GRAPH-ADAPTIVE ACTIVATION FUNCTIONS FOR GRAPH NEURAL NETWORKS

GRAPH-ADAPTIVE ACTIVATION FUNCTIONS FOR GRAPH NEURAL NETWORKS

Thesis

to obtain the degree of Master in Computer Science with Specialization in Data Science and Technology at Delft University of Technology,
to be publicly defended on Thursday, July 23rd 2020 at 15:00

by

Alina Bianca IANCU

Born in Ploiesti, Romania.

Multimedia Computing Group,
Faculty of Electrical Engineering, Mathematics and Computer Science (Faculteit Elektrotechniek, Wiskunde en Informatica),
Delft University of Technology,
Delft, The Netherlands.

Thesis committee:

Chair:	Dr. Odette Scharenborg, Faculty EEMCS, TU Delft
Daily Supervisor:	Dr. Elvin Isufi, Faculty EEMCS, TU Delft
Committee Member:	Dr. Riccardo Taormina, Faculty CEG, TU Delft

Keywords: Activation functions; graph neural networks; graph signal processing; permutation equivariance.

Summary

Network data are essential in applications such as recommender systems, social networks, and sensor networks. A unique characteristic that these data encompass is the coupling between the data values and the underlying network structure on which these data are defined. Graph Neural Networks (GNNs) have been designed as tools to extend the benefits of deep learning approaches to network data. One crucial component of GNNs is the nonlinear component, also known as the activation function. The activation function allows capturing the nonlinear relationships present in the input data. However, in the current literature, the essential data-network topology coupling is ignored in the nonlinear component of the GNN. To address this limitation, we propose in this thesis a new family of activation functions for GNNs that account for the graph structure and capture the data-network topology coupling, while also allowing for a distributed implementation. Specifically, we propose an initial diffusion of the data over the graph, prior to the local nonlinearization operation. The nonlinearization is designed in a form akin to graph convolutions. The latter leads to a graph-adaptive trainable nonlinear component of the GNN that can be implemented directly or via kernel transformations, therefore, enriching the class of functions to represent the network data. Whether in the direct or kernel form, we show the permutation equivariance property is always preserved. This ensures the output of the GNN is independent of node labeling and that the GNN exploits graph symmetries to generalize to different graphs sharing similar symmetries. Numerical experiments with distributed source localization, finite-time consensus, and distributed regression demonstrate the applicability of the proposed graph-adaptive activation functions in distributed scenarios and show improved or comparable performance to pointwise as well as state-of-the-art localized nonlinearities. Our findings also suggest the benefits of the proposed activation functions in situations where the communication resources in the network are limited.

Acknowledgements

I would like to thank the persons who made this journey an incredible one and were there for me all along the way.

First, I would like to thank Elvin for his time, energy, and patience in supervising my thesis. It has been a wonderful learning experience for me, and that would not have been possible without your guidance and constructive feedback. Thank you for your support, for all the insightful discussions that we had throughout these nine months, and for all the things you taught me. Finally, I would like to thank you for all the constructive comments regarding my technical writing, this thesis is definitely at a better quality due to your feedback. I would also like to acknowledge Odette Scharenborg and Riccardo Taormina for accepting to be part of my thesis committee.

Second, I would like to thank my friends Gabriele, Colm, Matteo, Panos, Rafal, Jody, and Kyriakos. You made the last nine months a lot more fun, thank you for all the coffee breaks and the good times we had together. Colm and Matteo, thank you for all the virtual coffee breaks that we had during the pandemic as well, you really helped me stay motivated. I would like to give special thanks to Gabriele, the last nine months would have not been the same without you. Thank you for all the discussions and feedback and for always being there for me and supporting me, whether it was for technical or personal reasons. I would also like to thank Dusan for all the moral support and for always encouraging me over the past nine months. Thank you for being part of my life, for being there for me and for always making me believe in myself.

Finally, I would like to thank my family for all their support, love, and for always believing in me. I would not be where I am today without you.

CONTENTS

1 Introduction

Network data have become increasingly important and are essential in applications such as recommender systems, social, sensor, and biological networks [1, 2]. Differently from time series and images, which are represented in a regular domain, network data present an irregular structure, that is best encoded in graphs. For example, consider a network of weather stations. A node in the graph represents each station, and each edge incorporates the distance between two stations, as depicted in Figure 1.1. The data associated with this network is the temperature measurement recorded at each node (station). We call these data a graph signal. A unique characteristic that network data encompass is the coupling between the data and the underlying graph structure on which these data are defined. Concretely, the data and the graph topology are inherently coupled, as the network structure implies the values of the data defined on top of it. That is, considering the example in Figure 1.1, temperatures recorded at a certain node are related to the values recorded at neighboring nodes. This relation is encoded in the distances associated with each edge between two nodes: the lower the distance between the two stations, the higher the correlation between the temperatures recorded at those two stations. Network data can be processed in both centralized and distributed computation scenarios. However, a centralized computation might not always be feasible in practical settings, when the graph becomes too large or when the network does not have access to a centralized processing unit. Moreover, distributed computation facilitates the scalability of computation and endows the system with robustness to failures of the processing unit. The latter is fundamental in applications involving consensus, optimisation, and control [3–5]. Hence, it is essential to define tools that can process network data while capturing the essential data-network topology coupling and allowing for a distributed implementation.

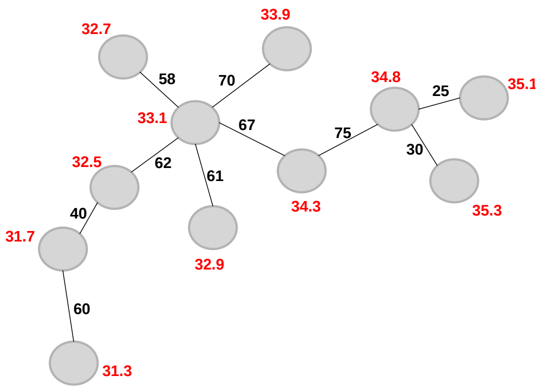


Figure 1.1: Example of graph encoding a network of weather stations. Each node represents a station. Each edge has associated, in black, the distance between two stations in kilometers. Each node has associated, in red, the temperature measured at the corresponding station in degrees Celsius.

In an effort to define tools that can process network data, a considerable amount of research was steered towards designing deep learning models that can exploit the irregularities in these data. More specifically, efforts were focused on graph neural networks (GNNs), and later on Graph Convolutional Neural Networks (GCNNs). These architectures were designed to reproduce the effectiveness that convolutional neural networks (CNNs) [6] proved to have in processing images and time series. Just as CNNs, GCNNs are composed of a linear and nonlinear component. However, differently from CNNs, GCNNs replace the traditional convolution performed using linear filters with the graph convolution defined by means of graph convolutional filters [7]. One property GCNNs inherit from graph filters is the distributed implementation [8–10].

The nonlinear component of GNNs is the essential element in learning nonlinear representations. Several approaches have been proposed in the GNNs literature for implementing this nonlinear component. First of all, traditional CNN activation functions, such as Rectified Linear Unit (ReLU), sigmoid or hyperbolic tangent (tanh), have been applied. Additionally, kernel activation functions (KAFs) have been recently introduced [11], which improve on the aforementioned ones since they can adapt to the data. However, these different methods are limited in the sense that they are applied locally at each node. In graphs, unlike in an image or time signal, the local neighborhoods vary, both within a graph and between different graphs, as depicted in Figure 1.2. This motivated the need to define an activation function that can adapt to the structure of the graph. To this end, [12] proposed localized activation functions that account for the graph topology by operating on node neighborhoods of different resolutions. However, the latter incorporate only the graph and not the data-topology coupling, since they ignore the edge weights and the data propagation between neighbors. Thus, when employing such activation functions, learning lacks meaningfulness, as it fails to capture an essential relation between the data and their underlying graph. Moreover, localized activation functions are not distributable beyond the one-hop neighborhood, hence missing multi-hop information between nodes in distributed scenarios.

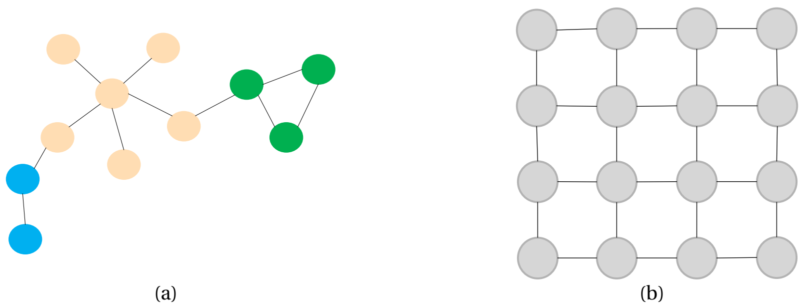


Figure 1.2: An illustration of a irregular and regular graph structure. (a) In a graph with irregular structure, local neighborhoods can have varying structures, as depicted by the different colors. (b) In an image, when represented as a graph (where each pixel is a node), the local neighborhoods have the same structure.

In the current literature, the data-graph coupling is only captured partially in the linear component of GNNs. However, the nonlinear component is essential for the discriminatory power of the network. Hence, to enhance the learning performance and significance, it is important to ensure that the data-graph coupling is fully captured by the GNN, not only in the linear component, but also in the nonlinear one. To address this, we put forward a new family of activation functions that adapt to the data-topology coupling in the surrounding of a node. The main research questions we focus on in this thesis are:

(RQ.1) *How can we embed the data-graph topology coupling in the nonlinear component of Graph Neural Networks?*

(RQ.2) *How can we develop activation functions that besides capturing the data-graph topology coupling are also distributable?*

To address these research questions, we propose activation functions in which the data associated with each node, obtained from the linear graph filtering, are diffused over the graph topology prior to the local nonlinear operation. The nonlinearization is performed in a form similar to graph convolutions, thus embedding the data-graph topology coupling. Since the diffusion process is performed before the nonlinear operation, the latter can capture information from arbitrarily far away neighbors by only operating locally at each node. Thus, our proposed activation functions lead to a distributed implementation. The resulting nonlinear features are subsequently combined with a set of trainable parameters to weigh the information at different neighborhood resolutions accordingly. This allows adapting the GNN's nonlinear component to the task at hand. Besides being graph-adaptive and distributable, these activation functions preserve a property of theoretical interest for GNNs, namely, permutation equivariance [13]. This implies the output of a GNN is invariant to node relabeling and, more importantly, that GNNs exploit graph symmetries to generalize learned representations to different graph signals that share some of these symmetries. Once we define the graph-adaptive, distributable activation functions, we address the following research question:

(RQ.3) *How can we employ our proposed graph-adaptive activation functions to address distributed processing tasks?*

To answer this research question, we analyze the performance of our proposed graph-adaptive activation functions in several scenarios, using both real-world and simulated data. First of all, we address the problem of distributed classification in the context of the source localization problem. Additionally, we focus on distributed finite-time consensus, a central problem in distributed and adaptive optimisation, signal processing, and control [4, 5, 14–18]. Specifically, we address distributed finite-time consensus as a learning problem on graphs. We employ a distributed GCNN to learn the consensus function in a data-driven fashion. Lastly, we experiment with distributed regression in the context of signal denoising. We employ a real-world dataset recording temperature measurements across an area of France.

Concretely, the answers to these research questions lead to the following contributions:

1. We develop a new family of nonlinearities for GNNs that are graph-adaptive to the surrounding of a node and distributable. The first class [Def. 5] nonlinearizes shifted features in the surrounding of a node in their direct form. The second class [Def. 8] transforms the shifted features with graph-adaptive kernels prior to nonlinearization.

2. We prove the proposed nonlinearities are permutation equivariant [Prop. 2], i.e., the output of the respective GNN architecture is agnostic to node labeling.

3. We propose distributed GNN tasks with graph-adaptive nonlinearities for source localization, finite-time consensus, and signal denoising.

The findings of this research lead to two papers, which we attach to this document, namely:

1. **Bianca Iancu**, Luana Ruiz, Alejandro Ribeiro, and Elvin Isufi, "Graph-Adaptive Activation Functions for Graph Neural Networks". *IEEE International Workshop on MACHINE LEARNING FOR SIGNAL PROCESSING (MLSP 2020)*, IEEE, September 21-24, 2020. [19];

2. **Bianca Iancu**, and Elvin Isufi, "Towards Finite-Time Consensus with Graph Convolutional Neural Networks". *28th European Signal Processing Conference (EUSIPCO 2020)*, IEEE, January 18-22, 2021. [20].

The content of these papers will be used throughout the upcoming chapters.

The remainder of this thesis proceeds as follows. Chapter 2 recalls the background material necessary for the research carried out in this thesis. Chapter 3 discusses the relevant literature for setting the context of our research. Chapter 4 defines the graph-adaptive nonlinearities and derives their theoretical properties. Chapter 5 contains the numerical experiments and Chapter 6 the conclusions.

Notation. Throughout this thesis the following notation is adopted. Scalars are denoted by plain letters (i.e., x, X), vectors by lowercase boldface letters (i.e., $\mathbf{x}$), matrices by uppercase boldface letters (i.e., $\mathbf{X}$), and sets by uppercase calligraphic letters (i.e., $\mathcal{X}$). x_i represents the ith entry of the vector $\mathbf{x}$, whereas X_{ij} represents the entry at the position (i,j) of the matrix $\mathbf{X}$. $\mathbf{X}^\top$ and $\mathbf{X}^{-1}$ denote the transpose and the inverse of the matrix $\mathbf{X}$, respectively.

2 BACKGROUND

Graph Signal Processing (GSP) extends the concepts and tools of traditional signal processing to data residing in irregular domains, best encoded in graphs [21]. In this chapter, we will introduce the background material necessary for the research carried out in this thesis. We will start with some basic concepts from graph signal processing in Section 2.1 and then introduce the graph shift operator in Section 2.2. We will then continue in Section 2.3 with graph filters and discuss their role in graph convolution. In Section 2.4, we will introduce the Graph Fourier Transform, which we will employ next to address the link between consensus and graph filtering in Section 2.5. In Section 2.6, we will introduce Graph Convolutional Neural Networks (GCNNs). Subsequently, we will present different activation functions employed in GCNNs in Section 2.7 and discuss the properties of GCNNs in Section 2.8. These include an overview of the learnable parameters in Section 2.8.1 and the permutation equivariance property in Section 2.8.2. Finally, we will conclude this chapter in Section 2.9.

2.1. Graphs and Graph Signals

$\mathcal{N}_i = \{j | (i,j) \in \mathcal{E}\}$

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ of cardinality $|\mathcal{V}| = N$ and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ of cardinality $|\mathcal{E}| = M$. An edge is a tuple $e_{ij} = (i,j)$ connecting nodes i and j. Depending on their edge types, graphs can be *undirected* or *directed*, as shown in Figure 2.1. In *undirected* graphs, edges do not present any orientation. That is, if node i is connected to node j (i.e., $(i,j) \in \mathcal{E}$), then also node j is connected to node i (i.e., $(j,i) \in \mathcal{E}$). In the case of *directed* graphs, edges present an orientation. Specifically, node i being connected to node j (i.e., $(i,j) \in \mathcal{E}$), does not imply that also node j is connected to node i (i.e., $(j,i) \in \mathcal{E}$). Moreover, we define the k-hop neighborhood of node i, N_i^k, as the set of nodes reachable from node i in at most k hopes. That is, the set of nodes that are reachable by following a path starting in node i and consisting of at most k edges. For simplicity of notation, throughout this thesis we denote the one-hop neighborhood of node i simply as N_i.

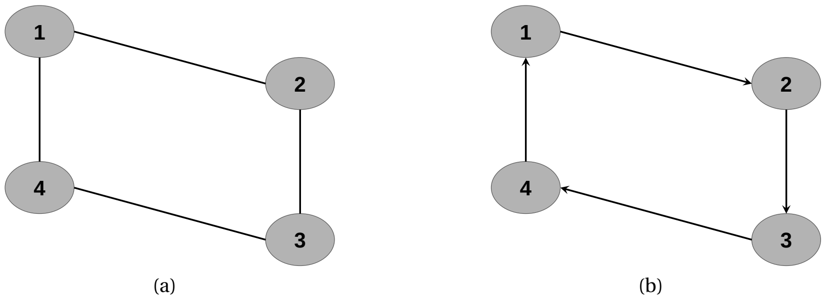


Figure 2.1: An illustration of a directed and an undirected graph. (a) In an undirected graph the edges have no orientation. (b) In a directed graph the edges present an orientation from one node to another.

On the vertices of $\mathcal{G}$, we define a graph signal $\mathbf{x} = [x_1, x_2, ..., x_N]^\top \in \mathbb{R}^N$, whose ith component x_i is the signal value of node i. An example of a graph signal is illustrated in Figure 2.2. If an edge (i,j) is present, it implies a relationship between the signal components at the nodes i and j. A typical example of a graph signal is a sensor network. The network topology encodes the pairwise relationships between the sensors, such as their distance or communication pattern, whereas the graph signal represents the actual measurements recorded at each sensor.

