mathbf{x} &= w_k(0)\mathbf{x} + w_k(1)\mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}\mathbf{x} \\ &= w_k(0)\mathbf{x} + w_k(1)\mathbf{W}_N\mathbf{x}. \end{aligned} \tag{31}$$

Analogously, for $M = 3$, the matched filter channels within GCNNs employ a second-order system on a graph

$$\mathbf{y}_k = \mathbf{w}_k \star \mathbf{x} = w_k(0)\mathbf{x} + w_k(1)\mathbf{W}_N\mathbf{x} + w_k(2)\mathbf{W}_N^2\mathbf{x}. \tag{32}$$

With K convolution kernels in a graph convolutional layer, the total number of the output signal elements is $K \times N$. From (31), for the normalized graph Laplacian, $\mathbf{L}_N$, as the shift operator, the graph convolution filter becomes

$$\mathbf{y}_k = \mathbf{w}_k \star \mathbf{x} = w_k(0)\mathbf{x} + w_k(1)\mathbf{L}_N\mathbf{x}. \tag{33}$$

Remark 10: The total number of convolution filter weights, $w_k(m)$, $k = 1,2,\ldots,K$, $m = 0,1,\ldots,M-1$, in the graph convolutional layer is $M \times K$, with M as the length of the graph convolution filter and K as the number of convolution filters (features). Given that the size of the convolution filter is much smaller than the number of the input signals, i.e. $M \ll N$, this makes GCNNs exhibit considerable computational advantages over fully connected GNNs, where the number of weights is $N \times K$.

Example 2. ***Relation to standard CNNs.*** *The input-output relation of standard CNNs is as a special case of that for GCNNs, for circular undirected and unweighted graphs, as shown in Fig. 3. This immediately follows from the element-wise form of the graph convolution filter in (31) [3]*

$$y_k(n) = w_k(0)x(n) + \frac{1}{2}w_k(1)[x(n+1) + x(n-1)],$$

where the factor $1/2$ arises due to the degree matrix, and can be absorbed into the weights, $w_k(v)$. This form is symmetric, since the graph in Fig. 3 is undirected. An asymmetric form may be obtained by using the adjacency matrix, $\mathbf{A}$, of a directed unweighted circular graph, instead of the normalized weight matrix, $\mathbf{W}_N$, to yield the standard CNN convolution

$$y_k(n) = w_k(0)x(n) + w_k(1)x(n+1) + w_k(2)x(n+2).$$

3) **Bias.** As in standard NN layers, a constant bias term, b_k, may be added at every graph convolutional layer, to yield

$$\mathbf{y}_k = \mathbf{w}_k \star \mathbf{x} + b_k.$$

at a cost of one more coefficient per convolution.

4) **Nonlinear activation function.** The convolution is a linear operation, however, signals and images largely exhibit a nonlinear nature. This is catered for by applying a non-linearity at the output of convolutional layers. The most common nonlinear activation function in CNNs and GCNNs is the Rectified Linear Unit (ReLU), defined by

$$f(y) = \max\{0, y\}. \tag{34}$$

Within GCNNs, ReLU exhibits several advantages over sigmoidal-type activation functions: (i) it does not saturate for positive inputs, producing nonzero gradient for large input values; (ii) it is computationally cheap to implement; (iii) in practice, ReLU-based neural networks converge faster during the learning process than saturation-type nonlinearities (logistic, tanh). Moreover, the use of ReLU facilitates further computational advantages through "*sparsification by deactivation*", as from (34) neurons with negative output values are omitted. *Recall, as shown in Remark 3, that ReLU can be naturally considered within our matched filtering perspective of CNNs and GCNNs.* The output of the graph convolutional layer, after the activation function and with the bias term, now becomes

$$f(\mathbf{y}_k) = f(\mathbf{w}_k \star \mathbf{x} + b_k).$$

Notice that a problem arises for many consecutive negative inputs to a neuron, whereby the corresponding zero-output of ReLU means that such a neuron will be left without its weight update (dying ReLU). This can be avoided through the *Leaky ReLU* function, where the negative inputs to ReLU are scaled by small factors, for example, $f(y_k(n)) = 0.01y_k(n)$, for $y_k(n) < 0$.

5) **Pooling.** As stated in Remark 4, to further reduce computational complexity, the output signals at GCNN layers are typically down-sampled – the so called *pooling* operation

$$\mathbf{o}_k = F\Big(f(\mathbf{w}_k \star \mathbf{x} + b_k)\Big). \tag{35}$$

The max-pooling in GCNN considers only the maximum output of feature filters, thus considerably reducing the number of parameters, and is closely related to *graph downscaling* [5]. One approach to graph downscaling is termed *graph coarsening*, and is given in Supplement B.

6) **Flattening.** Assume that for every channel, k, the number of remaining vertices (signal samples) after the ReLU and max-pooling operations is $N' < N$. Then, the K output vectors, $\mathbf{o}_k$ in (35), can be concatenated to form the overall output vector, $\mathbf{o}_F$, of which the elements are given by

$$o_F(m) = o_F\big((k-1)N' + n\big) = o_k(n),$$

with $k = 1,2,\ldots,K$, $n = 1,2,\ldots,N'$. Therefore, with no pooling the *flattened* output $\mathbf{o}_F$ would be of size KN while with max-pooling with a factor of P, its size reduces to $KN' = KN/P$.

7) **Repeated graph convolutions.** To identify quite complex patterns, like *hierarchically composed features*, the convolution step can be repeated one or more times before producing the overall output of the convolutional layer, prior to flattening. Each repeated convolution step may employ a different set of filter functions (features), and may or may not use the nonlinear activation and pooling steps – the so-called *convolution-activation-pooling* chain.

8) **Fully Connected (FC) Layers.** The flattened output

of the convolutional layer is typically fed into a fully connected neural network layer with e.g. a standard *multilayer structure*, followed by the output layer.
Fig. 4 illustrates the operation of a GCNN with:

- A graph input, $\mathbf{x}$, and the graph weight matrix, $\mathbf{W}$;
- One convolutional layer with weights $w_k(m)$, two features, $K=2$, and convolution filter length of $M=2$;
- One fully connected standard neural network layer with $KN=16$ input neurons and $S=2$ output neurons;
- Softmax output layer with $S=2$ outputs.

This model is also used in Example 3, which provides an in–depth illustration of the operation of the forward path in GCNNs.

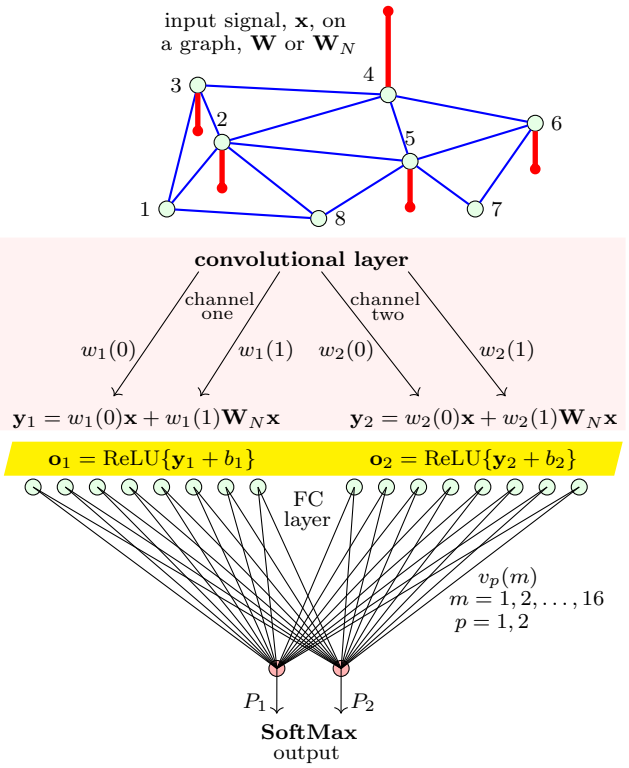

Fig. 4. Principle of operation of GCNNs, based on a GCNN with one graph convolutional layer, one FC layer, and two (SoftMax) neurons at the output layer. The network is presented with noisy versions of either $\mathbf{feature}_1$ or $\mathbf{feature}_2$ from Fig. 2 (c) and (d). Shown on top panel is $\mathbf{feature}_2$.

Example 3. (Operation of GCNN) *Without loss of generality, the operation of GCNNs is elucidated over a two-layer network with one graph convolutional layer and one fully connected layer, given in Fig. 4. The input signal,* $\mathbf{x}$*, with* $N=8$ *samples represents a noisy version of either of the first or second feature from Figure 2, that is*

$$\mathbf{x} = \mathbf{feature}_1 + \boldsymbol{\varepsilon} = \mathbf{x}_0 - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}\mathbf{x}_0 + \boldsymbol{\varepsilon} \tag{36}$$

$$\mathbf{x} = \mathbf{feature}_2 + \boldsymbol{\varepsilon} = \mathbf{x}_0 + \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}\mathbf{x}_0 + \boldsymbol{\varepsilon} \tag{37}$$

with $\boldsymbol{\varepsilon}$ *as noise, while the graph signal,* $\mathbf{x}_0$*, was a pulse* $x_0(n) = \delta(n-n_0)$ *at a random vertex,* n_0*. These features are similar to the graph signals in Fig. 2(c),(d), with the weights,* $\mathbf{W}$*, given in (26).*

The target signal for $\mathbf{feature}_1$ *was* $\mathbf{t}_1 = [1,\ 0]^T$ *and* $\mathbf{t}_2 = [0,\ 1]^T$ *for* $\mathbf{feature}_2$.
The task *was to confirm the presence of either of these two features in a noisy input, using graph convolution filters in (31) of length* $M=2$*, which corresponds to* $K=2$ *channels at the graph convolutional layer, followed by the ReLU nonlinearity in (7). The FC layer employed the SoftMax output with target values that correspond to the two patterns in the target signal,* $\mathbf{t}$*. For more detail, see Supplement C.*
Training *was performed based on* 200 *random realizations of the input signal,* $\mathbf{x}$*, which for each realization randomly assumed either* $\mathbf{feature}_1$ *or* $\mathbf{feature}_2$ *and a random central vertex,* n_0*. This cycle of* 200 *realizations is called an epoch. The GCNN in Fig. 4 was trained over 5 epochs, that is, the same set of* 200 *random realizations was employed* 5 *times, and no max-pooling was employed. Fig. 5 illustrates the evolution of the GCNN output and the weights of the output FC layer along the training process.*
The network output *represents the "probabilities" of the presence of the two features in the noisy input, denoted respectively by* P_1 *and* P_2*, which are given in the top panel of Fig. 5 and were obtained from the two SoftMax outputs of the GCNN from Fig. 4. Therefore, when* $\mathbf{feature}_1$ *is present in the noisy input,* $\mathbf{x}$*, the target signal is* $\mathbf{t}_1 = [1,\ 0]^T$ *and the output of the SoftMax layer in an ideal case should be close to* $P_1 = 1$ *and* $P_2 = 0$*. Regarding the presence of* $\mathbf{feature}_2$ *in* $\mathbf{x}$*, the corresponding target signal is* $\mathbf{t}_2 = [0,\ 1]^T$ *and the SoftMax outputs should ideally approach* $P_1 = 0$ *and* $P_2 = 1$*. The evolution of the SoftMax output during training is illustrated in Fig.5 (top), where for a consistent treatment of the overall accuracy:*

- *For the target* $\mathbf{t}_1$*, the black "+" denotes the values of* P_1 *and the green"·" the values of* P_2*,*
- *For the target* $\mathbf{t}_2$*, the black "+" denotes the values of* P_2 *and the green"·" the values of* P_1*.*

In this way, a perfectly trained GCNN will have all black "+" marks at 1 and all green "." marks at 0. Note that for the considered scenario, $P_1 + P_2 = 1$ *always holds for the SoftMax outputs.*

The graph backpropagation approach to the weight update in GNCCs is elaborated in detail in Supplement C.
The output convergence *is depicted in Fig.5 (top). Observe the success of training, as the network outputs initially assumed indecisive values around 0.5, and subsequently (over iterations with random pulse positions and feature choice) the black "+" gradually approached the correct value of* 1 *and the green"·" the correct value of* 0*.*
The weight convergence *of the* $K \times N \times S = 32$ *weights in the FC layer, described in the middle panel of Fig. 5, reflects the overall convergence of the GCNN. An almost flat weight evolution after the the initial 200 random realizations of* $\mathbf{x}$ *indicates successful training.*
Testing stage: *The success of GCNN training was evaluated over 100 new random realizations of the noisy input,* $\mathbf{x}$*. Fig. 5 (bottom) indicates the correct and highly reliable GCNN decisions, as over all 100 new random inputs presented to the network, the black "+" and green "·" were at (or very close to) their correct positions (successful training).*

All the steps and calculations in the training process are illustrated in a step-by-step fashion in Supplement D.

V. What we have learned

Graph Convolutional Neural Networks (GCNN) have been developed as extensions of standard Convolutional Neural Networks (CNN), with the aim to cater for irregular domains represented by connected graphs. Despite facilitating the transition from NNs to GNNs, such an approach harbours intrinsic disadvantages – as much of the effort has revolved around the issues surrounding domain adaptation, rather than resorting to first principles. To this end, we have revisited graph

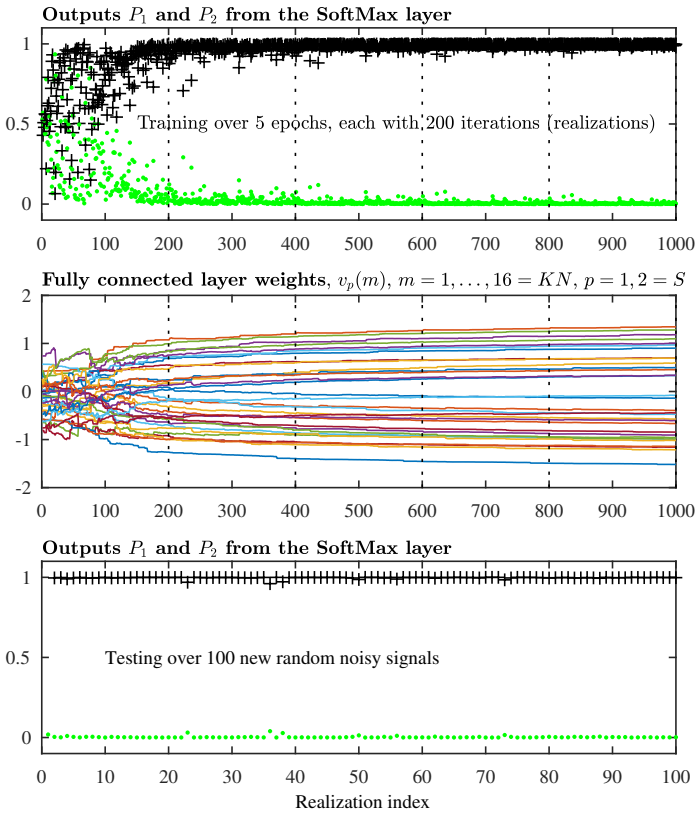

Fig. 5. Illustration of the operation of the GCNN from Fig. 4, with one graph convolutional layer, $K = 2$ convolution kernels, one FC layer, and two neurons at the output SoftMax layer, to yield $(NK) \times 2 = 16 \times 2 = 32$ weights. The task was to identify the two features from Example 1, also given in Fig. 2 c) and d). *(Top)* The output probabilities of the GCNN for the task of identification of the presence of a correct feature (either $\mathbf{feature}_1$ or $\mathbf{feature}_2$) are denoted by a black "+" if the corresponding SoftMax output should be equal to 1, and by a green "·" if the SoftMax output should be 0. (Middle) Evolution of the 32 weights in the FC layer along the training process. (Bottom) Test results over 100 random realisations of $\mathbf{x}$, with "+" and "·" as above. Observe an almost perfect identification of the two patterns in $\mathbf{x}$.

convolutional neural networks starting from the notion of a system on a graph, which has served to establish a matched-filtering interpretation of the whole *convolution-activation-pooling* chain within GCNNs, while inheriting the rigour and intuition from signal detection theory. Such an approach is shown to be quite general, and yields both standard CNNs and fully connected NNs as special cases. It is our hope that, by revisiting the underpinning principles of GCNNs through the lens of matched filtering, we have helped shed new light onto the otherwise black box approach to GCNNs, together with demystifying GCNNs to practitioners and for educational purposes. We also hope that the resulting the well-motivated and physically meaningful interpretation at every step of the operation and adaptation of GCNNs will help establish a common language between the diverse communities working on Data Analytics on Graphs.

AUTHOR BIOGRAPHIES

Ljubiša Stanković (ljubisa@ac.me) is a professor at the University of Montenegro. He was an Alexander von Humbolt fellow at the Ruhr University Bochum, Germany from 1997-1999, the Rector of the University of Montenegro from 2003-2008, and a Visiting Academic at Imperial College London, UK, from 2012-2013. He has published almost 200 journal papers, is a member of the National Academy of Science and Arts (CANU) and of Academia Europaea. Ljubiša won the Best Paper Award from EURASIP in 2017 and the IEEE Signal Processing Magazine Best Column Award for 2020. He is a Fellow of the IEEE.

Danilo P. Mandic (d.mandic@imperial.ac.uk) received his Ph.D. degree in machine intelligence from Imperial College London in 1999. He is currently a professor in the Department of Electrical and Electronic Engineering, Imperial College London, London, SW7 2AZ, U.K. He is the recipient of the Dennis Gabor Award for Outstanding Contributions in Neural Engineering by the International Neural Networks Society in 2019, the 2018 IEEE Signal Processing Magazine Best Paper Award, the ICASSP 2021 Outstanding Paper Award, and the Presidents Award for Excellence in Postgraduate Supervision at Imperial College in 2014. He is a Fellow of IEEE.

REFERENCES

[1] C.-C. J. Kuo, "Understanding convolutional neural networks with a mathematical model," *Journal of Visual Communication and Image Representation*, vol. 41, pp. 406–413, 2016.

[2] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1D convolutional neural networks and applications: A survey," *Mechanical Systems and Signal Processing*, vol. 151, p. 107398, 2021.

[3] L. Stankovic and D. Mandic, "Convolutional neural networks demystified: A matched filtering perspective based tutorial," *arXiv preprint arXiv:2108.11663*, 2021.

[4] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[5] L. Stanković, D. Mandic, M. Daković, M. Brajović, B. Scalzo, S. Li, A. G. Constantinides, *et al.*, "Data analytics on graphs Part III: Machine learning on graphs, from graph topology to applications," *Foundations and Trends® in Machine Learning*, vol. 13, no. 4, pp. 332–530, 2020.

[6] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[7] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

[8] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceedings of International Conference on Machine Learning (ICML)*, pp. 2014–2023, 2016.

[9] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural network architectures for signals supported on graphs," *IEEE Transactions on Signal Processing*, vol. 67, no. 4, pp. 1034–1049, 2018.

[10] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[11] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[12] L. Stankovic, D. P. Mandic, M. Dakovic, I. Kisil, E. Sejdic, and A. G. Constantinides, "Understanding the basis of graph signal processing via an intuitive example-driven approach [lecture notes]," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 133–145, 2019.