2.2. Graph Shift Operator

Associated to $\mathcal{G}$ we have the *graph shift operator (GSO)* matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$, whose sparsity pattern matches the graph structure. That is, entry (i,j) satisfies $[\mathbf{S}]_{ij} = s_{ij} \neq 0$ only if $i = j$ or $(i,j) \in \mathcal{E}$. Commonly used GSOs include the adjacency matrix $\mathbf{A}$, the graph Laplacian $\mathbf{L}$ (for undirected graphs), and their normalized and translated forms, which we detail next.

The graph adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ is a sparse matrix with entries

$$A_{ij} \neq 0 \text{ if } (i,j) \in \mathcal{E}, \text{ and } A_{ij} = 0 \text{ if } (i,j) \notin \mathcal{E}.$$

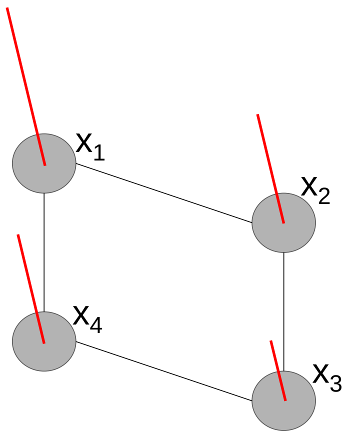


Figure 2.2: Graph signal $\mathbf{x} = [3,2,1,2]^\top$ illustrated on top of the underlying graph, composed of four nodes. Each node has a signal value associated. The signal values are represented with the red bars; the bar height indicates the signal value.

That is, A_{ij} has non-zero values only for the entries associated to edges in the graph. The entry A_{ij} captures the weight associated to the edge (i,j), which can be interpreted as the strength of the connection between nodes i and j. Depending on the values of the edge weights, we can identify two types of graphs: *weighted*, with weights $A_{ij} \in \mathbb{R}$, and *unweighted*, in which case the weight of each edge is considered to be one. In the particular case of an undirected graph, the adjacency matrix $\mathbf{A}$ is symmetric, i.e. $\mathbf{A} = \mathbf{A}^\top$, i.e., $A_{ij} = A_{ji}$.

The degree matrix $\mathbf{D} \in \mathbb{R}^{N\times N}$ for undirected graphs is defined as the diagonal matrix $\mathbf{D}$ with entries $D_{ii} = \sum_{j=1}^{N} A_{ij}$ and $D_{ij} = 0$ for $i \neq j$. That is, the degree of a node i is represented by the number of edges incident to i if the graph is unweighted, while it is the sum of the weights corresponding to the edges incident to i if the graph is weighted. By means of the degree matrix $\mathbf{D}$, we can define the normalized adjacency matrix of $\mathcal{G}$ as $\mathbf{A}_\mathrm{n} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$.

Moreover, the graph Laplacian matrix $\mathbf{L}$ is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Thus, we can identify two types of entries in $\mathbf{L}$:

1. diagonal entries: $L_{ii} = D_{ii}$, i.e. the degree of each node i;
2. off-diagonal entries: $L_{ij} = -A_{ij}$ where $i \neq j$.

The normalized Laplacian matrix $\mathbf{L}_\mathrm{n}$ is defined as $\mathbf{L}_\mathrm{n} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$.

The choice for the GSO varies depending on the application, and different choices have different trade-offs. The adjacency matrix $\mathbf{A}$ can be defined for both undirected and directed graphs, whereas the Laplacian matrix $\mathbf{L}$ only applies to undirected graphs. Moreover, $\mathbf{L}$ is symmetric and positive semidefinite, which reduces the analytical and numerical difficulties that arise when working with the adjacency matrix [21]. However, one should consider the different options and choose the matrix representation that suits the best in the application of interest.

The main operation carried out by the GSO is the shifting of a graph signal $\mathbf{x}$ over graph $\mathcal{G}$, that is

$$\mathbf{x}^{(1)} = \mathbf{S}\mathbf{x} \tag{2.1}$$

where $\mathbf{x}^{(1)}$ represents the graph signal shifted once by $\mathbf{S}$. We can also define the initial graph signal $\mathbf{x}$ as its zero-shifting by $\mathbf{S}$, or more precisely as $\mathbf{x}^{(0)} = \mathbf{S}^0\mathbf{x}$. More generally, the kth shift of the graph signal $\mathbf{x}$ can be computed recursively as

$$\mathbf{x}^{(k)} = \mathbf{S}^k\mathbf{x} = \mathbf{S}(\mathbf{S}^{k-1}\mathbf{x}) = \mathbf{S}\mathbf{x}^{(k-1)}. \tag{2.2}$$

By breaking down the computation of one entry of the kth shifted graph signal $\mathbf{x}^{(k)}$, we can highlight the locality property of the GSO $\mathbf{S}$. For each node i, the kth shifted signal is computed locally as a linear combination of the values of the $(k-1)$th shifted signal at i and at its neighboring nodes $j \in \mathcal{N}_i$, as

$$x_i^{(k)} = S_{1i}x_1^{(k-1)} + S_{2i}x_2^{(k-1)} + \ldots + S_{Ni}x_N^{(k-1)}. \tag{2.3}$$

An essential property of the GSO is its readily distributed implementation. By exploiting recursion (2.2), we observe that node i can compute the shifted signal $\mathbf{x}^{(k)}$ by exchanging previous shift information $\mathbf{x}^{(k-1)}$ with its direct one-hop neighbors $\mathcal{N}_i$, since the shift operator is local [cf. (2.3)]. Shifting the the graph signal $\mathbf{x}$ once, as in (2.1), amounts to a computational complexity of $\mathcal{O}(M)$, since the GSO is a sparse matrix with non-zero elements only in the positions associated to an edge. Thus, the recursive implementation introduced in (2.2) allows for a distributed communication and computational cost of order $\mathcal{O}(Mk)$ [10], as it requires applying k times the operation in (2.1).

2.3. GRAPH CONVOLUTIONS

A graph convolution is defined as a graph filter $\mathbf{H}(\mathbf{S})$ that can be written as a polynomial of the GSO $\mathbf{S}$ as

$$\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{K} h_k\mathbf{S}^k \tag{2.4}$$

where $\mathbf{h} = [h_0, \ldots, h_K]^\top$ is a vector of coefficients [7]. Since the GSO $\mathbf{S}$ is local, graph filters benefit from a local implementation as well. For an input signal $\mathbf{x}$ and filter coefficients $\mathbf{h}$, the output $\mathbf{y} \in \mathbb{R}^N$ of the graph convolutional filter is computed as

$$\mathbf{y} = \mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{K} h_k\mathbf{S}^k\mathbf{x}. \tag{2.5}$$

In other words, the output graph signal $\mathbf{y}$ of the graph convolution applied to the input graph signal $\mathbf{x}$ is a linear combination of each kth shifted signal $\mathbf{S}^k\mathbf{x}$, where $0 \le k \le K$. Figure 2.3 illustrates this process.

As in traditional convolutions, the localization property is preserved in operation (2.5). Each term $h_k\mathbf{S}^k\mathbf{x}$ is associated with a k localized neighborhood within the graph. More specifically, the first polynomial $\mathbf{S}^0\mathbf{x}$ represents an operation at the level of the individual nodes; the second polynomial $\mathbf{S}^1\mathbf{x}$ represents an operation within the one-hop neighborhood of each node; the last polynomial $\mathbf{S}^K\mathbf{x}$ represents an operation within the

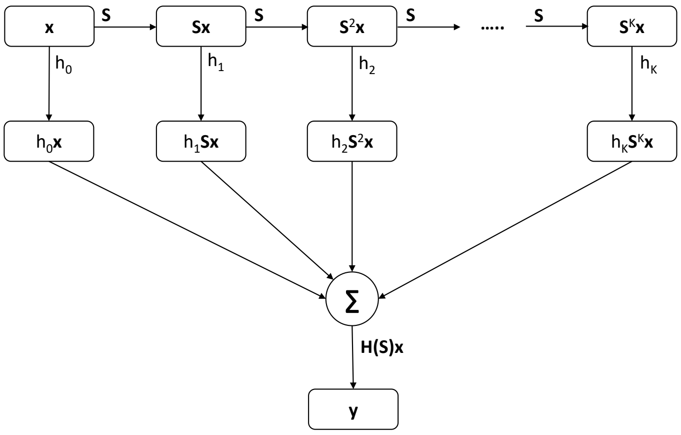


Figure 2.3: The convolution operation of a graph signal. The graph shift operator **S** is applied K times to the input signal **x**. Each kth shift is multiplied by its coefficient h_k. All the terms are summed up to obtain the output of the graph convolution.

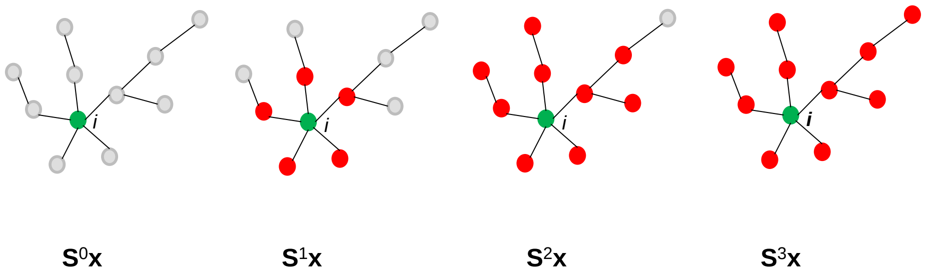


Figure 2.4: Illustration of the localized neighborhoods associated to each $\mathbf{S}^k\mathbf{x}$ term in the graph convolution operation (2.5). The neighborhoods (in red) of node i (in green) are up to four-hops. Term $\mathbf{S}^0\mathbf{x}$ represents an operation at the level of i; $\mathbf{S}^1\mathbf{x}$ represents an operation at the level of the one-hop neighborhood of i; $\mathbf{S}^2\mathbf{x}$ represents an operation at the level of the two-hop neighborhood of i; $\mathbf{S}^3\mathbf{x}$ represents an operation at the level of the three-hop neighborhood of i.

K-hop neighborhood of each node. This locality property is illustrated in Figure 2.4 for a three-hop neighborhood.

Due to their locality property, graph convolutions can be run distributively. When building the output **y** in (2.5), we need to compute the terms $\mathbf{S}\mathbf{x}, \ldots, \mathbf{S}^K\mathbf{x}$. As discussed in Section 2.1, **S** is local and operation **Sx** requires one-hop node exchanges between each node i and its one-hop neighbors. This implementation allows for distributed communications and computational cost of order $\mathcal{O}(MK)$, while the parameters defining (2.5) are of order $\mathcal{O}(K)$ [7].

2.4. Graph Fourier Transform

By exploiting the graph spectral domain, we can analyze the filtering behavior introduced in (2.4) from a graph frequency perspective [22]. Specifically, consider the eigendecomposition $\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{-1}$ with eigenvectors $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_N]$ and eigenvalues $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_N)$. The graph Fourier transform (GFT) of $\mathbf{x}$ is defined as $\hat{\mathbf{x}} = \mathbf{U}^{-1}\mathbf{x}$, where $\hat{x}_i$ quantifies how much eigenvector $\mathbf{u}_i$ contributes to the variation of signal $\mathbf{x}$ over the graph [22]. The inverse GFT is $\mathbf{x} = \mathbf{U}\hat{\mathbf{x}}$ and the eigenvalues $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_N)$ are referred to as the graph frequencies.

By exploiting the GFT, we can write the input-output graph filtering relation in (2.4) as

$$\hat{\mathbf{y}} = \mathbf{H}(\boldsymbol{\Lambda})\hat{\mathbf{x}} \tag{2.6}$$

where the diagonal matrix $\mathbf{H}(\boldsymbol{\Lambda}) = \sum_{k=0}^{K} h_k \boldsymbol{\Lambda}^k$ contains the filter frequency response on the main diagonal. That is, graph convolutions are performed in the graph frequency domain as pointwise multiplications between the filter frequency response and the signal GFT. This is equivalent to the operation of convolution in time and space, where the filter temporal or spatial frequency response is pointwise multiplied with the signal discrete Fourier transform. By applying the inverse GFT on both sides of (2.6) we can retrieve the vertex domain filter output as

$$\mathbf{y} = \mathbf{H}(\mathbf{S})\mathbf{x}$$

where $\mathbf{H}(\mathbf{S}) = \mathbf{U}\mathbf{H}(\boldsymbol{\Lambda})\mathbf{U}^{-1}$.

2.5. CONSENSUS AS GRAPH SIGNAL FILTERING

One of the numerical application considered in this thesis is finite-time consensus. It is a fundamental problem in graph signal processing and it aims to achieve consensus among all nodes in a graph in finite-time, by accessing only local information at each node. The GFT introduced in Section 2.4 plays a role in approaching consensus from a spectral perspective, as we will detail next.

Consider the signal $\mathbf{x}$ and the consensus version $\bar{\mathbf{x}} = \bar{x}\mathbf{1}$, with $\bar{x}$ being the mean of $\mathbf{x}$ and $\mathbf{1}$ the vector of all ones. We can think of $\bar{\mathbf{x}}$ as a signal whose GFT coefficients $\hat{\bar{\mathbf{x}}}$ are such that the combined eigenvectors yield the DC component. For the GSO being the graph Laplacian $\mathbf{S} = \mathbf{L}$, this is straightforward since eigenvector $\mathbf{u}_1$ associated to the smallest eigenvalue $\lambda_1 = 0$ is constant, i.e., $\mathbf{u}_1 = 1/\sqrt{N}\mathbf{1}$. Only the first coefficient $\hat{\bar{x}}_1$ is necessary to represent the consensus signal, while all other coefficients can be null, $\hat{\bar{x}}_2 = \dots = \hat{\bar{x}}_N = 0$. For $\mathbf{S}$ being the adjacency matrix or any other graph representation matrix that does not have a constant eigenvector, vector $\hat{\bar{\mathbf{x}}}$ will have more than one entry (if not all) non-zero to represent the constant signal.

We can think of consensus as a graph filter, as introduced in (2.4) that takes an heterogeneous graph signal $\mathbf{x}$ and filters it to return the constant mean signal $\bar{\mathbf{x}} = \bar{x}\mathbf{1}$ over the nodes [3]. Exploiting the GFT, we can write the input-output graph filtering relation as $\hat{\mathbf{y}} = \mathbf{H}(\boldsymbol{\Lambda})\hat{\mathbf{x}}$, where the diagonal matrix $\mathbf{H}(\boldsymbol{\Lambda}) = \sum_{k=0}^{K} h_k \boldsymbol{\Lambda}^k$ contains the filter frequency response on the main diagonal. Reaching consensus with graph filters of the form in (2.4) accounts for learning the filter parameters $\mathbf{h}$ such that the signal is low-pass filtered to pass only the DC component. Another advantage of (2.4) is its readily distributed implementation, as discussed in Section 2.3.

The main benefit of (2.4) is that, under appropriate conditions on the spectrum of $\mathbf{S}$ [23], coefficients $\mathbf{h}$ can be designed to achieve exact finite-time consensus in at most $K = N$ iterations [3, 17]. However, their applicability is limited to simple (cyclic or star) graphs, since these approaches require high numerical precision of the eigenvalues. An approach to tackle the numerical issues is to consider a different graph filter in (2.4), such as ARMA [9], node varying [10], or edge varying [22]. Of particular interest is the so-called edge varying graph filter [22], which substitutes scalars h_k with $N \times N$ coefficient matrices $\mathbf{H}_k$ in which entry (i,j) is the coefficient applied to edge e_{ij}. In this case, finite-time consensus can be approximated with higher accuracy compared with (2.4), but the graph structure and its labeling should be fixed. The latter is also practically non-transferable to a slightly different graph, such as a graph affected by link losses.

In this thesis, we employ a GCNN instead with filters of the form in (2.4). This does not require the GSO eigendecomposition, fixed labeling, and it is better transferable to unseen graphs than the linear graph filter [13]. We introduce GCNNs next.

2.6. GRAPH CONVOLUTIONAL NEURAL NETWORKS

Consider a training set $\mathcal{T} = \{(\mathbf{x},\mathbf{y})\}$ consisting of $|\mathcal{T}|$ tuples $(\mathbf{x},\mathbf{y})$, representing input-output pairs. Each input graph signal $\mathbf{x}$ is associated with an output $\mathbf{y}$, which can be represented by a class label in the case of a classification problem or another graph signal in the case of regression. The goal of Graph Convolutional Neural Networks (GCNNs) is to learn a representation based on these data to predict the output $\tilde{\mathbf{y}}$ for a new, unseen graph signal, outside of the training set $\mathcal{T}$ (i.e. $\mathbf{x} \notin \mathcal{T}$).

Analogously to CNNs, GCNNs are composed of L convolutional layers. Each convolutional layer consists of two fundamental elements: a linear and a nonlinear component. The linear component comprises a collection of graph filters to perform graph convolutions [cf. 2.5]. This is followed by an activation function, which we will address in Section 2.7.