[13] L. Stanković, D. P. Mandic, M. Daković, M. Brajović, B. Scalzo, S. Li, and A. G. Constantinides, "Data analytics on graphs Part II: Signals on graphs," *Foundations and Trends® in Machine Learning*, vol. 13, no. 3, pp. 157–331, 2020.

[14] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings of the IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 729–734, 2005.

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

[16] F. Gama, E. Isufi, G. Leus, and A. Ribeiro, "Graphs, convolutions, and neural networks: From graph filters to graph neural networks," *IEEE Signal Processing Magazine*, vol. 37, no. 6, pp. 128–138, 2020.

[17] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs," *IEEE Transactions on Signal Processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

Understanding the Basis of Graph Convolutional Neural Networks via an Intuitive Matched Filtering Approach

Ljubiša Stanković, *Fellow, IEEE* and Danilo Mandic, *Fellow, IEEE*

SUPPLEMENTARY MATERIAL

This Supplement complements the main text body of this Lecture Note, for the completeness of the material and to make it self-sufficient.

Notation convention. For convenience of cross-referencing, the equations and figures from the main text body of this Lecture Note will be denoted with the prefix 'LN', for example, (LN-1) refers to equation (1) in the Lecture Note and Fig. LN-1 refers to Fig. 1 in the Lecture Note.

SUPPLEMENT A: GRAPH CONVOLUTION

The perspective of employing convolutions to implement cross-correlations in classical signal analysis [1], [2] will now be extended to graph signals. For completeness, we shall start from the notion of a system on a graph, which underpins the graph convolution (matched filtering) operation.

System on a graph. A system on a graph is defined in analogy to classical finite impulse response (FIR) systems, based on a set of filter coefficients, $\{h\}$, and a graph shift operator, $\mathbf{S}$, in the form given by (LN-9) [3]

$$\mathbf{y} = h_0\mathbf{x} + h_1\mathbf{S}\mathbf{x} + \cdots + h_{M-1}\mathbf{S}^{M-1}\mathbf{x}. \tag{1}$$

For undirected graphs, the graph Laplacian, $\mathbf{L}$, is commonly used as a graph shift operator for systems on a graph. Other graph shift operators may be equally used, such as the adjacency matrix, $\mathbf{A}$, and the normalized versions of the adjacency matrix, ($\mathbf{A}_N = \mathbf{A}/\lambda_{\max}$), and the graph Laplacian ($\mathbf{L}_N = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$). The random walk (diffusion) matrix ($\mathbf{S} = \mathbf{D}^{-1}\mathbf{W}$) is one more possible graph shift operator.

The spectral domain description of a system on a graph is obtained when any form of the graph shift operator, $\mathbf{S}$, is represented in its eigendecomposition form

$$\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1} \quad \text{and} \quad \mathbf{\Lambda} = \mathbf{U}^{-1}\mathbf{S}\mathbf{U}, \tag{2}$$

where $\mathbf{U}$ is the transformation matrix with the eigenvectors as its columns and $\mathbf{\Lambda}$ is a diagonal matrix with the corresponding eigenvalues on the main diagonal (the graph Laplacian, $\mathbf{L}$, is always diagonizable, being a real-valued symmetric matrix). A left-multiplication of the vertex domain relation in (1) by the inverse transformation matrix, $\mathbf{U}^{-1}$, yields

$$\mathbf{U}^{-1}\mathbf{y} = h_0\mathbf{U}^{-1}\mathbf{x} + h_1\mathbf{U}^{-1}\mathbf{S}\mathbf{x} + \cdots + h_{M-1}\mathbf{U}^{-1}\mathbf{S}^{M-1}\mathbf{x}.$$

Upon expanding, $\mathbf{U}^{-1}\mathbf{S}\mathbf{x} = \mathbf{U}^{-1}\mathbf{S}\mathbf{U}\mathbf{U}^{-1}\mathbf{x} = \mathbf{\Lambda}\mathbf{X}$, we arrive at the spectral domain description of a system on a graph [3]

$$\mathbf{Y} = h_0\mathbf{X} + h_1\mathbf{\Lambda}\mathbf{X} + \cdots + h_{M-1}\mathbf{\Lambda}^{M-1}\mathbf{X}, \tag{3}$$

where

$$\mathbf{X} = \mathbf{U}^{-1}\mathbf{x} \quad \text{and} \quad \mathbf{Y} = \mathbf{U}^{-1}\mathbf{y}$$

are respectively the graph Fourier transforms (GFT) of the graph signals $\mathbf{x}$ and $\mathbf{y}$. The transfer function of the graph system in (1), is now obtained from

$$\mathbf{Y} = H(\mathbf{\Lambda})\mathbf{X},$$

to arrive at a diagonal graph transfer function matrix defined by

$$H(\mathbf{\Lambda}) = h_0 + h_1\mathbf{\Lambda} + \cdots + h_{M-1}\mathbf{\Lambda}^{M-1}. \tag{4}$$

Filtering and convolving graph signals. The three approaches to filtering (convolutions) of a graph signal using a system transfer function, $G(\mathbf{\Lambda})$, with the elements on the diagonal $G(\lambda_k)$, $k = 1,2,\ldots,N$, are as follows.

(a) The simplest approach is based on the direct use of the GFT, and is performed by:
   (i) Calculating the GFT of the input signal, $\mathbf{X} = \mathbf{U}^{-1}\mathbf{x}$,
   (ii) Producing the output GFT by multiplying $\mathbf{X}$ by $G(\mathbf{\Lambda})$, to yield $\mathbf{Y} = G(\mathbf{\Lambda})\mathbf{X}$,
   (iii) Calculating the output (filtered) signal, $\mathbf{y}$, as the inverse GFT of $\mathbf{Y}$, that is $\mathbf{y} = \mathbf{U}\mathbf{Y}$.

   The result of this operation,

$$\begin{aligned} y(n) = x(n) * g(n) &= \text{IGFT}\{\text{GFT}\{x(n)\}\text{GFT}\{g(n)\}\} \\ &= \text{IGFT}\{X(k)G(\lambda_k)\}, \end{aligned}$$

   is called *the convolution of signals on a graph*.
   However, this procedure quickly becomes *computationally prohibitive for graphs with a large number of vertices*, N, since only the computation of the eigendecomposition in (2) requires at least $\mathcal{O}(N^2)$ operations over N-dimensional vectors and matrices.

(b) A way to avoid the full size transformation matrices for

L. Stanković is with the University of Montenegro, Podgorica, Montenegro. D. P. Mandic is with Imperial College London, London, United Kingdom. Contact e-mail: ljubisa@ucg.ac.me

large graphs is to approximate the filter transfer function, $G(\lambda)$, at the positions of the eigenvalues, $\lambda = \lambda_k$, $k = 1,2,\ldots,N$, by a polynomial, $h_0 + h_1\lambda + h_2\lambda^2 + \cdots + h_{M-1}\lambda^{M-1}$, that is

$$h_0 + h_1\lambda_k + \cdots + h_{M-1}\lambda_k^{M-1} = G(\lambda_k), \quad k = 1,2,\ldots,N. \tag{5}$$

The resulting system of N equations

$$\mathbf{V}\mathbf{h} = \text{diag}\{G(\mathbf{\Lambda})\}, \tag{6}$$

is solved in the least squares sense for $M < N$ unknown parameters of the system, $\mathbf{h} = [h_0, h_1, \ldots, h_{M-1}]^T$, with a given M and

$$\text{diag}\{G(\mathbf{\Lambda})\} = [G(\lambda_1), G(\lambda_2), \ldots, G(\lambda_N)]^T$$

as the column vector of diagonal elements of $G(\mathbf{\Lambda})$. The elements of the matrix $\mathbf{V}$ are $V(k,m) = \lambda_k^m$, $m = 0,1,\ldots,M-1$, $k = 1,2,\ldots,N$ (Vandermonde matrix).

This system can be efficiently solved for a relatively small M [3]. The implementation of the graph filter is then performed in the vertex domain using $h_0, h_1, \ldots, h_{M-1}$ obtained in (1), with $\mathbf{S} = \mathbf{L}$ and the $(M-1)$-neighborhood for every considered vertex. Notice that the relation between the IGFT of $\text{diag}\{G(\mathbf{\Lambda})\}$ and the system coefficients $h_0, h_1, \ldots, h_{M-1}$ is a direct one in the classical DFT case only, while it is more complex in the general graph case [3], as further elaborated in Remark 14 in this Supplement.

For a large M, the solution to the system of equations in (5), for the unknown parameters $h_0, h_1, \ldots, h_{M-1}$, can be *numerically unstable due to large values of* λ_k^{M-1} for large M.

(c) Another way for avoiding the direct GFT calculation in the implementation of graph filters is by approximating the graph system transfer function, $G(\lambda)$, by a polynomial $H(\lambda)$ of a continuous variable λ [4]–[6].

Notice that this approximation does not guarantee that the transfer function, $G(\lambda)$, and its polynomial approximation

$$H(\lambda) = h_0 + h_1\lambda + \cdots + h_{M-1}\lambda^{M-1}$$

will be close at a discrete set of points $\lambda = \lambda_p$, $p = 1,2,\ldots,N$. However, the maximum absolute deviation of this polynomial approximation can be kept small using the so called *min-max polynomials* (Chebyshev polynomial approximation of the transfer function $G(\lambda)$ is one such example). After such a polynomial approximation, $H(\lambda)$, the output of the graph system, $\mathbf{Y} = H(\mathbf{\Lambda})\mathbf{X}$, is calculated in the vertex domain using

$$\mathbf{y} = \Big(\sum_{m=0}^{M-1} h_m \mathbf{L}^m\Big)\mathbf{x} = H(\mathbf{L})\,\mathbf{x}.$$

In other words, the calculation of the output signal, $y(n)$, at a vertex, n, is now localized to the input signal sample at the same vertex, n, and its small $(M-1)$-neighborhood. In other words, as desired, in this way there is no need to perform any operation over the whole (possibly very large) graph, as in the GFT approach.

Remark 11: Some of the first-order systems on a graph which are commonly used in the GCNN, are given by (LN-10)-(LN-14) of our earlier Lecture Notes article [7].

Graph transfer function and graph "impulse response". The relation between the transfer function of a system on a graph, $H(\lambda_k)$, and the graph signal (*cf.* impulse response), $h(n)$, in the generalized convolutions

$$y(n) = x(n) \star h(n)$$

is not as straightforward as in classical analysis. This relation can be established based on the definitions of the graph system function, $H(\lambda_k)$, and the corresponding GFT. To this end, consider $H(\lambda_k)$, which is defined in (LN-15) as

$$H(\lambda_k) = h_0 + h_1\lambda_k + \cdots + h_{M-1}\lambda_k^{M-1}. \tag{7}$$

The samples of the graph "impulse response" signal, $h(n)$, are equal to the IGFT of $H(\lambda_k)$, and are by definition given by

$$h(n) = \sum_{k=0}^{N-1} H(\lambda_k) u_k(n). \tag{8}$$

Recall that in the classical system (transfer function) analysis, we have $u_k(n) = \exp(j2\pi nk/N)/\sqrt{N}$ and $\lambda_k = \exp(-j2\pi nk/N)\sqrt{N}$, which results in the coefficients of the transfer functions being equal to the impulse response of the convolution filter, that is

$$h(n) = h_n.$$

This is, however, not the case for systems on graphs. To show this, for notational simplicity and without loss of generality, we assume $M = N$. The vector form of the "impulse response" in (8) is then

$$[h(0),\, h(1), \ldots, h(N-1)]^T = \mathbf{U}H(\mathbf{\Lambda}),$$

while the vector form of the system coefficients in (7) is given by

$$H(\mathbf{\Lambda}) = \mathbf{V}_\lambda [h_0,\, h_1, \ldots, h_{N-1}]^T$$

where $\mathbf{V}_\lambda$ is a Vandermonde matrix whose rows are $[1,\, \lambda_k,\, \lambda_k^2,\, \ldots, \lambda_k^{N-1}]$, $k = 0,1,2,\ldots,N-1$, see also (6).

Using the last two equations, the graph signal, $h(n)$, and the system coefficients, h_n, can now be related via $H(\mathbf{\Lambda})$, as

$$[h(0),\, h(1), \ldots, h(N-1)]^T = \mathbf{U}\mathbf{V}_\lambda [h_0,\, h_1, \ldots, h_{N-1}]^T. \tag{9}$$

Remark 12: Unlike in standard system analysis, the coefficients of a system on a graph, h_n, $n = 1,\ldots,M$, are different (both in terms of their number and physical meaning) from the graph "impulse response" signal, $h(n)$, $n = 1,\ldots,N$, which is obtained through the Inverse Graph Fourier Transform (IGFT) of the graph system function in (8). Fig. 1 illustrates this important point for the graph from Fig. LN-2 and the first-order transfer function, $H(\lambda_k) = 1 + 0.5\lambda_k$. The top panel shows the weights (system coefficients) h_0 and h_1, the middle panel shows the corresponding graph "impulse response" signal, $h(n) = \text{IGFT}(H(\lambda_k))$, along the vertex index axis, and the bottom panel shows $h(n)$ on the original graph [3].

In classical system analysis (the case of a directed circular

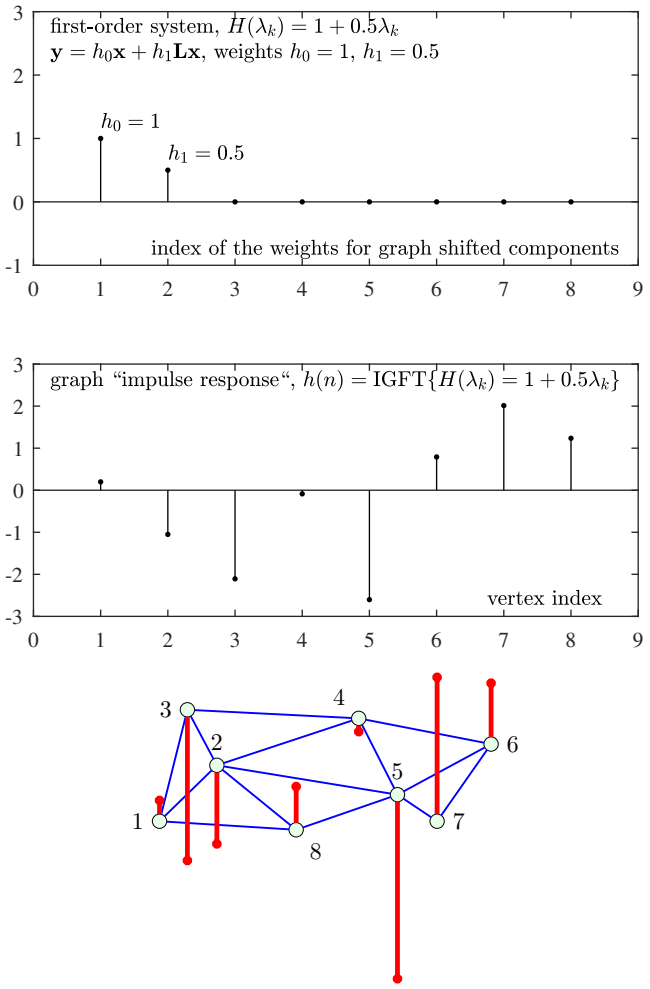

Fig. 1. Representation of a system on a graph. (Top:) The first-order transfer function, $H(\lambda_k) = 1 + 0.5\lambda_k$, for the graph from Fig. LN-2, with the weights (system coefficients) h_0 and h_1. (Middle) The corresponding graph "impulse response" signal, $h(n)$, plotted along the vertex index axis. (Bottom:) The signal $h(n)$ plotted on the original graph from Fig. LN-2. Observe that the graph "impulse response" signal $h(n)$ differs significantly from the classical case, for which the impulse response would be $h(n) = \delta(n) + 0.5\delta(n-1)$.

graph and its adjacency matrix), the graph signal samples, $h(n)$, which are obtained as the inverse DFT of the system (transfer) function $H(\lambda_k)$ and the system coefficients (weights of the shifted/delayed signals), h_n, are the same, since the eigenvalues of the shift operator (adjacency matrix of a directed unweighted circular graph) are equal to the corresponding shift operators in the spectral domain, $\lambda_k = \exp(-j2\pi k/N)/\sqrt{N}$ and $u_k(n) = \exp(j2\pi nk/N)/\sqrt{N} = \lambda_k^{-n}$, with $h_n = h(n)$ and

$$H(\lambda_k) = \sum_{n=0}^{N-1} h(n) e^{-j2\pi nk/N}.$$

Therefore, in classical DFT analysis (signals on a directed circular graph), the following relation holds

$$\mathbf{U}\mathbf{V}_\lambda = \mathbf{I}.$$

which is different from the corresponding relation for the graph system in (9).

SUPPLEMENT B: MAX-POOLING THROUGH GRAPH COARSENING

Graph coarsening is a graph down-sampling strategy which refers to the mechanisms for the reduction in the number of vertices of the original graph [8], [9]. Graph coarsening is typically performed in graph partitioning strategies, and also for the visualization of large graphs in a computationally efficient and intuitive manner [10]. In general, graph coarsening is performed by grouping the vertices into $N_c < N$ nonoverlapping subsets, subsequently forming new vertices by merging the vertices in these subsets, and finally connecting these new "super–vertices" (former groups of vertices) with the new "equivalent" weights, which represent a sum of all weights between the vertex groups. The weight matrix of the so coarsened graph is given by

$$\mathbf{W}_c = \mathbf{P}\mathbf{W}\mathbf{P}^T,$$

with $\mathbf{P}$ as the indicator matrix of these groups of vertices [8].

Example 1. (Max pooling through graph coarsening). *To illustrate the principle of graph coarsening, recall that in the max-pooling operation in the Lecture Note we used the first iteration of the graph signal in the first channel of the considered GCNN, after the ReLU operation (see Supplement D, Table 1), given by* $f(\mathbf{y}_1) = [0.012 \quad 0.037 \quad 0 \quad 0.121 \quad 0 \quad 0 \quad 0 \quad 0.053]^T$ *and shown in Fig. 2 (a). The weight matrix,* $\mathbf{W}$*, for this graph is given by (10). Observe that the maximum signal value is located at the vertex* $n = 4$*. The coarsening of this graph can be performed in the following way:*

1) *Fuse (merge) the vertices* $n = 2,3,5,6$ *which are in the one-neighbourhood of the vertex with the maximum value of the signal (in this case* $n = 4$ *with the signal value of* 0.121*), to form a super-vertex* 2**3**4**5**6*, which inherits the signal value of* $f(y_1(4)) = 0.121$*. Based on the set of weights,* $\mathbf{W}$ *in (10), the creation of this super-vertex is described by the second row of the indicator matrix,* $\mathbf{P}$ *in (11).*
2) *The remaining graph from Fig. 2 (a) now has* 4 *vertices, the super-vertex n=2**3**4**5**6, and the vertices n=1,7,8, with the corresponding signal values* $[0.012, 0, 0.053]$*. The maximum signal value of* 0.053 *at the remaining original vertices is located at vertex* $n = 8$*, and this vertex is therefore fused with the vertices within its one-neighbourhood (in this case only vertex n=1), to form the second super-vertex 1**8**, with the signal value of* 0.053*. This is described by the first row of the indicator matrix,* $\mathbf{P}$*, in (11).*
3) *Finally, the only remaining original vertex is now vertex* $n = 7$*, which becomes the last super-vertex and keeps its original signal value of* 0*. The process is described by the third row of the indicator matrix,* $\mathbf{P}$ *in (11).*