At layer l, the GCNN takes as input F_{l-1} features $\{\mathbf{x}_{l-1}^g\}_{g=1}^{F_{l-1}}$ from layer $(l-1)$ and produces F_l output features $\{\mathbf{x}_l^f\}_{f=1}^{F}{}_l$. Each input feature $\mathbf{x}_{l-1}^g$ is processed by a parallel bank of F_l graph filters $\{\mathbf{H}_l^{fg}(\mathbf{S})\}_f$. The filter outputs are aggregated over the input index g to yield the fth convolved feature

$$\mathbf{z}_l^f = \sum_{g=1}^{F_{l-1}} \mathbf{H}_l^{fg}(\mathbf{S})\mathbf{x}_{l-1}^g = \sum_{g=1}^{F_{l-1}} \sum_{k=0}^{K} h_{kl}^{fg} \mathbf{S}^k \mathbf{x}_{l-1}^g, \text{ for } f = 1,\ldots,F_l. \tag{2.7}$$

The convolved feature $\mathbf{z}_l^f$ is a graph signal and it is subsequently passed through an activation function $\sigma(\cdot)$ to obtain the fth convolutional layer output

$$\mathbf{x}_l^f = \sigma(\mathbf{z}_l^f), \quad \text{for } f = 1,\ldots,F. \tag{2.8}$$

The resulting output of the lth layer, $\mathbf{x}_l^1,\ldots,\mathbf{x}_l^{F_l}$, serves as the input to the next convolutional layer. The steps in (2.7) and (2.8) are followed recursively until the last layer L of the network. The output features of the last convolutional layer L, $\mathbf{x}_L^1,\ldots,\mathbf{x}_L^{F_L}$, represent the final convolutional features.

Each layer l of the GCNN is characterized by $F_{l-1} \times F_l$ coefficient vectors $\mathbf{h}_l^{fg} = [h_{0l}^{fg},\ldots,h_{Kl}^{fg}]^\top$ of dimensions $K+1$ defining the filters $\mathbf{H}_l^{fg}(\mathbf{S})$ in (2.7). The total number of learnable parameters for layer l is $F_{l-1}F_l(K+1)$, which divides as: for each F_{l-1} input feature $\mathbf{x}_{l-1}^g$

from layer $(l-1)$, there are F_l filters $\mathbf{H}^{fg}(\mathbf{S})$ at layer l; each of these filters consists of $(K+1)$ learnable coefficients h_l^{fg}. The total computational cost of a convolutional layer is $\mathcal{O}(M(K+1)F_{l-1}F_l)$. This is because we perform $F_{l-1}F_l$ convolution operations, each of cost $\mathcal{O}(M(K+1))$.

The output features of the last convolutional layer $\mathbf{x}_L^1,\ldots,\mathbf{x}_L^{F_L}$ can be interpreted as a collection of F_L graph signals, where on node i we have the $F_L \times 1$ feature vector $\boldsymbol{\chi}_{Li} = [x_{Li}^1,\ldots,x_{Li}^{F_L}]^\top$. These features can be further combined by means of a one-layer perceptron[1] to obtain the final GCNN output. Based on how the perceptron is implemented, we have two types of GCNNs:

1. **Centralized GCNN**. Such an architecture is depicted in Figure 2.5. The one layer perceptron is defined for the whole graph features, such that it combines the features concatenated from all the nodes into the $F_L \times N$ feature vector $\boldsymbol{\chi}_L = [\boldsymbol{\chi}_{L1}^\top,\ldots,\boldsymbol{\chi}_{LN}^\top]^\top$ to obtain the GCNN output as
$$\tilde{\mathbf{y}} = \mathbf{H}_{\text{FC}}\boldsymbol{\chi}_L. \tag{2.9}$$
Here, matrix $\mathbf{H}_{\text{FC}} \in \mathbb{R}^{F_o \times NF_L}$ maps the NF_L convolutional features $\boldsymbol{\chi}_L$ into the F_o output features (e.g., the number of classes).

2. **Distributed GCNN**. Such an architecture is depicted in Figure 2.6. The one-layer perceptron is defined per node, such that each node i locally combines the features $\boldsymbol{\chi}_{Li}$ to obtain the GCNN output as
$$\tilde{y}_i = \mathbf{h}_{\text{FC}}^\top \boldsymbol{\chi}_{Li}. \tag{2.10}$$
Here, $\mathbf{h}_{\text{FC}} = [h_1,\ldots,h_{F_L}]^\top$ is the $F_L \times 1$ vector of parameters in the local fully connected layer. Vector $\mathbf{h}_{\text{FC}}$ is shared among nodes to keep the number of trainable parameters independent of the graph dimensions (i.e. N and M), but only dependent on the filter order and the number of features and layers. Recall this architecture is distributable since the graph convolutional filters are distributable [cf. Section 2.3] and if the activation functions are chosen accordingly.

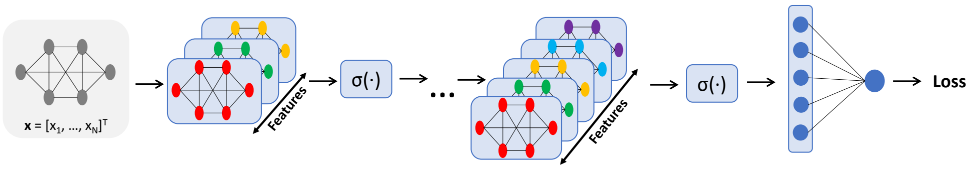


Figure 2.5: Centralized GCNN architecture. The input is a graph signal **x**, which is filtered by a filter bank of F graph filters [cf. (2.5)] and then passed through an activation function $\sigma(\cdot)$ (e.g., ReLU). This forms a graph convolutional layer, which is cascaded L times. The final convolutional features are concatenated for all nodes and passed to a fully connected layer to compute the final output. This output is used during training the minimze the loss.

In this thesis, we will focus on the distributed GCNN. However, our findings apply directly to the centralized GCNN, since only the final fully connected layer differs.

[1]A multi-layer perceptron can be employed as well.

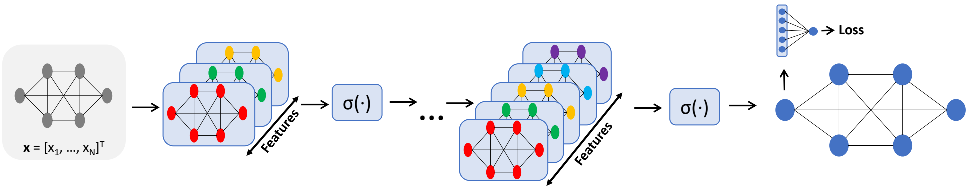


Figure 2.6: Distributed GCNN architecture. The input is a graph signal **x**, which is filtered by a filter bank of F graph filters [cf. (2.5)] and then passed through an activation function $\sigma(\cdot)$ (e.g., ReLU). This forms a graph convolutional layer, which is cascaded L times. The final convolutional features are concatenated per node and passed to a *per-node* fully connected layer to compute the final output. This output is used during training the minimze the loss.

By grouping all learnable parameters in set $\mathscr{H} = \{\mathbf{h}_l^{fg}; \mathbf{h}_{\mathrm{FC}}\}_{lfg}$, we can consider the GCNN as a map $\mathbf{\Phi}(\cdot)$ that takes as input a graph signal **x**, a GSO **S**, and a set of parameters $\mathscr{H}$ to produce the output

$$\mathbf{\Phi}(\mathbf{x}; \mathbf{S}; \mathscr{H}) := \tilde{\mathbf{y}}. \tag{2.11}$$

The output in (2.11) is computed for the training set $\mathscr{T}$. During training, the goal is to learn representations **y** of input data **x** by minimizing the loss over the training set $\mathscr{T}$

$$\mathscr{L} = \sum_{r=1}^{|\mathscr{T}|} J(y_r, \tilde{y}(x_r)) \tag{2.12}$$

where $J(\cdot)$ is the cost function. The generalization ability of the GCNN is tested by applying the model to unseen graph signals $\mathbf{x} \notin \mathscr{T}$. The choice for the loss function depends on the problem. For regression, we consider the mean squared error (MSE), whereas for classification we employ the cross-entropy (CE). The Mean Square Error (MSE) is defined as

$$\mathscr{L} = \frac{1}{|\mathscr{T}|} \sum_{m=1}^{|\mathscr{T}|} (\mathbf{y}_m - \tilde{\mathbf{y}}(x_m))^2. \tag{2.13}$$

The cross-entropy (CE) for a data sample when considering a problem of C classes is defined as

$$\mathscr{L}_r = -\sum_{c=1}^{C} y_{r,c} log(p_{r,c}) \tag{2.14}$$

where $y_{r,c}$ represents the groundtruth binary indicator of whether the data point $\mathbf{x}_r$ belongs to class c or not, whereas $p_{r,c}$ is the predicted probability by the GCNN of the data point $\mathbf{x}_r$ belonging to class c. The cross entropy loss for the entire training set is the average of the sample cross entropy

$$\mathscr{L} = \frac{1}{|\mathscr{T}|} \sum_{r=0}^{|\mathscr{T}|} \mathscr{L}_r. \tag{2.15}$$

The loss is minimized w.r.t. parameters $\mathscr{H}$ using standard backpropagation with stochastic gradient descent or any other preferred descent method [24]. This is because all operations are differentiable [25].

2.7. ACTIVATION FUNCTIONS

There are two main directions when it comes to activation functions employed with GCNNs: pointwise and localized approaches. In this section, we recall the most popular nonlinearities used with GCNNs in the literature and discuss their limitations, which also motivate the research in this thesis. These include the conventional pointwise activation functions, the pointwise kernel activation functions, and the localized activation functions [12].

2.7.1. CONVENTIONAL POINTWISE ACTIVATION FUNCTIONS

Pointwise nonlinearities employed in GCNNs are those encountered also in traditional CNNs, such as the rectified linear unit (ReLU), the Leaky ReLU, the sigmoid, or hyperbolic tangent (tanh), which we recall here.

- **ReLU.** The ReLU activation function, illustrated in Figure 2.7a, is defined as

$$\mathrm{ReLU}(x) = \max(0, x). \tag{2.16}$$

- **Leaky ReLU.** Compared to ReLU, Leaky ReLU allows for a small, positive gradient α when $x < 0$. It is shown in Figure 2.7b with slope 0.01. The Leaky ReLU is defined as

$$\mathrm{Leaky\_ReLU}(x) = \begin{cases} x, & \text{if } x > 0. \\ \alpha x, & \text{otherwise, where } \alpha \text{ indicated the slope.} \end{cases} \tag{2.17}$$

- **Sigmoid.** The sigmoid activation function, whose plot is shown in Figure 2.7c, is defined as

$$\mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}. \tag{2.18}$$

- **Tanh.** The tanh activation function, shown in Figure 2.7d, is defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.19}$$

2.7.2. KERNEL ACTIVATION FUNCTIONS

One recently proposed pointwise activation function is the Kernel Activation Function (KAF) [11]. The KAF is modeled in terms of a kernel expansion over D terms as

$$g(x) = \sum_{i=1}^{D} \alpha_i k(x, d_i) \tag{2.20}$$

where $\alpha_1, \ldots, \alpha_D$ are the mixing coefficients and represent the linear coefficients of the kernel expansion; $d_1, \ldots, d_D$ represent the dictionary elements (i.e. the elements used for computing the kernel values); $k(\cdot,\cdot) : \mathbb{R} \to \mathbb{R}$ is an one-dimensional kernel.

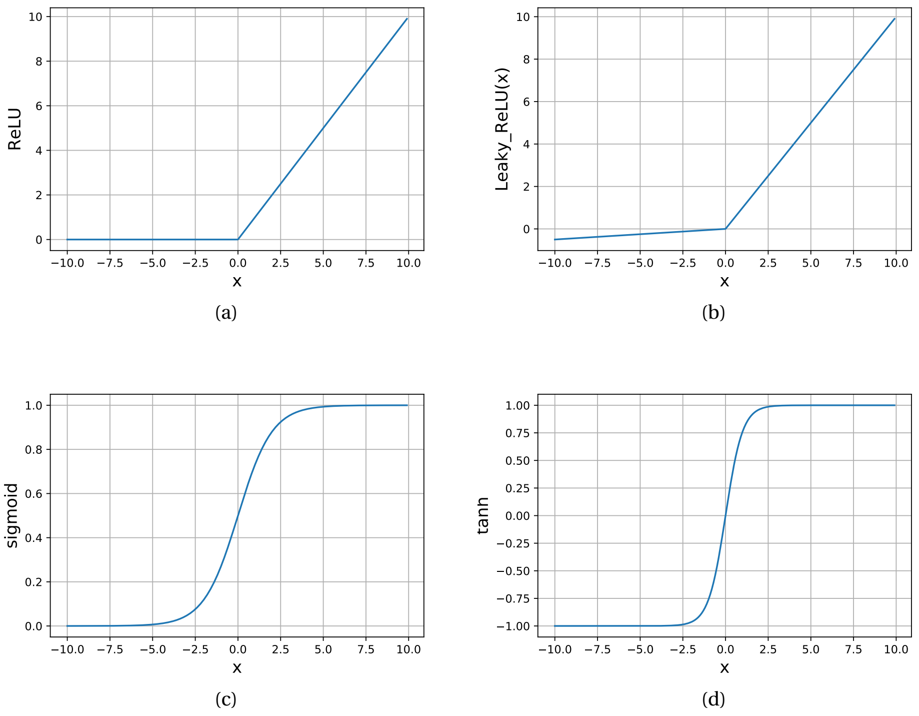


Figure 2.7: Pointwise Activation Functions defined over the interval $[-10,10]$. (a) ReLU Activation Function. (b) Leaky ReLU Activation Function. (c) Sigmoid Activation Function. (d) Tanh Activation Function.

The dictionary elements are fixed and only the mixing coefficients are adapted. More specifically, the dictionary elements $d_1,\dots,d_D$ are obtained by sampling D values uniformly around zero, where D is a hyperparameter. Thus, D controls the method flexibility: the larger D, the higher the model flexibility at the cost of a larger number of parameters. Kernel $k(\cdot,\cdot)$ can be any generic kernel function that satisfies the semi-definiteness property [26]. That is, for any choice of parameters $\alpha_1,\dots,\alpha_D$ and dictionary elements $d_1,\dots,d_D$, the following inequality needs to be satisfied

$$\sum_{i=1}^{D}\sum_{j=1}^{D}\alpha_i\alpha_j k(d_i,d_j)\geq 0. \tag{2.21}$$

The main contribution that KAFs bring to GCNNs is the increased flexibility, as they allow to model a larger class of shapes by adapting the activation functions themselves from the training data. By properly choosing the elements of the kernel expansion, each function can be represented with a small set of linear mixing coefficients, which are trained together with the weights of the graph convolutional layers through standard backpropagation.

2.7.3. LOCALIZED ACTIVATION FUNCTIONS

Localized activation functions are the first approach focusing on embedding the graph topology in the nonlinear component of the GCNN [12]. These are implemented by means of localized graph filters, as defined below.

To start, let us define the localized operator, which represents the building block of the localized graph filters [12].

Definition 1 *(Localized Operator). Let the localized operator $f(\mathbf{S},\mathbf{x})$ be a non-parametric function that performs a nonlinear aggregation within the neighborhood of each node i, induced by the GSO $\mathbf{S}$ in the graph $\mathcal{G}$. When applied to a signal $\mathbf{x}$, the output of a localized operator is a signal $\mathbf{z}$ with ith entry*

$$\mathbf{z}_i = [f(\mathbf{S},\mathbf{x})]_i = f(x_j : j \in \mathcal{N}_i). \tag{2.22}$$

A localized operator accounts for the neighborhood topology within a graph. For each node i the information captured by the graph signal x_j within its neighborhood $\mathcal{N}_i$ is aggregated by means of any nonlinear aggregation function $f(\cdot)$. Two examples of $f(\cdot)$ are the median and max functions.

Based on the localized operator, we can define the localized graph filter as follows.

Definition 2 *(Localized Graph Filter). Let $\mathbf{S}$ be a graph shift operator and consider the parameter vector $\mathbf{h} = [h_1,\ldots,h_K]^\top$ consisting of K elements. The output of the localized graph filter applied to signal $\mathbf{x}$ is the signal*

$$\mathbf{z} = \sum_{k=1}^{K} h_k f(\mathbf{S}^k,\mathbf{x}). \tag{2.23}$$

We can contrast Definition 2 with the graph convolution in (2.5). Contrary to the graph convolution (2.5), the terms in (2.23) are aggregated nonlinearly by $f(\cdot)$. This function summarizes at each node the signal within the k-hop neighborhood induced by $\mathbf{S}^k$.