In this way, we have coarsened the original 8-vertex graph from Fig. 2 (a) into a resulting 3-vertex graph in Fig. 2 (b) which corresponds to the max-pooling operation *on graphs. The numerical values of the matrices used in graph coarsening are given below.*

In order to allocate new weights to the coarsened graph in Fig. 2 (b), it is important to realise that the weight matrix, $\mathbf{W}_c = \mathbf{P}\mathbf{W}\mathbf{P}^T$ *in (12), is calculated for the three new super-vertices 1**8**, 2**3**4**5**6, and 7. Observe that the non-zero values on the diagonal of* $\mathbf{W}_c$ *correspond to self-loops at the corresponding graph vertices, as shown in the*

final coarsened graph in Fig. 2(c).

$$\mathbf{W} = \begin{matrix}1\\2\\3\\4\\5\\6\\7\\8\end{matrix}\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}\begin{matrix}\\\\\\\\\\\\\\\\\end{matrix}, \quad (10)$$
$$\begin{matrix}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8\end{matrix}$$

$$\mathbf{P} = \begin{matrix}1,8\\2,3,4,5,6\\7\end{matrix}\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}\begin{matrix}\\\\\\\\\\\\\\\\\end{matrix}, \quad (11)$$
$$\begin{matrix}1 & 2 & 3 & 4 & 5 & 6 & 7 & 8\end{matrix}$$

$$\mathbf{W}_c = \mathbf{P}\mathbf{W}\mathbf{P}^T = \begin{matrix}1,8\\2,3,4,5,6\\7\end{matrix}\begin{bmatrix} 2 & 4 & 0 \\ 4 & 14 & 2 \\ 0 & 2 & 0 \end{bmatrix}\begin{matrix}\\\\\\\\\\\\\\\\\end{matrix}. \quad (12)$$
$$\begin{matrix}1,8 & 2,3,4,5,6 & 7\end{matrix}$$

The indicator matrix for the ReLU and max-pooling stages *indicates the values of the signal which keep their nonzero values after the ReLU and the max-pooling (MP) operations, and is given by*

$$\mathbf{M}^{ReLU+MP} = \begin{bmatrix} 0 & 0 & 0 & \boldsymbol{1} & 0 & 0 & \boldsymbol{0} & \boldsymbol{1} \\ 0 & \boldsymbol{1} & 0 & 0 & 0 & 0 & \boldsymbol{1} & 0 \end{bmatrix}^T. \quad (13)$$

The first row shows that, for the first channel (convolution filter, kernel), the signal values at the vertices $n = 4$ and $n = 8$ have "survived" both the ReLU and the max-pooling operation. In a similar way, the signal values at the vertices, $n = 7$ and $n = 2$ have "survived" in the second channel. Importantly, when max-pooling is used, the above indicator matrix is employed during the learning process to reposition the gradient updates to their correct positions, as elaborated in Supplement D. Fig. 3 illustrates the max-pooling operation for the GCNN from Fig. LN-4 in the article. For more detail on the implementation, we refer to our sister paper [1].

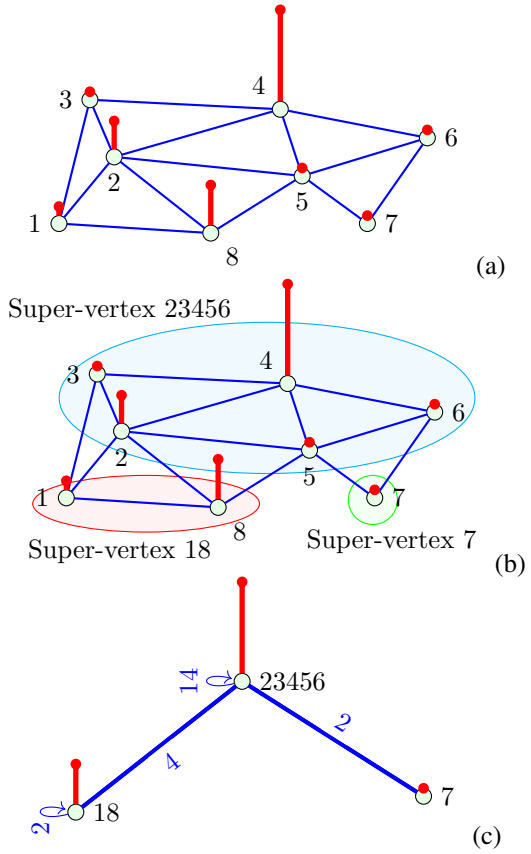

Fig. 2. Illustration of the max–pooling operation in a graph CNN, when implemented through graph coarsening. (a) The considered graph signal. (b) The coarsening process for the graph from a), whereby the creation of super-vertices (max-pooling) is performed for: i) graph signal at the vertex $n = 4$ with its one-neighborhood, $n = 2,3,5,6$, ii) graph signal at the vertex n=8 with its one-neighbourhood, n=1, and iii) remaining graph signal at $n = 7$. c) The resulting coarsened graph, where the three super-vertices are $n = 23\mathbf{4}56$, $n = \mathbf{18}$, and n=**7**, while the values of weights, $\mathbf{W}_c$ in (12), in the self-loops (on the diagonal of (12)) and between the super-vertices, are given in blue.

Remark 13: In the process of graph coarsening, described by (10)–(12), the graph topology changes for each channel and at each iteration, since the operation is signal dependent.

Graph lifting (uncoarsening). Graph lifting is an inverse operation to graph coarsening, and represents a process of obtaining a larger scale (fine) graph from a coarsened (smaller) graph. The weight matrix, $\mathbf{W}_L$, of the lifted graph is obtained from the weight matrix of the coarsened graph, $\mathbf{W}_c$, as

$$\mathbf{W}_L = \mathbf{P}^+\mathbf{W}_c(\mathbf{P}^+)^T,$$

where $\mathbf{P}^+$ is the pseudo-inverse of the indicator matrix, such that $\mathbf{P}\mathbf{P}^+ = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

The same relations as for the normalized weights in (12) hold for the corresponding graph Laplacian of the original graph, $\mathbf{L}$, graph Laplacian of the coarsened graph, $\mathbf{L}_c$, and the graph Laplacian of the lifted graph, $\mathbf{L}_L$, that is

$$\mathbf{L}_c = \mathbf{P}\mathbf{L}\mathbf{P}^T,$$
$$\mathbf{L}_L = \mathbf{P}^+\mathbf{L}_c(\mathbf{P}^+)^T.$$

Notice that for the normalized graph Laplacian, the definition of the indicator matrix should be slightly modified [11].

Generalization of graph coarsening. The process of coarsening a graph $\mathcal{G}$ (with vertices $\mathcal{V}$, edges $\mathcal{B}$, and weights $\mathcal{W}$) may be continued until a desired number of vertices is obtained. In general, the coarsening operation involves a sequence of graphs

$$\mathcal{G} = \mathcal{G}_0 = \{\mathcal{V},\mathcal{B},\mathcal{W}\} = \{\mathcal{V}_0,\mathcal{B}_0,\mathcal{W}_0\}$$
$$\mathcal{G}_1 = \{\mathcal{V}_1,\mathcal{B}_1,\mathcal{W}_1\}$$
$$\vdots$$
$$\mathcal{G}_c = \{\mathcal{V}_c,\mathcal{B}_c,\mathcal{W}_c\},$$

whereby at every iteration, the coarsened graph, $\mathcal{G}_{l+1} = \{\mathcal{V}_{l+1},\mathcal{B}_{l+1},\mathcal{W}_{l+1}\}$, is obtained from the previous one through a weight matrix transformation based on the corresponding

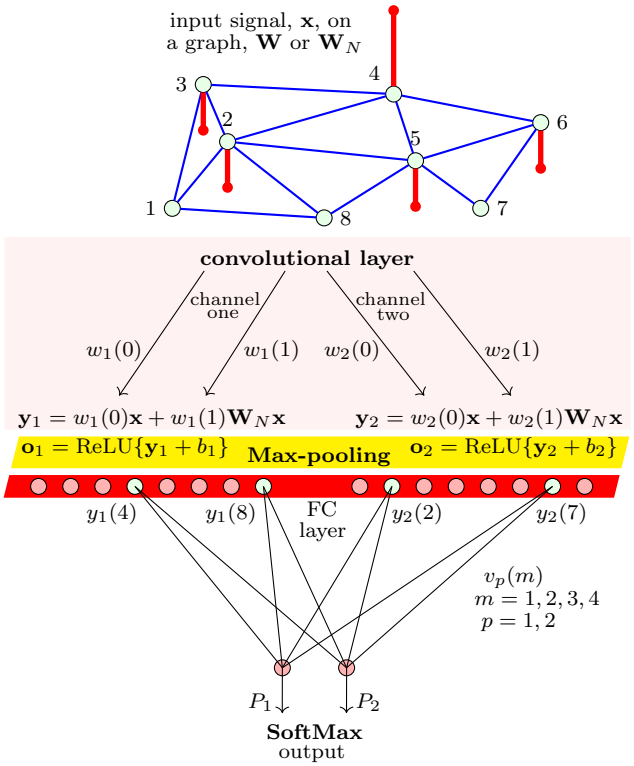

Fig. 3. Principle of max-pooling in GCNNs, which corresponds to selecting the signal values through the indicator matrix in (13). The considered GCNN comprises one graph convolutional layer and one fully connected (FC) output layer with two SoftMax neurons.

indicator matrices, as in (12), that is

$$\mathbf{W}_l = \mathbf{P}_l \mathbf{W}_{l-1} \mathbf{P}_l^T,$$

The inverse operation is referred to as *graph lifting*, and is performed as $\mathbf{W}_{l-1} = \mathbf{P}_l^+ \mathbf{W}_l (\mathbf{P}_l^+)^T$.