Considering a localized graph filter according to (2.23), a scalar β, and a collection of filter coefficients $[h_{l1}^f,\ldots,h_{lK}^f]$ for a layer l of a GCNN, we can replace the activation function in (2.8) with a localized activation function to obtain the output

$$\mathbf{x}_l^f = \beta\mathrm{ReLU}(\mathbf{z}_l^f) + \sum_{k=1}^{K} h_{lk} f(\mathbf{S}^k,\mathbf{z}_l^f) \tag{2.24}$$

where K is the maximum neighborhood resolution considered and $\mathbf{z}_l^f$ are the convolved features from the lth layer in the network. Localized activation functions can be trained using backpropagation [12]. An example of how these activation functions operate is shown in Figure 2.8 for a maximum neighborhood resolution of $K = 3$.

A limitation of the localized activation functions is that they cannot be implemented in applications requiring distributed communication. At each node i, information from the nodes of up to K-hops away is accessed, which is beyond the local accessibility of i. Hence, in distributed settings, the order K in (2.24) is limited to one. Moreover, localized activation functions do not account for the edge weights when aggregating the graph

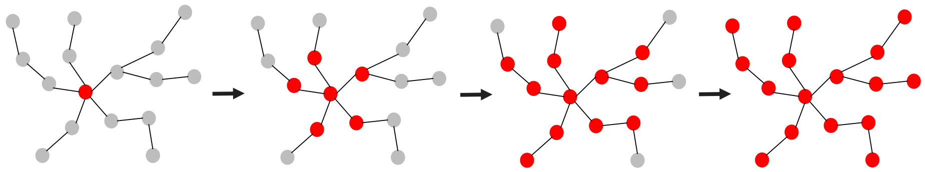


Figure 2.8: Localized activation functions [12] applied for a maximum neighborhood resolution of $K = 3$. The nonlinearity is applied for the node initially highlighted in red. At each step, we show in red the neighborhood resolution for which the signal is aggregated. Initially, the node's signal is considered individually, followed by sequentially accounting for the multiresolution neighborhoods. This is achieved by applying the nonlinear aggregation in the one-hop, two-hops and, finally, three-hops neighborhoods.

signal within a neighborhood. When building x_l^f, they directly access the signal information at nodes up to K-hops away, without accounting at all for the signal propagation along the edges, and, thus, for the contribution of the edge weights in the signals aggregation. To address these limitations, we propose two new activation functions based on local operators and kernel functions that not only account for the graph structure, but also incorporate edge weights and can be run distributively.

2.8. Graph Convolutional Neural Networks Properties

2.8.1. Learnable Parameters

Depending on the activation function employed, the total number of trainable parameters defining the GCNN varies.

Conventional Pointwise Activation Functions. If conventional pointwise activation functions are used, the total number of parameters of the GCNN is $\sum_{l=2}^{l=L} F_{l-1}F_l(K+1) + F_1(K+1) + F_L$. This divides as: $i)$ $F_1(K+1)$ parameters for the F_1 filters in the first graph convolutional layer; $ii)$ $\sum_{l=2}^{l=L} F_{l-1}F_l(K+1)$ for the parameters of the $F_{l-1}F_l$ filters in each of the remaining $L-1$ graph convolutional layers; and $iii)$ F_L parameters in the final fully-connected layer.

Kernel Activation Functions. If the kernel activation function is employed, DL extra parameters are added compared to the conventional pointwise activation functions. Thus, the total number of parameters of the GCNN is $\sum_{l=2}^{l=L} F_{l-1}F_l(K+1) + F_1(K+1) + F_L + DL$.

Localized Activation Functions. If localized activation functions are employed, $(K+1)L$ extra parameters are added compared to the conventional pointwise activation functions. Thus, the total number of parameters of the GCNN is $\sum_{l=2}^{l=L} F_{l-1}F_l(K+1) + F_1(K+1) + F_L + (K+1)L$.

2.8.2. Permutation Equivariance

The coupling *graph filter-activation function* embodies the GCNN with the permutation equivariance property. In simple words, permutation equivariance implies that processing a graph signal with the GCNN is independent of the labeling [12].

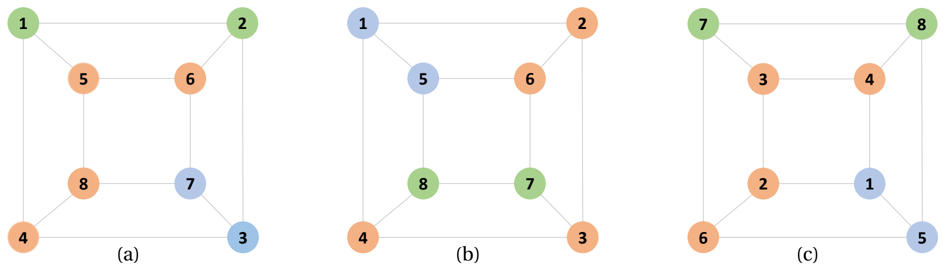


Figure 2.9: Permutation equivariance property of GCNNs inspired by Proposition 1. Illustrated is a graph with eight nodes and a signal defined on top of it, where different colors represent different signal values. Nodes are labeled as integers from 1 to 8. The graph signals in (a) and (b) are different signals on the same graph, but they are permutations of each other - interchange inner and outer squares in (b) and rotate the graph 180° [cf. (c)]; we observe the same signal in (c) as in (a). The permutation equivariance property, stated in Proposition 1, implies that a GCNN would be capable of classifying the signal in (b) by only seeing examples as in (a).

To start, let us introduce the permutation matrix $\mathbf{P}$. It is a square binary matrix with exactly one non-zero entry in each row and column. When pre-multiplied with another matrix it results in permuting the rows of that matrix, whereas post-multiplication permutes the columns. The permutation matrix $\mathbf{P}$ is orthonormal, that is $\mathbf{P}\mathbf{P}^\top = \mathbf{P}^\top\mathbf{P} = \mathbf{I}$. This implies the inverse matrix exists and satisfies the equality $\mathbf{P}^{-1} = \mathbf{P}^\top$. Employing the permutation matrix, we can now introduce next the permutation equivariance property.

Proposition 1 *(Permutation Equivariance) Consider a graph signal* $\mathbf{x}$ *defined over the graph* $\mathscr{G}$ *with associated graph shift operator* $\mathbf{S}$*. Consider also the GCNN mapping* $\mathbf{\Phi}(\mathbf{x};\mathbf{S};\mathscr{H})$ *with coefficient set* $\mathscr{H}$*. Given a permutation matrix* $\mathbf{P}$*, then*

$$\mathbf{\Phi}(\mathbf{P}^\top\mathbf{x};\mathbf{P}^\top\mathbf{S}\mathbf{P};\mathscr{H}) = \mathbf{P}^\top\mathbf{\Phi}(\mathbf{x};\mathbf{S};\mathscr{H}). \tag{2.25}$$

That is, the output of the GCNN computed using the mapping $\mathbf{\Phi}(\mathbf{x};\mathbf{S};\mathscr{H})$ *is equivariant to permutations.*

Proposition 1 shows the output of a GCNN is invariant to node relabeling and, more importantly, GCNNs exploit graph symmetries to generalize learned representations to different graph signals that share some of these symmetries. This is a beneficial aspect, as multiple symmetric signals can be processed by only seeing one instance out of these during training. An example is graphically depicted in Figure 2.9.

Permutation equivariance is an important aspect to account for when designing activation functions for GCNNs and it restricts the family of activation functions that can be employed. Pointwise activation functions are by definition permutation equivariant. Thus, GCNNs employing pointwise nonlinearities (i.e., traditional CNNs nonlinearities and KAFs) preserve the permutation equivariance property and are invariant to node relabeling. The localized activation functions preserve this property as well, as proved in [12]. In this thesis, we are concerned with activation functions that make the GCNN equivariant. When designing the proposed graph-adaptive activation functions, we will aim to maintain this property.

2.9. CONCLUSION

In this chapter, we discussed the background material necessary for the research carried out in this thesis. We introduced basic concepts from graph signal processing, followed by the graph shift operator, graph filters and their role in graph convolutions. We detailed next the Graph Fourier Transform and discussed the link between consensus and graph signal filtering, which we will further exploit in one of the numerical experiments carried out in this research. Next, we presented GCNNs, as well as different activation functions employed with GCNNs. Finally, we addressed GCNN properties, such as an overview of the learnable parameters and the permutation equivariance property.

3 Literature Review

In this chapter, we will provide an overview of the relevant literature for the research carried out in this thesis. We will start with a discussion on existing Graph Neural Network approaches and their main properties in Section 3.1. In Section 3.2, we will present the activation functions that have been employed in the literature and discuss their properties and limitations. Subsequently, in Section 3.3, we will review the relevant literature regarding distributed signal processing over graphs, with a particular focus on the distributed finite-time consensus, which we will also address in this thesis.

3.1. GRAPH NEURAL NETWORKS

THE development of neural network architectures that can work with irregular data has seen an increased interest over the past years. Since these data are best encoded in graphs, Graph Neural Networks (GNNs) were developed. GNNs were designed to reproduce the effectiveness that Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) proved to have in processing spatial and temporal data.

The effectiveness of GNNs was demonstrated for the first time in [27] and [28], where GNNs were implemented as a recursive aggregation of labels over nodes' neighbors, combined with pointwise nonlinearities. Their convolutional counterparts, the Graph Convolutional Neural Networks (GCNNs), consider exploiting graph convolutions [7] to perform label aggregations and they extend CNNs [29] beyond Euclidean domains, to the graph domain. We identify different research directions in the literature based on the methods employed for aggregating the neighboring information. We will further detail the proposed models for each of these categories.

FINITE IMPULSE RESPONSE (FIR) GRAPH FILTERS.

Works in this category use polynomial graph filters to aggregate incoming information from neighbors. GCNNs were first introduced in [30], where, building on spectral graph theory, graph convolutions were defined by multiplying feature representations in the Laplacian eigenspace with trainable kernels. Subsequently, the work in [31] avoids the cost and numerical instability of spectral decomposition by using FIR graph filters to combine features in the vertex domain by means of a Chebyshev polynomial [32] on the Laplacian matrix. The work in [33] follows the same idea but builds a polynomial filter in any graph representation matrix (e.g., adjacency, Laplacian). The method proposed in [34] is based on [31] and proposes an order one Chebyshev polynomial to aggregate information. Specifically, a layer-wise propagation rule is introduced that operates directly on the graph structure encoded in the neural network model and aggregates the nodes' features with their neighbors' features.

AUTOREGRESSIVE MOVING AVERAGE (ARMA) GRAPH FILTERS.

Works in this category use ARMA graph filters to aggregate neighboring information. The work in [35] builds a GCNN with distributable ARMA graph filters [9], which capture a broader family of functions at the expense of computation cost. The authors argue that ARMA filters are more robust and have a more flexible graph frequency response than polynomial filters. The work in [36] considers an ARMA GCNN as well, which generalizes rational functions based on Cayley polynomials [37]. These allow for efficient computation of spectral filters on graphs [37].

NODE VARYING GRAPH FILTERS.

In this category, we consider the work in [38], which extends CNNs to graph signals by employing node varying graph filters. To keep the number of parameters independent of the data size, the authors also introduce hybrid node varying graph filters. These are implemented by grouping the nodes and using the same filter coefficients for the nodes belonging to the same group. The node varying graph filters can extract features from different resolutions by only applying local exchanges at each node.

ATTENTION MECHANISM.

Parallel to the previously discussed works, a more recent direction in the literature focuses on combining neighboring information through attention mechanisms. [39] proposes Graph Attention Networks (GATs) that, by leveraging attention-like mechanisms, address the limitations of priorly discussed methods based on graph convolutions and their approximations. More specifically, they allow for implicitly assigning different weights to different neighbors of a node by stacking multiple layers in which nodes can visit their neighbors' features. An extension of GATs is provided in [36], where the authors propose Graph Convolutional Attention Networks (GCATs). Just as GCNNs, these employ convolutional filters, but utilizing a layer-specific matrix that could differ from the graph shift operator.

EDGE VARYING RECURSIONS AND HYBRID FILTERS.

The work in [36] employs edge varying recursions to propose a generic framework for GNNs. The authors show that all the aforementioned architectures for GCNNs, as well as GATs, are equivalent and can be encompassed under the same formulation. They argue these approaches can be seen as particular cases of edge varying GNNs (EdgeNets), which are architectures that allow each node to assign different weights to information coming from different direct neighbors. The authors also propose hybrid approaches employed using hybrid filters. These are defined as a linear combination between convolutional and edge varying filters that only operate based on a subset of nodes.

It is important to highlight that the discussed works capture the data-graph topology coupling only linearly through graph filters. They ignore the coupling in the nonlinear pointwise component (e.g., ReLU). We will discuss this limitation in the following section of this chapter, as well as throughout the research carried out in this thesis.

3.1.1. PERMUTATION EQUIVARIANCE

An essential property of GNNs is *permutation equivariance* [2, 13, 40]. Permutation equivariance captures the fact that, as nodes in a graph are naturally unordered, the GNN output should be unaffected by the ordering. That is, any change happening in the node ordering should be accompanied by a corresponding reordering of the GNN output. The permutation equivariance property guarantees GNNs' independence of node labeling. It also implies that GNNs are capable of exploiting the internal symmetries present in the graph signals [2, 12]. This means that, by learning how to process a node with a certain topological neighborhood, GNNs can further generalize to any other node with the same topological neighborhood.

Depending on the implementation approach, GNNs can preserve or not permutation equivariance. As discussed in [2], any architecture employing graph convolutional filters are permutation equivariant. From the GNN approaches discussed previously, the node varying, edge varying, hybrid edge varying, and some of the architectures employing ARMA filters (for low Jacobi orders) do not preserve the permutation equivariance property, as explained in [36].

3.2. ACTIVATION FUNCTIONS IN GRAPH NEURAL NETWORKS

The activation function is an essential component of GNNs and the way it is implemented has an impact on the output. We identify two main research directions in the literature based on the activation functions employed with GNNs:

1. GNNs with graph-independent pointwise activation functions;
2. GNNs with graph-based localized activation functions.

POINTWISE ACTIVATION FUNCTIONS

The common approach in the GNN literature is to apply traditional CNN pointwise activation functions, such as the Rectified Linear Unit (ReLU) [41], the sigmoid [42], or the hyperbolic tangent (tanh) [43]. In GNNs, these pointwise approaches are applied locally at each node [31, 34]. Attempting to improve on these traditional pointwise activation functions, the authors in [11] investigate more complex pointwise nonlinearities for GCNN that can adapt to the training data. This work proposes kernel activation functions (KAFs), initially introduced in [26] for GNNs and further extended in [11] for GCNNs. These are defined as activation functions modeled in terms of a kernel expansion over a dictionary. The dictionary elements are chosen ahead, such that each function can be represented with only a set of linear coefficients. These coefficients are adapted from the data together with the weights of the convolutional layers by means of backpropagation. KAFs increase the flexibility of GCNNs, as they allow modeling a larger class of functions by adapting the activation functions themselves from the training data. However, KAFs are limited when applied to GCNNs because they are applied individually at each node. Therefore, they ignore the data-graph topology coupling.

While traditional pointwise activation functions and KAFs may be relevant to CNNs, because in an image or time signal the neighborhoods are always the same, they ignore that the neighbors and topology can vary significantly from graph to graph. Therefore, it became essential to design activation functions that account for the underlying graph topology in the learning process.

LOCALIZED ACTIVATION FUNCTIONS

Addressing the necessity of designing activation functions that can adapt to the graph, the authors in [12] propose localized activation functions. Besides accounting for the graph topology, these activation functions are also multiresolution and trainable. This is achieved by aggregating information incoming from one-hop up to K-hop neighborhoods and assigning different weights to different neighborhood resolutions. Specifically, the work in [12] introduced two localized activation functions by employing the maximum and median graph filters [44, 45]. Since the maximum and median graph filters perform a parametric nonlinear aggregation of the information coming from different neighborhood resolutions, they implicitly capture the graph structure and adapt to the training data. This is achieved by assigning different weights to different neighborhood resolutions. Moreover, the median and max localized nonlinearities enhance GNNs by extracting nonlinear features within localized neighborhoods for each node, extending the linear features extracted through graph convolutions. The localized activation functions also preserve the permutation equivariance property of GNNs, previously discussed in Section 3.1.1. This is important because they preserve the GNNs' capability of learning features independently of the graph labeling and of exploiting internal signal symmetries.

While accounting for the graph structure, localized activation functions have two main limitations. Firstly, when aggregating the graph signal within a neighborhood, they do not account for the edge weights and treat all nodes equally. By only accounting for the proximity between nodes, and not for the strength of the connection between them, localized activation functions ignore the data-graph topology coupling. Secondly, information from multiresolution neighborhoods is aggregated by directly accessing the signal value at the multi-hop neighbors. This means that when the neighborhood resolution is higher than one, each node in the graph accesses information that is not locally available. Thus, localized activation functions are not distributable beyond one-hop neighborhoods. When employed in distributed applications, these miss relevant information from far away neighbors, which is essential in capturing both local and global details during learning.

3.3. Distributed Processing over Graphs

Data generated by nodes in network applications such as robot swarms [46], sensor networks [47–49], or multi-agent control [14, 15, 18] need to be processed distributively, as the nodes do not always have access to a central processing unit. This is due to different limitations, such as privacy concerns or limited communication resources. Thus, these data have to be processed locally in the network. Different approaches can be identified in the literature when addressing distributed processing over graphs dealing with different applications. However, lately, graph signal processing (GSP) [7, 21] has become a popular tool in addressing distributed processing over graphs, mainly through graph filters. It is important to note that these are linear methods and, thus, they cannot capture the nonlinear relations in the data. Only recently, learning based methods have been proposed. Next, we will review the works employing GSP in distributed applications over graphs, based on graph filters, as well as recent works employing GNNs.

In [50], the authors study how graph filters can be used to implement network linear operations in a distributed fashion, such that each node in the network needs to exchange a finite number of messages with its direct neighbors. The investigation is carried out within the framework of shift-invariant graph filters. Their framework is applied to the problem of finite-time consensus. Moreover, the work in [10] further studies the optimal design of graph filters to implement arbitrary linear transformations between graph signals. The authors argue that, since the graph filters capture the graph structure by employing the graph shift operator, they naturally give rise to distributed linear network operators. The authors demonstrate the relevance of their research in the field of distributed applications by addressing the finite-time consensus and analog network coding problems. The work in [8] proposes a method for distributing the application of unions of graph multiplier operators, which are linear operators for processing graph signals. Specifically, the authors propose approximating the graph multipliers by using shifted Chebyshev polynomials. They show that several distributed signal processing tasks can be represented as distributed applications of unions of graph multiplier operators, such as semi-supervised learning, smoothing, denoising, or inverse filtering.

Another approach to distributed graph signal processing tasks is employing infinite impulse response (IIR) graph spectral filters. In [49], the authors propose a family of IIR graph filters and provide an approach for applying these distributively in the context of wireless sensor networks. They demonstrate through numerical experiments that IIR graph filters are more accurate when approximating ideal graph filters and more robust against network changes than FIR graph filters. Additionally, in [51], the authors introduce the autoregressive moving average (ARMA) graph filters and demonstrate their implementation in a distributed fashion. The ARMA graph filters are further studied in [9] in the context of distributed graph filtering problems. The authors design a collection of ARMA recursions, which can approximate any graph frequency response and give exact solutions in graph signal denoising and interpolation tasks.

To address the communication and computational complexity of distributed implementation of graph filters, edge-varying graph filters were proposed. Initially introduced in [52], and later analyzed in [22], these are a generalization of state-of-the-art graph filters, in which every node in a network weights differently the signal coming from different neighbors. Through numerical experiments covering graph filter approximation, distributed linear operator approximation, and denoising [22] confirm the benefits of edge-varying graph filters over existing methods and show their potential for a broader range of applications, beyond graph filtering. Moreover, the work in [53] introduces a cascade implementation of the edge varying graph filters to address the numerical stability issues involved in their design. The authors also propose an algorithm for efficiently learning the filter coefficients. They demonstrate the efficiency of their proposed approach by experimenting with the distributed finite-time consensus task and achieving lower communication cost than the existing works in the literature.

More recently, learning based approaches have been proposed in the literature for addressing distributed processing tasks over graphs. The work in [46] employs an aggregation GNN, initially introduced in [33], for learning distributed controllers to coordinate robot swarms with limited access to communication resources. The aggregation GNN is extended to time-varying signals and graph topologies to deal with changes in the robot network over time. Their numerical experiments confirm the importance of incorporating information incoming from multi-hop neighbors in a distributed fashion to ensure good performance and robustness. In [54], the authors propose a distributed learning model that is robust to link losses in distributed settings. Specifically, they introduce a stochastic GNN (SGNN) model in which the distributed graph convolutional operator is implemented using random graph filters [55] and it accounts for link losses during training. Their numerical results, addressing distributed source localization, show an improved performance of the proposed SGNN compared to the traditional GNN model. Moreover, [56] puts forward an architecture applicable to distributed online learning scenarios, the Wide and Deep Graph Neural Network (WD-GNN), and implements a distributed online learning algorithm. The numerical experiments focus on learning a decentralized controller for robot swarm coordination. The results show the potential and efficiency of the WD-GNN in distributed online learning. Additionally, the authors analyze the stability of the WD-GNN to changes in the graph topology and demonstrate the transferability of this model to unseen testing setups in an online, distributed fashion.

As pointed out before and detailed throughout the discussion carried out in this section, many approaches present in the literature address the distributed processing over graphs mostly through graph filters, which are linear methods. Only a few works recently proposed learning based approaches using GCNNs, which can capture the nonlinear relations in the data. These are, however, limited to a small number of applications. One of our goals through the research carried out in this thesis is to introduce a nonlinear data-driven distributed framework, employing graph-adaptive activation functions, to extend the applicability of GCNNs to more distributed applications over graphs. One of the tasks that we address is distributed finite-time consensus. We discuss next the methods proposed in the literature for solving this task.

Finite-Time Consensus

Distributed finite-time consensus is a fundamental problem in signal processing, sensor networks, and multi-agent control [4, 5, 14–18]. This task aims to achieve average consensus among all nodes in a network in finite-time, by accessing only local information at each node.

A first approach to reach consensus is through distributed iterative solvers, such as randomized gossip [57] or methods of multipliers [58]. These methods reach consensus at steady-state, and their convergence rate is dominated by the network topology. A more recent direction considers reaching consensus within a finite number of iterations and frames this problem as a graph filtering operation [7].

The first work to formalize finite-time consensus through graph filters is [3]. This work uses the FIR graph filters and designs the filter coefficients by relying on the spectrum of the graph Laplacian matrix. Conditions on when consensus is feasible over a generic graph are analyzed in [10, 23]. The work in [10] focuses on the optimal design of graph filters to implement arbitrary linear operators in the distributed setup of finite-time consensus. The work in [23] analyzes the limits of distributed finite-time consensus by means of graph filters and matrix function theory. The main limitation of these theoretical contributions is that the filter coefficients depend on the specific eigenvalues of the graph Laplacian matrix. The cost of computing the eigendecomposition also limits their applicability to graphs of small dimensions. The designed filters also suffer from numerical issues due to the finite-precision of the eigenvalues. Besides the theoretical insights, the practical benefit of these works is to approximate better consensus in a finite number of iterations compared with the other distributed solvers. The fastest converging filter to reach consensus is the edge varying graph filter [22], which, differently from the FIRs, exploits also node's locality and sparsity to enhance the degrees of freedom. However, the edge varying graph filter requires all nodes to know the labeling of the graph structure and this structure to be fixed; both assumptions that might be infeasible in practical deployments or when the topology changes slightly in distributed settings (e.g., nodes and links that fall).

In this thesis, we propose a data-driven framework for addressing finite-time consensus with GCNNs, by exploiting the link between consensus, graph convolutional filters, and GCNNs. We propose a method that is readily distributable if the activation functions employed allow for a distributed implementation.

3.4. DISCUSSION

In this chapter, we provided an overview of the relevant literature related to the topics discussed in this thesis. In Section 3.1, we discussed the extension of neural network models to data defined in irregular domains (graphs) and presented the different approaches for implementing GNNs and their convolutional counterparts. We also highlighted that all approaches discussed capture the coupling between the data and the graph topology only in the linear component through the graph filters, while this coupling is ignored in the nonlinear component. In Section 3.1.1, we discussed permutation equivariance, a property of GNNs that guarantees independence on node labeling and allows GNNs to exploit internal graph symmetries. Subsequently, in Section 3.2, we identified the two main directions for the activation functions employed in the existing GNN literature, namely the pointwise approach and the localized approach. The pointwise activation functions include the traditional ReLU, sigmoid, and tanh, as well as the activation functions via kernel transforms. More recently, localized activation functions have been proposed to account for the graph topology, which pointwise solutions ignore. Finally, in Section 3.3, we provided an overview of the different approaches dealing with distributed processing of graph signals. The majority of works in this area are based on graph filtering approaches and only a few focus on applying GNNs to distributed graph signal processing tasks. We further reviewed in Section 3.3 the literature dealing with finite-time consensus, a central problem in distributed signal processing. Current works propose a graph filtering based approach and, to the best of our knowledge, distributed finite-time consensus is addressed from a GNN perspective in this thesis for the first time.

In this thesis, we address the design of an activation function that is both graph-adaptive and allows for a distributed implementation, while preserving the permutation equivariance property of GNNs. The proposed method is different from pointwise approaches in the sense that it captures the graph topology in the nonlinear component of the GNN, thus accounting for the variations in the neighborhood structures. The proposed method also improves on localized activation functions by considering the edge weights in the graph and implicitly accounting for the data-graph topology coupling. Moreover, in designing the graph-adaptive activation functions, we will research the possibilities of embedding the graph structure in the nonlinear component of GNNs in a form akin to graph convolutions. This will allow the graph-adaptive nonlinearities to operate only within one-hop neighborhoods. Thus, unlike localized activation functions, the proposed method can be implemented distributively while capturing information from arbitrarily large neighborhoods. This allows us to further design a new GNN framework that accounts for the data-graph coupling also in the nonlinear component and that is distributable, so we can employ it in distributed graph processing tasks. Thus, we can improve on the existing literature by tackling these tasks from a learning perspective and capturing the nonlinear relations in the data.

4 Graph-Adaptive Activation Functions

This chapter contains the core contribution of the thesis. In essence, we propose a new family of nonlinearities that account for the data-graph topology coupling in a form akin to graph convolutions. This allows for capturing information incoming from arbitrarily large neighborhood resolutions, while applying the nonlinear operator only in the one-hop neighborhood of a node. These nonlinear features are subsequently combined with a set of trainable parameters to weight the information at different neighborhood resolutions accordingly. The resolution radius is a design parameter and allows adapting the GNN nonlinear component to the task at hand. First, in Section 4.1 we define the *graph-adaptive localized activation functions*, which are based on arbitrary nonlinear operators. Then, in Section 4.2, we define the *graph-adaptive kernel activation functions*. Finally, in Section 4.3 we discuss properties of the proposed graph-adaptive activation functions and prove these are permutation equivariant.

4.1. GRAPH-ADAPTIVE LOCALIZED ACTIVATION FUNCTIONS

To start, let us first define the basic building block for graph-adaptive activation functions: the *shifted localized operator* (SLO).

Definition 3 (Shifted Localized Operator) *Let $\mathcal{G}$ be an N-node graph with shift operator $\mathbf{S}$, $\mathbf{x}$ a signal, and $\mathbf{S}^k\mathbf{x}$ the kth shifted signal. Consider an arbitrary nonlinear localized function $f(\cdot,\mathcal{N}_i):\mathbb{R}^N\to\mathbb{R}^N$, which at node i computes the local nonlinear operation $[f(\mathbf{x},\mathcal{N}_i)]_i = f(\{x_j\}_{j\in\mathcal{N}_i})$ w.r.t its neighbors $\mathcal{N}_i$. The k-hop shifted localized operator maps input $\mathbf{x}$ to output $\mathbf{z}\in\mathbb{R}^N$ as*

$$z_i = [f(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i = f(\{[\mathbf{S}^k\mathbf{x}]_j : j\in\mathcal{N}_i\}),\ \textit{for } i=1,\ldots,N. \tag{4.1}$$

That is, the SLO shifts the signal k times to obtain $\mathbf{S}^k\mathbf{x}$, and then replaces the value of this signal at each node i by a nonlinear aggregation $f(\cdot,\mathcal{N}_i)$ of the signal values within the one-hop neighborhood of i, $\mathcal{N}_i$. The SLO utilizes information locally available at each node to account for the signal-topology coupling for nodes that are k-hops away. For the choice of $f(\cdot,\mathcal{N}_i)$ any nonlinear aggregation function can be applied. In this thesis, we employ the max and median operators, defined next.

Shifted Localized Max Operator. Let $\mathcal{G}$ be an N-node graph with shift operator $\mathbf{S}$, $\mathbf{x}$ a signal, and $\mathbf{S}^k\mathbf{x}$ the kth shifted signal. Consider also the one-hop neighborhood of a node i, $\mathcal{N}_i$, for $1\le i\le N$. The k-hop shifted localized max operator maps input $\mathbf{x}$ to output $\mathbf{z}\in\mathbb{R}^N$ as

$$z_i = [\max(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i = \max(\{[\mathbf{S}^k\mathbf{x}]_j : j\in\mathcal{N}_i\}),\ \text{for } i=1,\ldots,N. \tag{4.2}$$

Shifted Localized Median Operator. Let $\mathcal{G}$ be an N-node graph with shift operator $\mathbf{S}$, $\mathbf{x}$ a signal, and $\mathbf{S}^k\mathbf{x}$ the kth shifted signal. Consider also the one-hop neighborhood of a node i, $\mathcal{N}_i$, for $1\le i\le N$. The k-hop shifted localized median operator maps input $\mathbf{x}$ to output $\mathbf{z}\in\mathbb{R}^N$ as

$$z_i = [\text{median}(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i = \text{median}(\{[\mathbf{S}^k\mathbf{x}]_j : j\in\mathcal{N}_i\}),\ \text{for } i=1,\ldots,N. \tag{4.3}$$

Figure 4.1 illustrates the application of the shifted localized max and median operators. These are employed to aggregate a kth shifted signal within the one-hop neighborhood of a node.

Based on the SLO, we can now define *shifted localized graph filters* as follows.

Definition 4 (Shifted Localized Graph Filter) *Consider the shifted localized operator induced by an arbitrary nonlinear localized function $f(\cdot,\mathcal{N}_i)$ [cf. Def. 3], and let $\mathbf{h}_\sigma = [h_{\sigma 1},\ldots,h_{\sigma K}]^\top$ be a vector of parameters. The output of the shifted localized graph filter applied to signal $\mathbf{x}$, w.r.t. the shift operator $\mathbf{S}$, is the signal $\mathbf{z}\in\mathbb{R}^N$ with ith entry*

$$z_i = \sum_{k=1}^{K} h_{\sigma k}[f(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i,\ \textit{for } i=1,\ldots,N. \tag{4.4}$$

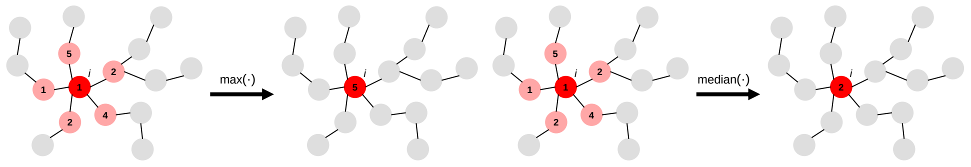


Figure 4.1: Illustration of applying the shifted localized max operator [cf. (4.2)] and shifted localized median operator [cf. (4.3)]. Both operators are applied for node i (highlighted in red) to aggregate the kth shifted signal values within its one-hop neighborhood (highlighted in light red). (*a*) Aggregation of the one-hop neighborhood signal of node i by employing the shifted localized max operator [cf. (4.2)]. (*b*) Aggregation of the one-hop neighborhood signal of node i by employing the shifted localized median operator [cf. (4.3)].

Definition 4 implies the output of a shifted localized graph filter is a linear combination of the SLOs $f(\cdot, \mathcal{N}_i)$ at different resolutions. Hence, shifted localized graph filters inherit the localization property of SLOs, as they incorporate the graph structure up to K-hops away accessing only neighboring information. These nonlinear filters can be employed to define *graph-adaptive localized activation functions*.

Definition 5 (Graph-Adaptive Localized Activation Function) *Consider a scalar β and vector $\mathbf{h}_\sigma = [h^f_{\sigma l1}, \ldots, h^f_{\sigma lK}]^\top$ of learnable parameters. At layer l, the graph-adaptive localized activation function maps the linear features $\mathbf{z}^f_l$ [cf. 2.7] to the output features $\mathbf{x}^f_l$ following the recursion*

$$[\mathbf{x}^f_l]_i = \beta ReLU([\mathbf{z}^f_l]_i) + \sum_{k=1}^{K} h^f_{\sigma lk}[f(\mathbf{S}^k\mathbf{z}^f_l, \mathcal{N}_i)]_i, \; for\; i = 1, \ldots, N. \tag{4.5}$$

Definition 5 combines the pointwise ReLU nonlinearity and the shifted localized graph filters [cf. Def. 4] into a single graph-adaptive localized nonlinearity for GNNs. The latter is distributable and localized because, even though the resolution —given by the shift order K— can be arbitrarily large, the SLO $f(\cdot, \mathcal{N}_i)$ [cf. Def. 3] operates only in the one-hop neighborhood. In Chapter 4, we evaluate this activation function for $f(\cdot, \mathcal{N}_i)$ being the max and median, leading to the *graph-adaptive max activation function* and *graph-adaptive median activation function*, respectively.