SUPPLEMENT C: UPDATING GRAPH CONVOLUTIONAL WEIGHTS: THE GRAPH BACKPROPAGATION

The initial parameters (weights) of a GCNN are typically updated in a supervised manner through a gradient-based learning process known as the backpropagation (BP). At each iteration of the BP algorithm, the gradient values (sensitivities) are computed for each network parameter – the weights in the convolutional layers, the weights in the fully-connected layers, and the biases. These sensitivities are then used to backpropagate the output errors in the process of iteratively updating all the GCNN parameters, until a certain stopping criterion is met or the training data set is exhausted [12]–[14].

1) Initialization: Unlike standard adaptive systems where the initial weight values are typically set to zero, the initial values of the weights in neural networks are usually set to random (and different) values for each convolution channel and network layer. Since, in general, graph convolutional weights, $w_k(v)$, multiply M input signal values at each convolution channel (for the set of the considered input neurons of the layer), the only requirement is that the choice of the initial weights preserves the expected energy of the output for the considered layers. This is achieved, for example, if the initial weights are *Gaussian distributed*, with

$$w_k(v) \sim \sqrt{\frac{2}{M}} \mathcal{N}(0,1), \quad v = 0,1,\ldots,M-1, \quad k = 1,2,\ldots,K. \tag{14}$$

The factor of 2 is used since the ReLU activation function will remove negative output values, which on the average accounts for a half of the expected energy.

Another possibility is to use *uniformly* distributed initial weights, $w_k(v)$, whereby the sum of M initial weights, $\sum_{v=0}^{M-1} w_k(v)$, is also a random variable with unit variance. Such uniformly distributed random weights are defined on the interval

$$w_k(v) \sim \Big[-\sqrt{\frac{6}{M}}, \sqrt{\frac{6}{M}} \Big],$$

and their variance is $\text{Var}\{w_k(v)\} = \frac{6}{M}\frac{1}{3} = \frac{2}{M}$. In this way, the variance of a sum of M values, divided by 2 (to account for the ReLU activation function), will produce a unit weight variance. Such initial weights are referred to as the *He initial values*.

We can use the same principles as above to generate the initial weights in the fully connected layers, just with the number of weights in the convolutional layer now as KN instead of M. The values of the initial weights in (14) can be further reduced based on the number of output neurons in the considered layer, S, to yield the *Xavier initial values*, given by

$$v_p(m) \sim \sqrt{\frac{2}{NK+S}} \mathcal{N}(0,1).$$

2) Backpropagation in a two-layer GCNN: The principle of adaptive learning in general GCNNs will be next illustrated through an example of weight update within a simple GCNN, shown in Fig. LN-4 of the Lecture Note, which comprises one convolutional layer and one fully connected output layer.

Convolutional layer. For the input graph signal, $\mathbf{x} = [x(1),\ x(2),\ldots,x(N)]^T$, the output signal of the convolutional layer of the GCNN, with K convolution filters (kernels) of width $M = 2$, is given by

$$\mathbf{y}_k = w_k(0)\mathbf{x} + w_k(1)\mathbf{W}_N\mathbf{x},$$

or in an element-wise form for the channels $k = 1,2,\ldots,K$

$$y_k(n) = w_k(0)x(n) + w_k(1)\sum_{\mu} W_N(n,\mu)x(\mu), \tag{15}$$

where $W_N(n,\mu)$ are the elements of the normalized weight matrix, $\mathbf{W}_N$. The overall output of the convolutional layer is then obtained after the bias term, b_k, is included and upon the application of the ReLU activation function, $f(\cdot)$, to yield

$$o_k(n) = f\Big(y_k(n) + b_k\Big). \tag{16}$$

For simplicity, we assumed that no max-pooling or any other down-sampling was performed.

The output from the convolutional layer is then reshaped (flattened) into a vector of length KN, which serves as input to the fully connected (FC) layer with S outputs (for clarity, we considered only one FC layer). Each of the KN nodes of the output of the convolutional layer, $o_F(m)$, with the

corresponding samples

$$[o_1(1),\ldots,o_1(N),\ o_2(1),\ldots,o_2(N),\ldots,o_K(1),\ldots,o_K(N)]^T$$

is connected to each of the S nodes of the fully connected output layer, through the weights, $v_p(m)$, $p=1,2,\ldots,S$, to produce the overall GCNN output in the form

$$\begin{aligned} z_p &= v_p(0)o_1(1)+v_p(1)o_1(2)+\cdots+v_p(N-1)o_1(N)\\ &\quad +v_p(N)o_2(1)+\quad\ \ldots\quad\ +v_p(2N-1)o_2(N)\\ &\quad \vdots\\ &\quad +v_p((K-1)N)o_K(1)+\cdots+v_p(KN-1)o_K(N). \end{aligned} \tag{17}$$

Note that the number of weights, $v_p(m)$, $p=1,2,\ldots,S$, $m=1,2,\ldots,KN$ in the FC layer is $S\times KN$.

A commonly used *loss function* in the BP algorithm is the mean square error (MSE) between the network output, z_p, and the true label, t_p, given by

$$\mathcal{L}=\frac{1}{S}\sum_{p=1}^{S}(z_p-t_p)^2, \tag{18}$$

where t_p denotes the desired or target output (also called the teaching signal) [12].

Training process in the convolutional layer. To define the gradient descent relations for the update of all previous weights (within both the graph convolutional layer, $w_k(v)$, and the fully connected layer, $v_p(m)$) in the training process, consider first the convolutional layer, described by (15)-(16), to give the gradient weight update in the form

$$w_k(v)_{new}=w_k(v)_{old}-\alpha\frac{\partial\mathcal{L}}{\partial w_k(v)}\Big|_{w_k(v)=w_k(v)_{old}} \tag{19}$$

where α is a small positive constant known as the step-size or learning rate. The element-wise gradient values in the first (convolutional) layer are designated by the superscript $(\cdot)^{(1)}$, and calculated as

$$g_k^{(1)}(0)=\frac{\partial\mathcal{L}}{\partial w_k(0)}=\sum_n\frac{\partial\mathcal{L}}{\partial y_k(n)}\frac{\partial y_k(n)}{\partial w_k(0)}=\sum_n\frac{\partial\mathcal{L}}{\partial y_k(n)}x(n), \tag{20}$$

$$\begin{aligned} g_k^{(1)}(1)&=\frac{\partial\mathcal{L}}{\partial w_k(1)}=\sum_n\frac{\partial\mathcal{L}}{\partial y_k(n)}\frac{\partial y_k(n)}{\partial w_k(1)}\\ &=\sum_n\frac{\partial\mathcal{L}}{\partial y_k(n)}\sum_\mu W_N(n,\mu)x(\mu), \end{aligned} \tag{21}$$

based on the input-output relation in (15).

Next, the terms $\partial\mathcal{L}/\partial y_k(n)$, also termed the *delta error* function, $\Delta_k^{(1)}(n)$, are calculated using the chain rule, as

$$\begin{aligned} \Delta_k^{(1)}(n)&=\frac{\partial\mathcal{L}}{\partial y_k(n)}=\sum_p\frac{\partial\mathcal{L}}{\partial z_p}\frac{\partial z_p}{\partial y_k(n)}=\sum_p\frac{\partial\mathcal{L}}{\partial z_p}\frac{\partial z_p}{\partial o_k(n)}\frac{\partial o_k(n)}{\partial y_k(n)}\\ &=\sum_p\Delta_p^{(2)}v_p((k-1)N+n-1)\ u(y_k(n)) \end{aligned} \tag{22}$$

where, according to (16) and (LN-34), $\partial o_k(n)/\partial y_k(n)=u(y_k(n))$, while the relation in (17) is used for the calculation of $\partial z_p/\partial o_k(n)=v_p((k-1)N+n-1)$, with

$$\Delta_p^{(2)}=\frac{\partial\mathcal{L}}{\partial z_p}=z_p-t_p,\quad p=1,2,\ldots,S$$

as the delta error in the final (the second, in this case) stage.

The relation in (22) back-propagates the error from layer 2, denoted by $\Delta_p^{(2)}$, to layer 1, to give the portion of the overall error attributed to the neuron k of layer 1, denoted by $\Delta_k^{(1)}(n)$. We can now calculate $\partial\mathcal{L}/\partial y_k(n)=\Delta_k^{(1)}(n)$ and the gradient for the update in (19).

Remark 14: The values of the gradients $g_k^{(1)}(0)$ and $g_k^{(1)}(1)$ in (20) and (21), for the update of the convolutional (*matched filter*) weights, $w_k(m)$, can now be expressed as

$$\begin{aligned} g_k^{(1)}(0)&=\sum_n\frac{\partial\mathcal{L}}{\partial y_k(n)}x(n)=\sum_n\Delta_k^{(1)}(n)x(n),\quad\text{and}\\ g_k^{(1)}(1)&=\sum_n\Delta_k^{(1)}(n)\sum_\mu W_N(n,\mu)x(\mu), \end{aligned} \tag{23}$$

or in a compact vector/matrix form

$$\mathbf{g}_k^{(1)}=[\mathbf{x}^T\boldsymbol{\Delta}_k^{(1)},\ \ (\mathbf{W}_N\mathbf{x})^T\boldsymbol{\Delta}_k^{(1)}]^T=\begin{bmatrix}\mathbf{x}^T\\(\mathbf{W}_N\mathbf{x})^T\end{bmatrix}\boldsymbol{\Delta}_k^{(1)}. \tag{24}$$

This expression can be straightforwardly generalized to cater for higher-order matched filters. For example, for $M=3$ and based on (LN-32), the gradient vector will become

$$\mathbf{g}_k^{(1)}=[\mathbf{x}^T\boldsymbol{\Delta}_k^{(1)},\ \ (\mathbf{W}_N\mathbf{x})^T\boldsymbol{\Delta}_k^{(1)},\ \ (\mathbf{W}_N^2\mathbf{x})^T\boldsymbol{\Delta}_k^{(1)}]^T. \tag{25}$$

From (23), (24), and (25) it can be concluded that, in general, the elements of the gradient, $g_k^{(1)}(m)$, are obtained from

$$g_k^{(1)}(m)=\sum_n\Delta_k^{(1)}(n)\mathcal{D}^m\{x(n)\}, \tag{26}$$

where $\mathcal{D}^m\{x(n)\}$ is a graph signal which is shifted by m steps. A vector form of the signal shifted for m steps, $\mathcal{D}^m\{x(n)\}$, is $(\mathbf{W}_N^m\mathbf{x})^T$. In classical systems, the shifted version of the signal is given by $\mathcal{D}^m\{x(n)\}=x(n+m)$ and relation (26) reduces to the classical convolution (cross-correlation) form. Therefore, (26) can be understood as the graph convolution (cross-correlation) of the graph shifted input signal, $\mathcal{D}^m\{x(n)\}$, and the delta error signal, $\Delta_k^{(1)}(n)$. This is conformal with the cross-correlation concept of matched filters.