4.2. Graph-Adaptive Kernel Activation Functions

The graph-adaptive kernel activation functions replace the localized nonlinear function $f(\cdot, \mathcal{N}_i)$ by a localized *kernel* to increase the flexibility of the graph-adaptive nonlinearities and enrich their representation power.

Let $\mathbf{x}^{(k)}_i \in \mathbb{R}^{|\mathcal{N}_i|}$ denote the vector containing $|\mathcal{N}_i|$ copies of the kth shifted signal at node i, $[\mathbf{S}^k\mathbf{x}]_i$, i.e. $\mathbf{x}^{(k)}_i = \mathbf{1}_{|\mathcal{N}_i|} \otimes [\mathbf{S}^k\mathbf{x}]_i$ where $\mathbf{1}_{|\mathcal{N}_i|}$ is the vector of ones of dimension $|\mathcal{N}_i|$ and $\otimes$ the Kronecker operator. Consider also the vector containing the values at neighbors $j \in \mathcal{N}_i$ of the kth shifted signal $\mathbf{S}^k\mathbf{x}$, i.e. $\mathbf{x}^{(k)}_{j\in\mathcal{N}_i} = [\mathbf{S}^k\mathbf{x}]_{j\in\mathcal{N}_i}$. With this notation in place, we define a graph kernel operator as follows.

Definition 6 (Kernel Operator) *Let $\mathcal{G}$ be an N-node graph with shift operator $\mathbf{S}$, $\mathbf{x}$ a signal, and $\mathbf{S}^k\mathbf{x}$ the kth shifted signal. Consider an arbitrary kernel function $g(\cdot,\mathcal{N}_i) : \mathbb{R}^{|\mathcal{N}_i|} \to \mathbb{R}^{|\mathcal{N}_i|}$, which at node i computes the nonlinear local operation $[g(\mathbf{x},\mathcal{N}_i)]_i = g(\widetilde{\mathbf{x}}_i, \mathbf{x}_{j\in\mathcal{N}_i})$, where $\widetilde{\mathbf{x}}_i = \mathbf{1}_{|\mathcal{N}_i|} \otimes [\mathbf{x}]_i$ is a vector of dimensionality $|\mathcal{N}_i|$ containing copies of signal $\mathbf{x}$ at node i. The k-hop shifted kernel operator mapping from $\mathbf{x}$ to $\mathbf{z} \in \mathbb{R}^N$ has the entries*

$$z_i = [g(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i := g(\mathbf{x}^{(k)}_i, \mathbf{x}^{(k)}_{j\in\mathcal{N}_i}),\ \textit{for}\ i = 1,\ldots,N. \tag{4.6}$$

Definition 6 shows the kernel operator first shifts the input signal $\mathbf{x}$ as $\mathbf{S}^k\mathbf{x}$ and then replaces the signal value at each i by the kernel value $g(\cdot,\mathcal{N}_i)$ in the one-hop neighborhood of i. Thus, the kernel operator employs only local information at each node to account for the signal-topology coupling up to k-hops away from a node. For the choice of $g(\cdot,\mathcal{N}_i)$ any generic kernel function can be applied. In this thesis, we will employ the Gaussian kernel, defined as

$$g(\mathbf{x},\mathbf{y}) = \exp\left(-||\mathbf{x}-\mathbf{y}||^2/2\gamma^2\right) \tag{4.7}$$

where scalar γ is tunable. For more possibilities to implement the kernel function, we refer the reader to Appendix A.

Based on the kernel operator, we can now define *kernel graph filters* as follows.

Definition 7 (Kernel Graph Filter) *Consider a kernel operator [cf. 6] with kernel function $g(\cdot,\mathcal{N}_i)$ and let $\mathbf{h}_\sigma = [h_{\sigma 1},\ldots,h_{\sigma K}]^\top$ be a vector of parameters. The output of the kernel graph filter applied to signal $\mathbf{x}$, w.r.t. the shift operator $\mathbf{S}$, is the signal $\mathbf{z} \in \mathbb{R}^N$ with ith entry*

$$z_i = \sum_{k=1}^{K} h_{\sigma k}[g(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i,\ \textit{for}\ i = 1,\ldots,N. \tag{4.8}$$

Definition 7 implies the output of the kernel graph filter is a linear combination of the kernel operator $g(\cdot,\mathcal{N}_i)$ applied to different resolutions. Kernel graph filters thus preserve the localization properties of kernel operators, i.e., they account for the topology of the graph up to K-hops away accessing only information in the one-hop neighborhood. These kernel graph filters can be further employed to define the *graph-adaptive kernel activation function* as follows.

Definition 8 (Graph-Adaptive Kernel Activation Function) *Consider a scalar β and vector $\mathbf{h}_\sigma = [h^f_{\sigma l1},\ldots,h^f_{\sigma lK}]^\top$ of learnable parameters. At layer l, the graph-adaptive kernel activation function maps the linear features $\mathbf{z}^f_l$ [cf. (2.7)] to the output features $\mathbf{x}^f_l$ following the recursion*

$$[\mathbf{x}^f_l]_i = \beta ReLU([\mathbf{z}^f_l]_i) + \sum_{k=1}^{K} h^f_{\sigma lk}[g(\mathbf{S}^k\mathbf{z}^f_l,\mathcal{N}_i)]_i,\ \textit{for}\ i = 1,\ldots,N. \tag{4.9}$$

Definition 8 combines the pointwise ReLU and kernel graph filters [cf. Def. 7] into a single graph-adaptive kernel activation function. This activation function is distributable and localized because, even though the resolution —given by the shift order K— can be arbitrarily large, the kernel $g(\cdot,\mathcal{N}_i)$ operates only in the one-hop neighborhood.

In the proposed graph-adaptive localized and kernel activation functions, coefficients $\{\beta, \mathbf{h}_\sigma\}$ are trainable, meaning these nonlinearities adapt the multi-hop resolution weights to the task at hand. Because these coefficients are shared among nodes, the number of parameters to learn for a graph-adaptive activation function is independent of the graph size. This allows GCNNs to *scale*. Note that even though the nonlinear functions $f(\cdot, \mathcal{N}_i)$ or the kernel functions $g(\cdot, \mathcal{N}_i)$ act only on the one-hop neighborhood, they are applied to the shifted signals $\mathbf{S}^k\mathbf{x}$. Therefore, they account for the feature-graph coupling (up to K-hops away) in a nonlinear fashion. This is an advantage over traditional GCNNs with pointwise nonlinearities, in which the graph topology is only incorporated through linear encodings generated by graph convolutions. Moreover, since the graph-adaptive nonlinearities only operate on the one-hop neighborhood, they allow for a distributed implementation, unlike the localized nonlinearities proposed in [12]. An example of how the graph-adaptive activation functions operate is shown in Figure 4.2. We use a maximum neighborhood resolution of three-hops to highlight the difference with the localized nonlinearities illustrated before in Figure 2.8.

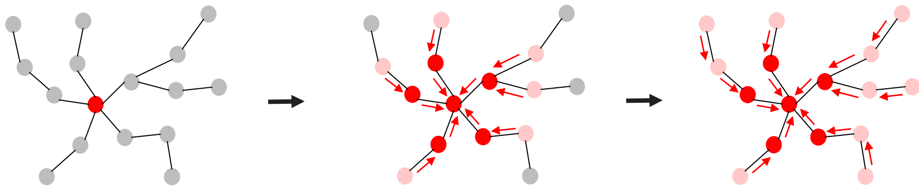


Figure 4.2: Graph-adaptive activation functions applied for a maximum neighborhood resolution of three-hops. The nonlinearity is applied for the node initially highlighted in red. The signal-graph topology coupling within multiresolution neighborhoods is captured by diffusing the graph signal (illustrated with the red arrows) prior to the nonlinear aggregation. The latter is only applied in the one-hop neighborhood (illustrated in red), while also incorporating information from further away neighbors (illustrated in pink).

Definitions 5 and 8 implement *fully graph-adaptive* GCNNs that, at each layer, apply a graph convolution followed by a graph-adaptive activation function. The distributed GCNN is given by the map

$$\mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}, \mathcal{W}) := \tilde{\mathbf{y}} \tag{4.10}$$

which depends on both the coefficients $\mathcal{H}$ [cf. (2.11)] and on the activation function coefficients $\mathcal{W} = \{\mathbf{h}_{\sigma l}^{f}\}_{lf} \cup \{\beta\}$.

4.3. PROPERTIES OF GRAPH-ADAPTIVE ACTIVATION FUNCTIONS

4.3.1. LEARNABLE PARAMETERS

If graph-adaptive activation functions are employed, $(K+1)L$ extra parameters are added compared to the conventional pointwise activation functions, discussed in Section 2.8.1. This divides as: i) parameter β; ii) K parameters in vector $\mathbf{h}_\sigma$. Thus, the total number of parameters of the GCNN employing graph-adaptive activation functions is $\sum_{l=2}^{l=L} F_{l-1}F_l(K+1) + F_1(K+1) + F_L + (K+1)L$.

4.3.2. PERMUTATION EQUIVARIANCE

A key property GCNNs with pointwise activation functions inherit from graph convolutions is *permutation equivariance* [12]. The output of a GCNN is invariant to node relabeling and, more importantly, GCNNs exploit graph symmetries to generalize learned representations to different graph signals that share some of these symmetries. Herein, we show that permutation equivariance also holds for graph-adaptive nonlinearities.

Permutation equivariance. Consider the graph convolutional filter $\mathbf{H}(\mathbf{S})$ [cf. (2.5)] and let $\mathbf{P}$ be an $N \times N$ permutation matrix satisfying $\mathbf{P}^\top\mathbf{P} = \mathbf{P}\mathbf{P}^\top = \mathbf{I}$. If we permute the GSO $\mathbf{S}$ and input $\mathbf{x}$ respectively as $\mathbf{S}' = \mathbf{P}^\top\mathbf{S}\mathbf{P}$ and $\mathbf{x}' = \mathbf{P}^\top\mathbf{x}$, we get the corresponding graph convolution output

$$\mathbf{y}' = \mathbf{H}(\mathbf{S}')\mathbf{x}' = \mathbf{H}(\mathbf{P}^\top\mathbf{S}\mathbf{P})\mathbf{P}^\top\mathbf{x} = \mathbf{P}^\top\mathbf{H}(\mathbf{S})\mathbf{P}\mathbf{P}^\top\mathbf{x} = \mathbf{P}^\top\mathbf{y}\,. \tag{4.11}$$

4

Because pointwise activation functions are scalar and by definition permutation equivariant, (4.11) implies GCNNs with pointwise nonlinearities are invariant to node relabelings. For GCNNs with graph-adaptive activation functions, it is then desirable to retain this property. This is guaranteed by the following proposition.

Proposition 2 (Permutation equivariance) *Consider a graph signal* $\mathbf{x}$ *defined on an* N*-node graph* $\mathcal{G}$ *with GSO* $\mathbf{S}$ *. Let* $\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H},\mathcal{W})$ *be the output of a GCNN with graph-adaptive activation functions [cf.* (4.10)*] and let* $\mathbf{P}$ *be an* $N\times N$ *permutation matrix. The GCNN* $\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H},\mathcal{W})$ *satisfies*

$$\mathbf{\Phi}(\mathbf{P}^\top\mathbf{x};\mathbf{P}^\top\mathbf{S}\mathbf{P},\mathcal{H},\mathcal{W}) = \mathbf{P}^\top\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H},\mathcal{W}) \tag{4.12}$$

i.e., GCNNs with graph-adaptive activation functions are permutation equivariant.

Next, we will prove the permutation equivariance property of GCNNs employing our proposed graph-adaptive activation functions.

Proof of Prop. 2.
Let $\mathbf{S}' = \mathbf{P}^\top\mathbf{S}\mathbf{P}$ be the permuted shift operator and $\mathbf{x}' = \mathbf{P}^\top\mathbf{x}$ the permuted graph signal. From (4.11), the output of the graph convolution is equivariant to the action of $\mathbf{P}$. Hence, we only need to prove permutation equivariance of the graph-adaptive activation functions in (4.5) and (4.9). We write their output as the signal $\mathbf{z}$ with entries

$$\mathbf{z}_i = \beta\mathrm{ReLU}([\mathbf{x}]_i) + \sum_{k=1}^{K} h_{\sigma k}[g(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i \tag{4.13}$$

where $g(\cdot,\mathcal{N}_i)$ denotes either a shifted localized operator [cf. Def 3] or a kernel operator [cf. Def. 6]. Applying the activation functions in (4.13) to the permuted signal $\mathbf{x}'$, we obtain

$$\mathbf{z}'_i = \beta\mathrm{ReLU}([\mathbf{P}^\top\mathbf{x}]_i) + \sum_{k=1}^{K} h_{\sigma k}[g((\mathbf{P}^\top\mathbf{S}\mathbf{P})^k\mathbf{P}^\top\mathbf{x},\mathcal{N}_i)]_i. \tag{4.14}$$

Since the ReLU activation function is pointwise, it is permutation equivariant, i.e. $\mathrm{ReLU}(\mathbf{P}^\top\mathbf{x}) = \mathbf{P}^\top\mathrm{ReLU}(\mathbf{x})$. We then focus on the second term of the sum, where we observe that

$$(\mathbf{P}^\top\mathbf{S}\mathbf{P})^k = \mathbf{P}^\top\mathbf{S}\mathbf{P}\mathbf{P}^\top\mathbf{S}\mathbf{P}...\mathbf{P}^\top\mathbf{S}\mathbf{P} = \mathbf{P}^\top\mathbf{S}^k\mathbf{P}.$$

This implies

$$(\mathbf{P}^\top\mathbf{SP})^k\mathbf{P}^\top\mathbf{x} = \mathbf{P}^\top\mathbf{S}^k\mathbf{x}.$$

We can rewrite $\mathbf{z}'$ as

$$\mathbf{z}_i' = \beta[\mathbf{P}^\top \text{ReLU}(\mathbf{x})]_i + \sum_{k=1}^{K} h_{\sigma k}[g(\mathbf{P}^\top\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)]_i. \tag{4.15}$$

Because function $g(\cdot, \mathcal{N}_i)$ is localized, it acts on the one-hop neighborhoods of each node, which are preserved under node relabelings. Therefore, $g(\cdot, \mathcal{N}_i)$ is permutation equivariant and (4.15) becomes

$$\begin{aligned}\mathbf{z}_i' &= \beta[\mathbf{P}^\top \text{ReLU}(\mathbf{x})]_i + \sum_{k=1}^{K} h_{\sigma k}[\mathbf{P}^\top g(\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)]_i \\ &= [\mathbf{P}^\top \beta \text{ReLU}(\mathbf{x})]_i + \left[\mathbf{P}^\top \sum_{k=1}^{K} h_{\sigma k} g(\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)\right]_i.\end{aligned}$$

Therefore $\mathbf{z}' = \mathbf{P}^\top\mathbf{z}$ and, hence, GCNNs with graph-adaptive activation functions are permutation equivariant. □

4.4. CONCLUSION

To summarize, we proposed a new family of graph-adaptive activation functions that can be implemented distributively and are defined in terms of nonlinear operators or kernel functions. When the nonlinear operator is employed, the signal value at each node is replaced by a nonlinear aggregation of the signal values within that node's one-hop neighborhood. In the case of the kernel operator, a kernel function is applied within the one-hop neighborhood of each node. These graph-adaptive activation functions incorporate the signal-graph topology coupling into all the GCNNs components. The latter is achieved by combining the aggregated nodal features with a set of trainable parameters in a form akin to convolutions in the GCNN nonlinear component. We also proved that whether in the direct or kernel form, these graph-adaptive activation functions always preserve the permutation equivariance property of GCNNs. Next, we will assess and compare our proposed activation functions with established state-of-the-art GCNN nonlinearities.

5 NUMERICAL EXPERIMENTS

In this chapter, we will describe the numerical experiments and results carried out for evaluating the performance of our proposed graph-adaptive activation functions. More specifically, we consider three experimental tasks: source localization in Section 5.2, where the task is to identify the source community of a graph diffusion process after an unknown number of timestamps; distributed finite-time consensus in Section 5.3, which consists in employing a GCNN to achieve consensus among all nodes in a graph in finite-time; distributed regression in Section 5.4, where GCNNs are applied for distributed signal denoising. We conclude this chapter with a discussion in Section 5.5.

5.1. Overview

We evaluate the performance of six activation functions that include: ReLU, localized activation functions (max and median) [12], and our proposed graph-adaptive localized (max and median) and graph-adaptive kernel activation functions. Our goal is to highlight the benefits and limitations of the different nonlinearities in applications requiring distributed computations with synthetic and real data. The synthetic data employed in our numerical experiments are generated using a Stochastic Block Model (SBM) graph, which we introduce next.