This means that for the M-th order system, in an ideal case (after the GCNN is sufficiently trained) the gradients, $g_k^{(1)}(m)$, should be equal to zero, whereby the delta error signal, $\Delta_k^{(1)}(n)$, should be normal to the graph input signal, $x(n)$, and all its graph shifts, $\mathcal{D}^m\{x(n)\}$, up to the $(M-1)$th shift.

The bias terms are updated in the same way as the weights in (19), that is, based on

$$b_{k,new}=b_{k,old}-\beta\frac{\partial\mathcal{L}}{\partial b_k}\Big|_{b_k=b_{k,old}} \tag{27}$$

and

$$\frac{\partial\mathcal{L}}{\partial b_k}=\sum_n\frac{\partial\mathcal{L}}{\partial y_k(n)}\frac{\partial y_k(n)}{\partial b_k}=\sum_n\frac{\partial\mathcal{L}}{\partial y_k(n)}=\sum_n\Delta_k^{(1)}(n). \tag{28}$$

Fully connected (FC) layer. The input to the FC layer represents the flattened output from the convolutional layer, given by

$$o_F((k-1)N+n) = o_F(m),$$

where the indices m in $o_F(m)$ range from 1 to KN, with $k = 1,2,\ldots,K$ and $n = 1,2,\ldots,N$. Notice that the relation (17) can be equally written as

$$z_p = \sum_{m=1}^{KN} v_p(m-1)o_F(n).$$

The update of the fully connected layer weights, $v_p(m)$, is then performed in the same way as in (19), based on

$$v_p(m)_{new} = v_p(m)_{old} - \gamma \frac{\partial \mathcal{L}}{\partial v_p(m)} \Big|_{v_p(m)=v_p(m)_{old}}, \quad (29)$$

with the gradient elements in the form

$$g_p^{(2)}(m) = \frac{\partial \mathcal{L}}{\partial v_p(m)} = \frac{\partial \mathcal{L}}{\partial z_p}\frac{\partial z_p}{\partial v_p(m)} = (z_p - t_p)o_F(m) = \Delta_p^{(2)} o_F(m),$$

and with γ as the step-size.

If a nonlinear activation function is used at the output, the factor of $f'(z_p)$ will correspondingly multiply the right hand side of $\partial\mathcal{L}/\partial v_p(m)$.

3) SoftMax Output Layer: In some applications, it is desirable that the output layer gives the probabilities for the decision when performing classification tasks. In other words, such an output represents the probabilities of different possible outcomes (labels) which are associated with the analyzed signals or images (for example, dog, cat, bird in an image), whereby the label that receives the highest probability represents the overall classification decision. In the error calculation, the desired (target) output then assumes the value $t_p = 1$ for the highest value of the output probability, say $p = p_0$ (in the supervised training process, we have the knowledge of the signal/image which is analyzed by the GCNN) and $t_p = 0$ for all other lower probability values $p = 1,2,\ldots,S, p \neq p_o$.

Given that the probabilities are positive, while the output, z_p, from the last GCNN layer (the overall output) may assume various positive and negative real values, it is necessary to map the overall GCNN output, z_p, onto a probability-like range of positive values. This is typically achieved through a mapping of the form

$$P_p = \frac{e^{z_p}}{\sum_{i=1}^{S} e^{z_i}}, \quad p = 1,2,\ldots,S. \quad (30)$$

called the SoftMax. Obviously, $0 \leq P_p \leq 1$ and $\sum_{p=1}^{S} P_p = 1$.

When the SoftMax is used as the output mapping, the loss function is modified accordingly, that is, instead of the mean square error it assumes the cross-entropy form, given by

$$\mathcal{L} = -\sum_{p=1}^{S} t_p \ln(P_p).$$

Consider the case with $S = 2$. Physically, the cross-entropy should be zero (and no update of the weights should be performed) when, for example, $\mathbf{t} = [t_1,\ t_2]^T = [1,\ 0]^T$, and the value of corresponding probability, P_p, is such that $P_1 = 1$ and $P_2 = 0$, producing $\mathcal{L} = -1\ln(1) - 0\ln(0) = 0$. Conversely, the cross-entropy is very large if there is a target, for example, at p=1, that is $t_1 = 1$, and the corresponding output probability P_1 is small, so that $\mathcal{L} = -1\ln(0) - 0\ln(1) \to \infty$, thus indicating a big change in the weights. In practical GCNN training, we are always between these two theoretical extrema.

It is now straightforward to show that with cross-entropy as the cost function, the delta error function in the output layer is of the form

$$\Delta_p^{(2)} = \frac{\partial \mathcal{L}}{\partial z_p} = \sum_{i=1}^{S} \frac{\partial \mathcal{L}}{\partial P_i}\frac{\partial P_i}{\partial z_p} = \sum_{i=1}^{S} \left(\frac{t_i}{P_i} P_i P_p\right) - \frac{t_p}{P_p} P_p = P_p - t_p$$

since from (30) it follows that $\partial P_i/\partial z_p = -P_i P_p$ if $i \neq p$ and $\partial P_i/\partial z_p = P_i(1-P_p) = -P_i P_p + P_p$ if $i = p$, while $\sum_{i=1}^{S} t_i = 1$.

Therefore, as desired, there is no weight correction for the correct GCNN decision, $t_p = P_p$, while all other relations within the backpropagation algorithm do hold, without any modification.

Supplement D: A step-by-step calculation in the first iteration of the GCNN from Example LN-3

A step-by-step quantification of all the intermediate results in the forward pass, together with the backpropagation learning procedures is given in the Table 1 below (see the next page).

References

[1] L. Stankovic and D. Mandic, "Convolutional neural networks demystified: A matched filtering perspective based tutorial," *arXiv preprint arXiv:2108.11663*, 2021.

[2] L. Stanković, *Digital Signal Processing with Selected Topics*. CreateSpace Independent Publishing Platform, An Amazon.com Company, 2015.

[3] L. Stanković, D. P. Mandic, M. Daković, M. Brajović, B. Scalzo, S. Li, and A. G. Constantinides, "Data analytics on graphs Part II: Signals on graphs," *Foundations and Trends® in Machine Learning*, vol. 13, no. 3, pp. 157–331, 2020.

[4] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "The spectral graph wavelet transform: Fundamental theory and fast computation," in *Vertex-Frequency Analysis of Graph Signals* (L. Stanković and E. Sejdić, eds.), pp. 141–175, Springer, 2019.

[5] H. Behjat and D. Van De Ville, "Spectral design of signal-adapted tight frames on graphs," in *Vertex-Frequency Analysis of Graph Signals* (L. Stanković and E. Sejdić, eds.), pp. 177–206, Springer, 2019.

[6] L. Stanković, D. Mandic, M. Daković, B. Scalzo, M. Brajović, E. Sejdić, and A. G. Constantinides, "Vertex-frequency graph signal processing: A comprehensive review," *Digital Signal Processing*, p. 102802, 2020.

[7] C.-C. J. Kuo, "The CNN as a guided multilayer RECOS transform [lecture notes]," *IEEE Signal Processing Magazine*, vol. 34, no. 3, pp. 81–89, 2017.

[8] L. Stanković, D. Mandic, M. Daković, M. Brajović, B. Scalzo, S. Li, A. G. Constantinides, *et al.*, "Data analytics on graphs Part I: Graphs and spectra on graphs," *Foundations and Trends® in Machine Learning*, vol. 13, no. 1, pp. 1–157, 2020.

[9] L. Stanković, D. Mandic, M. Daković, M. Brajović, B. Scalzo, S. Li, A. G. Constantinides, *et al.*, "Data analytics on graphs Part III: Machine learning on graphs, from graph topology to applications," *Foundations and Trends® in Machine Learning*, vol. 13, no. 4, pp. 332–530, 2020.

[10] N. Tremblay and A. Loukas, "Approximating spectral clustering via sampling: A review," in *Sampling Techniques for Supervised or Unsupervised Tasks*, pp. 129–183, Springer, 2020.

[11] Y. Jin, A. Loukas, and J. JaJa, "Graph coarsening with preserved spectral properties," in *Proc. International Conference on Artificial Intelligence and Statistics*, pp. 4452–4462, 2020.

[12] D. Mandic and J. Chambers, *Recurrent neural networks for prediction: Learning algorithms, architectures and stability*. Wiley, 2001.

[13] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings of the IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 729–734, 2005.

[14] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 5680–5695, 2020.