5.1.1. Stochastic Block Model

SBM is a random graph characterized by some community structure. A community is a clustered set of nodes that share some degree of similarity. Communities are built by assigning a higher probability to edges drawn within each community rather than between communities. An SBM graph is, thus, defined by the following parameters:

- Number of nodes N;
- The community sets of nodes $\mathscr{C}_1,\ldots,\mathscr{C}_R$, where R denotes the total number of communities;
- Probability of randomly drawing an edge within a community p;
- Probability of randomly drawing an edge between communities q.

Based on these parameters, each edge is sampled randomly to generate an instance of the SBM graph. A realization of an SBM graph is shown in Figure 5.1 composed of three communities. Throughout our experiments, we vary the SBM parameters to test the generalization ability of our activation functions.

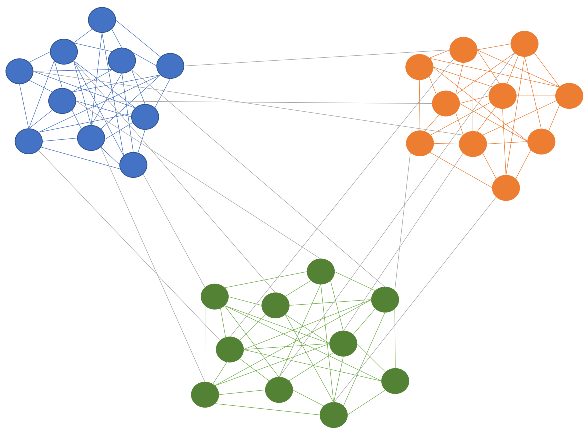


Figure 5.1: Stochastic Block Model (SBM) graph with three communities, each consisting of ten nodes. Each community is depicted in a different color. The inter-community edges are shown in gray.

5.2. Source Localization

5.2.1. Problem Definition

Source localization is a classification problem aiming at identifying the source community of a graph diffusion process after an unknown number of timestamps. Formally, consider an SBM graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with R communities $\mathcal{C}_1, \ldots, \mathcal{C}_R$, as discussed in Section 5.1.1. At timestamp $t = 0$ the graph signal $\mathbf{x}^{(0)}$ is a Kronecker delta signal $\mathbf{x}^{(0)} = \boldsymbol{\delta}_i \in \mathbb{R}^N$ centered at a source node i. That is, $\mathbf{x}^{(0)}$ has a non-zero value only at node i belonging to community $\mathcal{C}_j$, where $1 \leq j \leq R$. As the timestamp t increases up to $t_{\max}$, $\mathbf{x}^{(t)}$ is diffused over the graph by successive application of the GSO $\mathbf{S}$, that is, $\mathbf{x}^{(t)} = \mathbf{S}\mathbf{x}^{(t-1)}$. The GSO $\mathbf{S}$ is the adjacency matrix $\mathbf{A}$ normalized by the highest eigenvalue $\lambda_{\max}$ to avoid numerical instabilities, i.e., $\mathbf{S} = \frac{\mathbf{A}}{\lambda_{\max}}$. Given the graph signal $\mathbf{x}^{(t)}$ at an unknown diffusion timestamp t, we aim to identify the community $\mathcal{C}_j$ containing the node i that originated the initial graph signal $\mathbf{x}^{(0)}$. This task is illustrated in Figure 5.2.

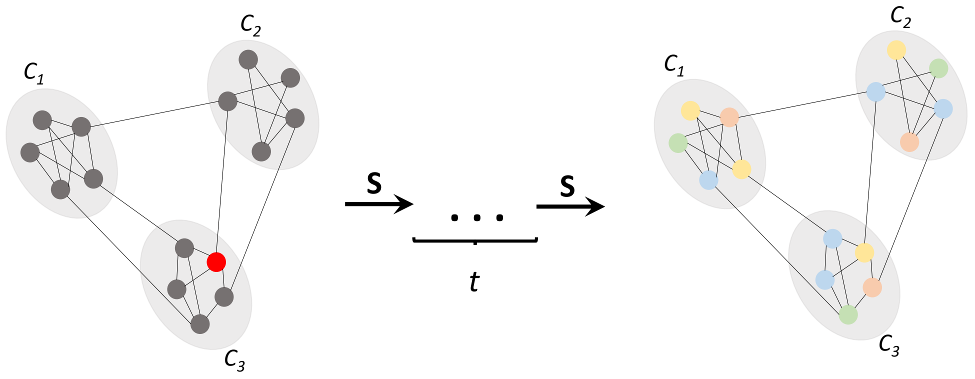


Figure 5.2: Example of the source localization problem. The initial signal $\mathbf{x}^{(0)}$ is denoted in the left SBM graph, where the node in red denotes a signal value of one. The rest of the signal values are zero. After applying the GSO $\mathbf{S}$ for an unknown number of timestamps t, we observe the graph signal $\mathbf{x}^{(t)}$ denoted on the SBM graph on the right. Different colors denote different graph signal values. The goal of the source localization problem is to identify the community that generated the signal observed in the right. In this example, the community is $\mathcal{C}_3$.

Formally, we define the source localization problem as follows

$$\mathbf{x}^{(t)} \xrightarrow{\boldsymbol{\Phi}(\cdot),\mathcal{G}} \mathcal{C}_j.$$

That is, we want to learn $\boldsymbol{\Phi}(\cdot)$ that maps a graph signal $\mathbf{x}^{(t)}$ observed at an unknown timestamp t to the community that originated it, $\mathcal{C}_j$, by exploiting the structure of the graph $\mathcal{G}$.

We considered the source localization problem both in a distributed and centralized scenario. In a distributed scenario, localized activation functions [12] are limited to a filter order of one. Thus, we employed the centralized scenario to compare the performance of our proposed graph-adaptive activation functions with the localized ones up to higher filter orders. More specifically, our research questions for this experiment were:

RQ.1 How do the graph-adaptive activation functions compare with the localized activation functions in a centralized scenario for different filter orders K?

RQ.2 How do the graph-adaptive activation functions perform when we consider the source localization problem in a distributed scenario for different filter orders K?

5.2.2. EXPERIMENTAL SETUP

We employ a different experimental setup for the two scenarios, namely the centralized and distributed classification. This is because, for each experiment, we considered the experimental scenarios introduced by other works in the literature. We will detail the setup for each of the experiments below. For both scenarios, we employed the *classification accuracy* as the evaluation metric. Additionally, to train the parameters, we considered the ADAM optimiser with learning rate 10^{-3} and forgetting factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

CENTRALIZED SOURCE LOCALIZATION

For this experiment, we employed the experimental scenario in [33]. Specifically, we considered a diffusion process over an SBM graph of $N = 100$ nodes divided into $R = 5$ communities. The results we report correspond to intra-community probability $p = 0.8$ and inter-community probability $q = 0.1$ for the SBM graph. For additional results corresponding to $(p, q) \in \{(0.8, 0.2), (0.7, 0, 1)\}$, we refer the reader to Appendix B. We varied the graph connectivities because we wanted to compare the performance of our proposed graph-adaptive activation functions to pointwise and localized activation functions for different graph structures. That is, graphs in which the nodes within the same community are more connected or less connected and in which different communities are more clearly or less clearly separated. The graph signals were defined as Kronecker deltas $\boldsymbol{\delta}_i \in \mathbb{R}^N$ centered at a source node i and diffused at a timestamp $t \in [0, 30]$, i.e. $\mathbf{x}^{(t)} = \mathbf{S}^t \boldsymbol{\delta}_i$. We chose as source node each of the 100 nodes, generating a data set consisting of 3000 graph signals. We split these samples into training, validation, and test set respectively as 80%, 10%, and 10%. We simulated 10 different graphs and generated 10 different splits per graph to account for the stochasticity in the setup. We considered a two layer centralized GCNN, as shown in Figure 2.5, with $F = 5$ features per layer and filters of order $K = 5$. Training was performed for 800 epochs with a batch size of 100 samples. For the localized nonlinearities in [12] and our proposed graph-adaptive nonlinearities, we evaluated resolutions $K \in \{1, 2, 3\}$.

DISTRIBUTED SOURCE LOCALIZATION

For this experiment, we employed the experimental scenario in [54]. We considered a diffusion process over an SBM graph of $N = 50$ nodes divided into $R = 5$ communities. The goal is to determine the source community of a given diffused signal locally at a selected node; hence, the distributed implementation is essential. The intra-community and inter-community probabilities for the SBM graph are $p = 0.8$ and $q = 0.1$, respectively. The graph signals were defined as Kronecker deltas $\boldsymbol{\delta}_i \in \mathbb{R}^N$ centered at a source node i and diffused at a timestamp $t \in [0, 30]$, i.e. $\mathbf{x}^{(t)} = \mathbf{S}^t \boldsymbol{\delta}_i$. We chose as source node each of the 50 nodes, thus generating a data set consisting of 1500 graph signals. We split these samples into training, validation, and test set respectively as 80%, 10%, and 10%.

We simulated 10 different graphs and generated 10 different splits per graph, to account for the stochasticity involved in generating the graph. The training and testing were performed for the highest connected node for each community, resulting in five nodes. We considered a two layer distributed GCNN, as shown in Figure 2.6, with a varying number of features per layer, $F \in \{2,4,8\}$, followed by a per-node fully connected layer. Training was performed for 800 epochs with a batch size of 100 samples. For the localized non-linearities, we considered $K = 1$, while for the graph-adaptive nonlinearities, we carried out the experiments with resolutions $K = 1$ and $K = 2$.

5.2.3. RESULTS

CENTRALIZED SOURCE LOCALIZATION

In this section, we present the results for the centralized source localization experiments. Only the results for the best performing experimental scenario are discussed. These are achieved when the communities are highly intra-connected and clearly separated, specifically for $p = 0.8$ and $q = 0.1$. Additional results are provided in Appendix B.

Table 5.1 shows the accuracies averaged across 100 runs (10 graphs with 10 splits each), together with their associated standard deviations. In Figure 5.3, we show the respective performances with a boxplot.

Table 5.1: Centralized Source Localization Test Accuracy. Graph generated with 0.8 intra-community and 0.1 inter-community edge probabilities. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified. In boldface we show the best performance for each family of nonlinearities.

Activation Function	Average Accuracy	Standard Deviation
ReLU	68.9%	14.6
Max L. (1)	**82.1** %	4.9
Max L. (2)	81.3 %	4.8
Max L. (3)	82.0 %	3.8
Max G.A. (1)	**80.8** %	3.8
Max G.A. (2)	79.5%	4.4
Max G.A. (3)	79.8%	4.5
Median L. (1)	82.0%	4.8
Median L. (2)	82.2%	4.7
Median L. (3)	**82.6%**	4.1
Median G.A. (1)	81.5%	4.3
Median G.A. (2)	80.9%	4.0
Median G.A. (3)	**81.9%**	3.8
Kernel G.A. (1)	**77.5%**	3.7
Kernel G.A. (2)	76.4%	4.2
Kernel G.A. (3)	76.6%	4.2

As we can observe both in Table 5.1 and in Figure 5.3, all the activation functions except ReLU have similar performances. They show an increase in the classification accuracy of at least 7% over ReLU. The lower test accuracy of ReLU, as well as the much

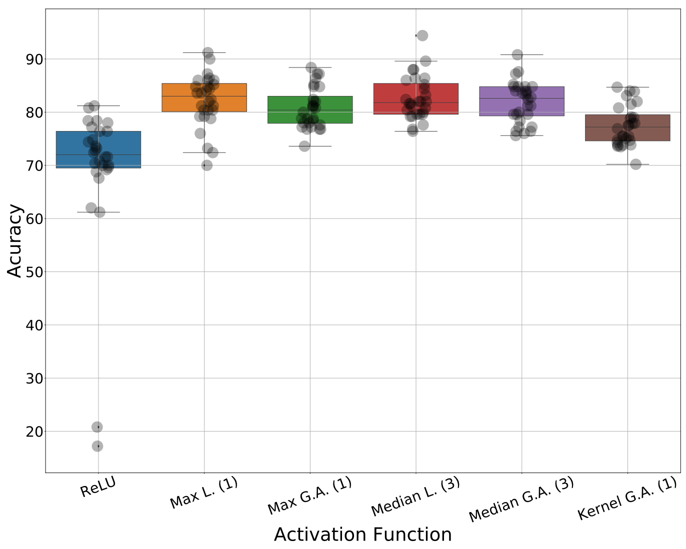
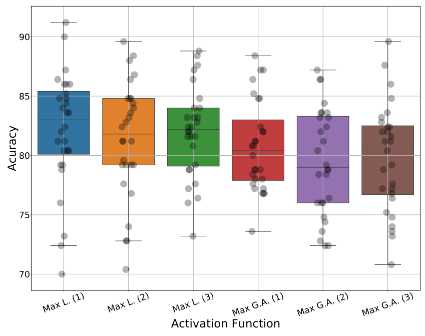


Figure 5.3: Boxplots illustrating the test accuracy of the different activation functions for the centralized source localization problem. For our proposed graph-adaptive and for the localized activation functions only the best performing order is shown. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

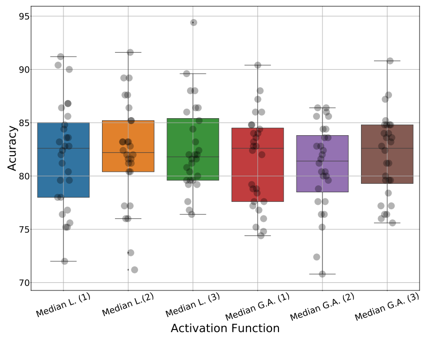
higher standard deviation (as seen in Table 5.1), results from the two instances when ReLU fails to learn, as illustrated in Figure 5.3. However, even when not considering these two occurrences, the performance of the ReLU GCNN is still lower than the GCNN employing localized and graph-adaptive activation functions. This could be explained by the fact that ReLU is applied pointwise, failing to account for the underlying graph topology. Since the graph has clearly separated communities, the neighbors of a node play an important role in propagating the signal. Therefore, it is essential for the activation function to capture the signal-graph topology coupling within each node's neighborhood. These results highlight the importance of accounting for the graph structure when classifying the different observed graph signals.

Moreover, when comparing the graph-adaptive localized activation functions with their localized alternatives, we can observe that the order K of applying the shift operator does not have a significant impact on the performance. These results are shown in in Figures B.5a and B.5b. When the value of K increases, the performance stagnates or even decreases. We attribute the latter to the high connectivity within each community, which could result in further away neighborhoods becoming redundant.

When comparing the maximum and median localized operators, we observe the performance for the maximum activation functions always decreases when K grows larger than one, compared with $K = 1$. This could be explained by the fact that the maximum function is more sensitive than the median to the possible redundancies induced by larger neighborhoods. These results are also in line with the findings in [12].



(a)



(b)

Figure 5.4: Performances of the max activation functions [in (a)] and median activation functions [in (b)] for the centralized source localization problem. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

While preserving the performance of localized activation functions, the proposed graph-adaptive activation functions also bring computational advantages. First, they can be run distributively, which increases scalability. Additionally, as shown in Figure 5.5, when K grows larger, their computation is faster than in the case of the localized activation functions. This is because there is no necessity to compute the K-hop neighborhood of each node every time the activation function is applied, as in the case of the localized activation functions [12]. These findings are also supported by the results in the additional experiments, shown in Appendix B.

To summarize, the results for the centralized source localization scenario show that the proposed graph-adaptive activation functions preserve the benefits of the localized activation functions [12] and significantly outperform pointwise approaches, while bringing computational advantages. Next, we will take full advantage of their implementation and employ them in a distributed scenario.

Distributed Source Localization

In this section, we present the results for the distributed source localization experiments. Table 5.2 shows the classification accuracy for different numbers of features, $F \in \{2,4,8\}$. For the graph-adaptive nonlinearities, we carried out the experiments with resolutions $K = 1$ and $K = 2$. Figure 5.6 shows the results for the best performing number of features F for each of the nonlinearities. We observe both the localized nonlinearities and the proposed graph-adaptive nonlinearities significantly outperform ReLU, with a difference in classification accuracy of at least 8%. As in the centralized source localization scenario, the lower test accuracy of ReLU and much higher standard deviation (as seen in Table 5.2), results from the instances when the ReLU GCNN fails to learn, as illustrated

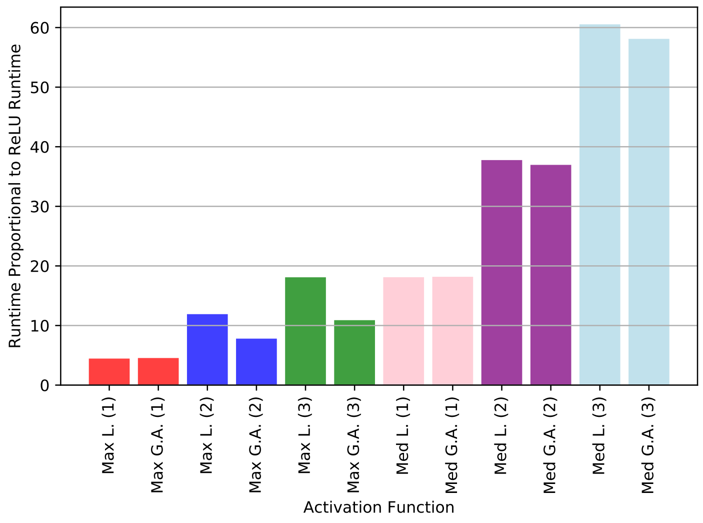


Figure 5.5: Bar plots illustrating the runtime of the max and median graph-adaptive activation functions compared to the max and median localized activation functions for the centralized source localization problem. The runtimes are raported to ReLU runtime. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

in Figure 5.6. This could be explained by the fact that, compared to the other nonlinearities, ReLU is applied pointwise, failing to account for the underlying graph topology. This result highlights the benefits of accounting for the graph topology during classification. The graph-adaptive max and median activation functions outperform their localized versions for $K = 2$, confirming the advantage of accounting for further away signal-graph topology coupling. The max nonlinearities achieve higher accuracy than medians in both the localized and graph-adaptive localized nonlinearities. This result could be caused by the fact that the median will overall smooth the signal, hence undermining some local variations important for classification. Additionally, this could also explain the lower performance of the graph-adaptive kernel nonlinearities compared to the localized nonlinearities, which may be affected by the possible redundancies in the extra information coming from neighbors. We also observe that the best performance is achieved across all activation functions for $F = 8$ features. This result can be explained by the increased expressivity power of the GCNN when the number of features increases.