TABLE 1: A STEP-BY-STEP CALCULATION OF ALL THE STEPS IN THE FORWARD PASS AND THE BACKPROPAGATION, FOR THE GCNN FROM EXAMPLE LN-3

FORWARD CALCULATIN: From the input signal to the output

FW1: Input signal, x, of length $N=8$,
$\mathbf{x}=[0.087 \quad 0.030 \quad -0.006 \quad 0.039 \quad -0.254 \quad -0.426 \quad 0.946 \quad -0.145]^T$.
The normalized weight matrix, $\mathbf{W}_N$ given in (LN-26), was used as the graph shift operator
The target signal was $\mathbf{t}=[1 \quad 0]^T$, since the $\mathbf{feature}_1=\mathbf{x}_0-\mathbf{W}_N\mathbf{x}_0=[0,0,0,0,-0.316,-0.408,\ 1,\ 0]^T$
was present in the input (corrupted by small levels of noise). This feature was obtained using (LN-36) and $x_0(n)=\delta(n-7)$.

FW2: Weight random initialization (convolutional layer): $w_k(v)\sim\mathcal{N}(0,1)\sqrt{2/M}$, $M=2$, for $K=2$ channels, $\mathbf{w}_k=[w_k(0),\ w_k(1)]^T$, to yield
$\mathbf{w}_1=[-0.221 \quad -0.741]^T$,
$\mathbf{w}_2=[\ 1.429 \quad 0.323]^T$.

FW3: Graph convolutions: $\mathbf{y}_k=\mathbf{x}\star\mathbf{w}_k+b_k=w_k(0)\mathbf{x}+w_k(1)\mathbf{W}_N\mathbf{x}$, $k=1,2$, with the initial bias values $b_k=0$.
$\mathbf{y}_1=[\ 0.012 \quad 0.037 \quad -0.034 \quad 0.121 \quad -0.067 \quad -0.152 \quad -0.021 \quad 0.053]^T$,
$\mathbf{y}_2=[\ 0.111 \quad 0.024 \quad 0.007 \quad -0.001 \quad -0.309 \quad -0.501 \quad 1.270 \quad -0.217]^T$.

FW4: Nonlinear activation function: The ReLU activation function, $f(\mathbf{y}_k)=\max\{0,\mathbf{y}_k\}$, was used, to give
$f(\mathbf{y}_1)=[0.012 \quad 0.037 \quad 0 \quad 0.121 \quad 0 \quad 0 \quad 0 \quad 0.053]^T$
$f(\mathbf{y}_2)=[0.011 \quad 0.024 \quad 0.007 \quad 0 \quad 0 \quad 0 \quad 1.270 \quad 0]^T$.

FW5: Indicator matrix: The indicator matrix of the output (nonzero) values from the ReLU, denoted by $\mathbf{M}^{ReLU}$, assumed the form
$\mathbf{M}^{ReLU}=\begin{bmatrix}\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1}\\ \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0\end{bmatrix}^T$.
It will be used to reposition the gradient updates to their correct positions if the downsampled (e.g. coarsened) graph signal was used, that is, it caters for the effect of the zeroing of negative inputs to the ReLU.

FW6: Flattening of the output of the "convolution-activation-pooling" chain, to give $o_F((k-1)N+n)=f(y_k(n))$, $k=1,2$, $n=1,2,3,4\ldots,8$.
$\mathbf{o}_F=[0.012,\ 0.037,\ 0,\ 0.121,\ 0,\ 0,\ 0,\ 0.053,\ 0.011,\ 0,\ 0.024,\ 0.007,\ 0,\ 0,\ 0,\ 1.270,\ 0]^T$,

FW7: Weight random initialization (FC layer): $v_p(m)\sim\mathcal{N}(0,1)\sqrt{2/(NK)}=\mathcal{N}(0,1)\sqrt{1/8}$,
$\mathbf{v}=\begin{bmatrix}-0.045, & 0.391, & -0.289, & 0.123, & 0.309, & 0.029, & 0.121, & -0.132, & -0.389, & 0.081, & -0.055, & -0.609, & -0.183, & -0.765, & 0.277, & 0.174\\ -0.248, & 0.023, & -0.085, & 0.543, & 0.102, & -0.548, & -0.542, & -0.360, & 0.706, & 0.412, & -0.590, & -0.714, & -0.445, & 0.102, & 0.245, & -0.226\end{bmatrix}^T$.

FW8: Output of the FC layer: The output signal of the FC layer (overall output the GCNN) is given by $z_p=\sum_{m=1}^{16}o_F(m)v_p(m-1)$.
For the considered GCNN from Fig. LN-4, we therefore have $\mathbf{z}=[z_1,\ z_2]^T=\mathbf{v}^T\mathbf{o}_F=[0.332 \quad 0.440]^T$.

FW9: Softmax: With $S=2$ output values, $P_p=e^{z_p}/(e^{z_1}+e^{z_2})$, $p=1,2$, we have
$\mathbf{P}=[P_1,\ P_2]=[0.4731 \quad 0.5269]^T$.

BACK-PROPAGATION: Delta error, gradient, weight updates

BP1: Output Delta error: $\mathbf{\Delta}^{(2)}=[\Delta_1^{(2)},\ \Delta_2^{(2)}]^T=\mathbf{P}-\mathbf{t}=[-0.5269 \quad 0.5269]^T$, $\Delta_p^{(2)}=P_p-t_p$

BP2: Gradient: For the FC layer weights update, $g_p^{(2)}(m)=\Delta_p^{(2)}o_F(m)$; $o_F(m)$ is the input to the FC layer and $\Delta_p^{(2)}$ is the output Delta error
$\mathbf{g}^{(2)}=\mathbf{o}_F(\mathbf{\Delta}^{(2)})^T=\begin{bmatrix}-0.006, & -0.019, & 0, & -0.064, & 0, & 0, & 0, & -0.028, & -0.058, & -0.013, & -0.004, & 0, & 0, & 0, & -0.669, & 0\\ 0.006, & 0.019, & 0, & 0.064, & 0, & 0, & 0, & 0.028, & 0.058, & 0.013, & 0.004, & 0, & 0, & 0, & 0.669, & 0\end{bmatrix}^T$.

BP3: Weight update in the FC layer using the gradient $\mathbf{g}^{(2)}$ and the stepsize $\alpha=0.1$ becomes, $\mathbf{v}\leftarrow\mathbf{v}-0.1\mathbf{g}^{(2)}$,
$\mathbf{v}=\begin{bmatrix}-0.044, & 0.394, & -0.289, & 0.132, & 0.309, & 0.029, & 0.121, & -0.128, & -0.380, & 0.082, & -0.054, & -0.609, & -0.183, & -0.765, & 0.378, & 0.174\\ -0.249, & 0.021, & -0.085, & 0.534, & 0.102, & -0.548, & -0.542, & -0.364, & 0.697, & 0.410, & -0.591, & -0.714, & -0.445, & 0.102, & 0.145, & -0.226\end{bmatrix}^T$
The unchanged weights in this iteration (in red) are defined by the zero-valued input, $o_F(m)$, to the FC layer (concatenated matrix $\mathbf{M}^{ReLU}$).

BP4: Delta error backpropagation from the output, $\Delta_p^{(2)}$, to the convolutional layer, $\Delta_k^{(1)}(n)=\sum_p\Delta_p^{(2)}v_p((k-1)N+n-1)\ u(y_k(n))$,
$\mathbf{\Delta}_1^{(1)}=[-0.108 \quad -0.197 \quad 0 \quad 0.211 \quad 0 \quad 0 \quad 0 \quad -0.125]^T$,
$\mathbf{\Delta}_2^{(1)}=[\ 0.567 \quad 0.173 \quad -0.283 \quad 0 \quad 0 \quad 0 \quad -0.123 \quad 0]^T$. Notice that the elements $u(y_k(m))$ in (22) are defined by $\mathbf{M}^{ReLU}$ in FW5.

BP5: Gradient for the weight update in the convolutional layer, $\mathbf{g}_k^{(1)}=[\mathbf{x}^T\mathbf{\Delta}_k^{(1)},\ (\mathbf{W}_N\mathbf{x})^T\mathbf{\Delta}_k^{(1)}]^T$, $k=1,2$,
$\mathbf{g}_1^{(1)}=[\ 0.011 \quad -0.017]^T$,
$\mathbf{g}_2^{(1)}=[-0.060 \quad -0.017]^T$.

BP6: Weight update in the convolutional layer $\mathbf{w}_k\leftarrow\mathbf{w}_k-0.1\mathbf{g}_k^{(1)}$, $k=1,2$,
$\mathbf{w}_1=[-0.223 \quad -0.739]^T$
$\mathbf{w}_2=[\ 1.437 \quad 0.325]^T$.

BP7: Bias update. According to (28), $b_k\leftarrow b_k-0.05\sum_n\Delta_k^{(1)}(n)$, $k=1,2$, to yield
$\mathbf{b}=\mathbf{0}-0.05([1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]\mathbf{\Delta}^{(1)})^T=[0.0109,\ -0.0167]^T$.

BP8: New iteration. With every new signal realization, e.g.
$\mathbf{x}=[-0.412 \quad 0.886 \quad -0.338 \quad -0.202 \quad -0.204 \quad 0.029 \quad 0.035 \quad -0.304]^T$, $\mathbf{t}=[1,\ 0]^T$,
go back to the first step with the new (updated) weights, $\mathbf{w}$ and $\mathbf{v}$, and bias weights $\mathbf{b}$.
Some of the results (the SoftMax output signal, P_1, and the FC layer weights, $v_p(m)$) over 1000 iterations are shown in Fig. LN-5.

$n=1,2,\ldots,N$ is the vertex index in the input layer
$k=1,2,\ldots,K$ is the channel (matched filter) index
$v=1,2,\ldots,M$ is the matched filter order index
$m=1,2,\ldots,KN$ is the FC input node index
$p=1,2,\ldots,S$ is the output node index