5.3. Distributed Finite-Time Consensus

5.3.1. Problem Definition

Distributed finite-time consensus aims to achieve consensus among all nodes in finite-time, by accessing only local information at each node. We formulated distributed finite-time consensus as a learning problem on graphs. We employed a distributed GCNN with different activation functions [cf.Figure 2.6] to learn the consensus function in a data-driven fashion. We compared the GCNNs performance with an FIR filter. Our goal was

Table 5.2: Distributed Source Localization Test Accuracy. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified. In boldface we show the best performance for each family of nonlinearities.

Nonlinearity/ F	2	4	8
ReLU	46.7(±11.5)%	44.6(±14.7)%	**47.1**(±**13.9**)%
Max L.	65.4(±7.7)%	68.5(±7.3)%	**71.5**(±**6.2**)%
Max G.A. (1)	65.1(±8.0)%	68.2(±6.5)%	71.2(±5.7)%
Max G.A. (2)	66.9(±8.3)%	69.2(±6.5)%	**72.2**(±**5.8**)%
Median L.	62.2(±6.2)%	65.3(±7.5)%	**69.6**(±**7.0**)%
Median G.A. (1)	63.5(±7.2)%	66.0(±7.2)%	68.9(±7.4)%
Median G.A. (2)	66.4(±7.6)%	66.7(±7.2)%	**71.0**(±**6.8**)%
Kernel G.A. (1)	57.9(±4.5)%	59.1(±6.4)%	**60.4**(±**6.5**)%
Kernel G.A. (2)	55.9(±5.1)%	58.4(±6.4)%	58.6(±7.0)%

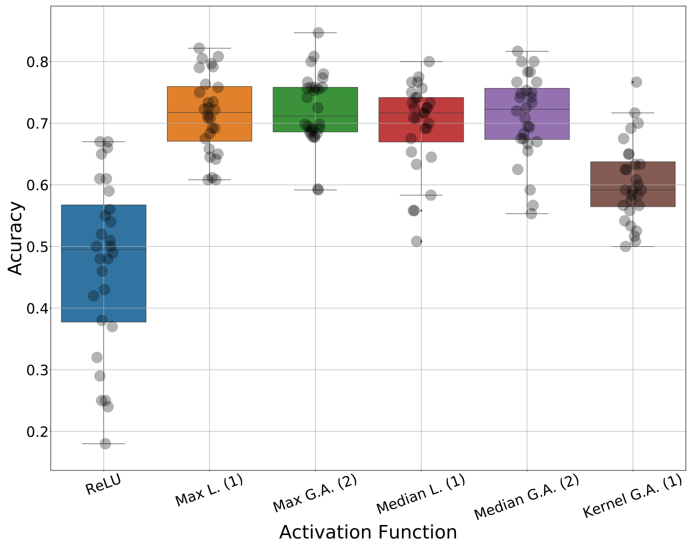


Figure 5.6: Boxplots illustrating the test accuracy of the different activation functions for the distributed source localization problem. Only the results for the best performing number of features F is shown. For our proposed graph-adaptive activation functions only the best performing order is shown. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

to highlight the benefits and limitations of the different activation functions and provide preliminary insights on the GCNN behavior when employed for distributed consensus. Moreover, we considered the transference of the GCNN to unseen graphs to be essential for distributed consensus since, in practical scenarios, communication links are prone to perturbations. Transference properties of the graph filters and the GCNN [cf. (2.11)] are recently linked with their ability to be robust to perturbations [13, 59]. Therefore, we also focused on investigating this property for consensus. Specifically, the research questions we aimed to answer are:

RQ.1 What is the impact of the activation function and filter order on the GCNN performance?

RQ.2 What is the impact of the graph connectivity when learning the GCNN consensus function?

RQ.3 What role do the activation functions play when deploying the GCNN on different graphs?

5.3.2. EXPERIMENTAL SETUP

We considered an undirected stochastic block model (SBM) graph of $N = 100$ nodes divided into $R = 5$ communities with intra- and inter-community probabilities $p = 0.8$ and $q = 0.1$, respectively. The graph signals were generated from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. We generated 2500 samples and split them into 80%, 10%, 10% training, validation and test sets, respectively. We averaged the performance across 10 different graph realizations and 10 different data splits for each graph. The GSO was the normalized adjacency matrix $\mathbf{S} = \mathbf{A}/\lambda_{\max}(\mathbf{A})$, where $\lambda_{\max}(\mathbf{A})$ is the maximum eigenvalue of the adjacency matrix[1]. The considered architecture was a two layer GCNN with $F = 32$ features per layer followed by a per-node fully connected layer. To train the parameters, we employed the ADAM optimiser with learning rate 0.001 and forgetting factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for 400 epochs and batch size of 100 samples. For this task, we also employed the pointwise kernel activation function in [11], for which we considered the same parameters as in the original paper. We considered RMSE as the evaluation metric.

5.3.3. NUMERICAL RESULTS

NON-LINEARITY AND FILTER ORDER

We analyzed the different activation functions considering convolution filters of orders $K \in \{20, 25, 30, 35, 40\}$. Since for consensus we want the filters to approximate a strongly low-pass transfer function, low filter orders (e.g., $K = 1, \ldots, 5$ as used for classification) significantly affect the performance. In Figure 5.7, we show the RMSE as a function of the filter order for the different nonlinearities. All GCNNs achieve a significantly lower RMSE compared with the FIR graph filter, with the GCNN employing the graph-adaptive kernel activation function performing the best across all filter orders. For the lowest order $K = 20$, ReLU yields a worse RMSE than the localized and graph-adaptive nonlinearities. Once the filter order increases, and thus the degrees of freedom, adding a parametric nonlinearity seems to be less beneficial because the network has enough degrees of freedom in the filter to model the consensus function. These observations suggest that parametric activation functions should be preferred when a GCNN architecture with non-parametric nonlinearity has low discriminatory power or when the communication cost is limited.

[1] We also experimented with the Laplacian as GSO but its performance was consistently worse compared with the normalized adjacency matrix.

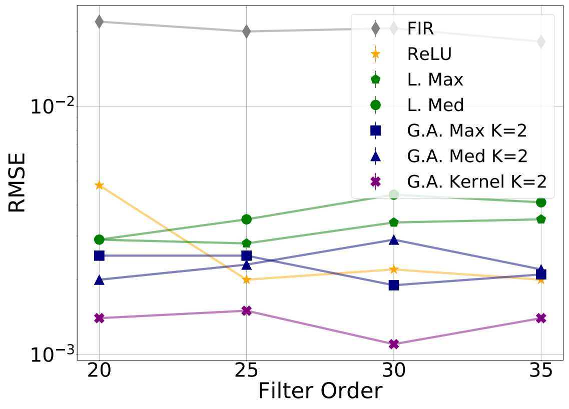


Figure 5.7: Root mean square error (RMSE) of the GCNNs and FIR graph filters for distributed finite-time consensus as a function of filter order. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. K denotes the filter order.

Graph Connectivity

To analyze the impact of the graph connectivity when learning the consensus function, we evaluated the inter-community edge formation probability in the interval $q \in [0.05, 0.25]$. For this experiment, we only considered the best performing architecture from the previous experiment for each of the different families of activation functions. That is: i) the max localized activation with a convolutional filter order of 25; ii) the max graph-adaptive activation function with filter order $K = 2$ and convolutional filter order of 30; iii) the kernel graph-adaptive activation function with filter order $K = 2$ and convolutional filter order of 30. Figure 5.8 shows the results in comparison with the FIR graph filter and the GCNN with the ReLU nonlinearity for $K = 25$. For all methods, we observe a lower RMSE when the communities are better connected. This finding is intuitively satisfying for distributed consensus, as the better connected the communities, the easier nodes get information from further away neighbors. We also see a more stable trend in the case of the graph-adaptive and localized nonlinearities when compared to ReLU. This is a sensible finding, as the graph-adaptive and localized activation functions account for the graph structure. Thus, the changes in the graph connectivity reflect more clearly in their performances. It is also important to note that the graph-adaptive nonlinearities outperform the other approaches across all the different graph connectivities. This supports the importance of accounting for further away neighborhoods in distributed scenarios.

Robustness

In this last experiment, we analyze the robustness of the different models to link losses by removing edges with different probabilities, following the random edge sampling model in [55]. For each method, we considered the best performing order. From the trained graph $\mathcal{G}$, we randomly removed edges with probabilities in the interval $[0.025, 0.15]$. Fig-

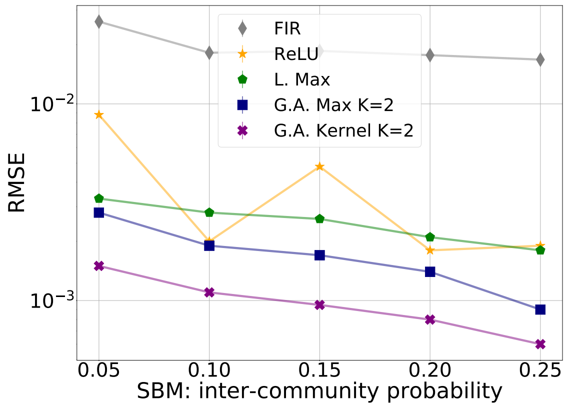


Figure 5.8: Root mean square error (RMSE) of the GCNNs and FIR graph filters for distributed finite-time consensus as a function of graph connectivity. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. K denotes the filter order.

ure 5.9 illustrates the performance averaged over 10 additional realizations. All GCNN models outperform the FIR. We observe, however, the FIR seems to be less sensitive to the link losses. The graph-adaptive nonlinearities seem to handle the stochasticity better than the localized ones, constantly retaining a performance that is equally good or even better than the localized nonlinearities. The graph-adaptive kernel nonlinearity seems to be the most sensitive as its performance reaches those of the other graph-adaptive alternatives, while initially having a significantly lower RMSE.

5.4. Distributed Regression

5.4.1. Data Preprocessing

We used the Molene dataset [60], which is an open access dataset published by the French national meteorological service. The dataset contains hourly temperature measurements across several stations recorded in January 2014 in the area of Brest (France) for $T = 744$ hours. The measurements are available for 55 stations, however, the geographical coordinates are only provided for 46 stations. Since we built the underlying graph based on the stations' location, as we will discuss next, we removed from the data the measurements for the six stations with an unidentified location. For the remaining 46 stations, we checked the number of hours for which the measurements have been taken. We found out that for 10 stations the temperatures have not been measured for all the 744 hours (i.e., the total occurrences in the dataset associated with each of those 10 station identifiers was less than 744). To ensure a balanced dataset, we removed the occurrences of these 10 stations from the data. Finally, four of the remaining stations did not have the actual value of the measurement recorded. That is, those four stations had 744 entries associated with them in the data, but the actual field containing the measured temperature was empty. We further removed these stations from the data. After

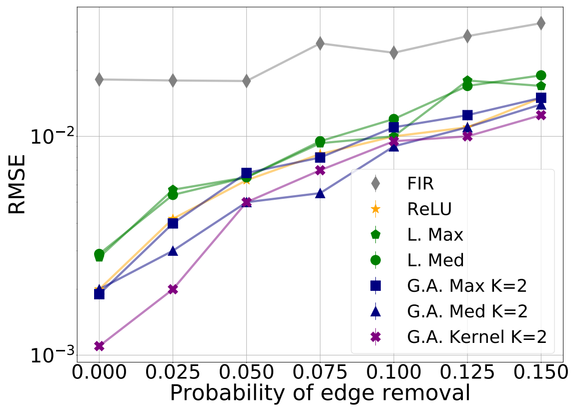


Figure 5.9: Robustness of the GCNNs and FIR graph filters for distributed finite-time consensus as a function of link-loss probabilities. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. K denotes the filter order.

these preprocessing steps, the final dataset consisted of 32 unique stations, each having temperature measurements across 744 hours. The in-sample mean was subtracted from the raw data for each station. That is, the mean per station from the data constituting the train and validation sets, also referred to as ’in-sample’ data. The names and locations of the stations considered are included in Appendix C.

5.4.2. PROBLEM DEFINITION

We performed distributed regression using the Molene dataset. More specifically, we added noise to the temperature measurements and our goal was to recover the clean measurements distributively at each station in the network. Formally, our research questions were:

RQ.1 What is the impact of the activation function and filter order on the GCNN performance when learning the denoising function?

RQ.2 How do different noise levels impact the GCNN performance when employing different activation functions?

5.4.3. EXPERIMENTAL SETUP

Using the node (station) coordinates, we generated a weighted geometric graph as in [61], using a nearest neighbor (NN) approach. More specifically, the graph followed a 10-NN construction and the weight w_{ij} of each edge (i, j) was calculated from Gaussian kernel as

$$w_{ij} = \exp(-d(i, j)/\overline{d})$$

where $d(i, j)$ represents the Euclidean distance between the locations of stations i and j and $\overline{d}$ is the average Euclidean distance between all locations. The graph was built by employing the GSPBox framework proposed in [62].

We considered as graph signals the measurements taken at different timestamps $t \in T$. Thus, our data set comprised 744 graph signals. On top of the original signals, we added zero-mean noise with a signal-to-noise ratio (SNR) of SNR = 3 dB for the first set of experiments, while for the second set we varied the SNR in the interval $[-5\text{ dB}, 5\text{ dB}]$. These noisy signals were split into 80%, 10%, 10% training, validation, and test sets, respectively. Our goal was to train a GCNN for removing the noise distributively. We employed a distributed GCNN [cf. Figure 2.6] with one layer and a varying number of features $F \in \{1,2,4\}$ and orders $K \in \{1,2,4,8\}$. We trained for 500 epochs with a batch size of 100 samples. We employed RMSE as the evaluation metric and averaged the final results across 20 different splits of the dataset.

5.4.4. NUMERICAL RESULTS

For both experiments, in the case of the graph-adaptive activation functions we present the results for the best performing filter order $K = 2$. This highlights the benefit of accounting for further away neighborhoods in tasks requiring distributed computations.

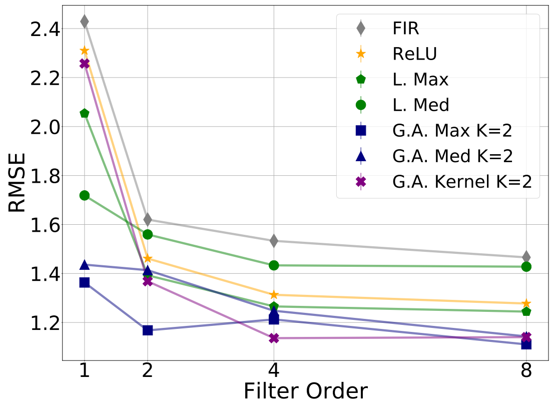


Figure 5.10: Root mean square error (RMSE) of the GCNNs and FIR graph filters for distributed regression as a function of filter order. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. K denotes the filter order.

NON-LINEARITY AND FILTER ORDER

Figure 5.10 shows the RSME as a function of the filter order for the different nonlinearities. Across all GCNNs, the best performance was achieved for the highest number of features, $F = 4$, so we only report these results for this setup. In the other setups, the performances were comparable. Those additional results can be found in Appendix D. All GCNNs perform better than the FIR, but the difference is more significant for the lowest filter order $K = 1$. This is especially the case for the graph-adaptive localized nonlinearities, which significantly outperform other nonlinearities when a filter order $K = 1$

is considered. This finding suggests the applicability of graph-adaptive nonlinearities in situations where the communication resources are limited. To further address this hypothesis, we experimented with different levels of noise added to the data.

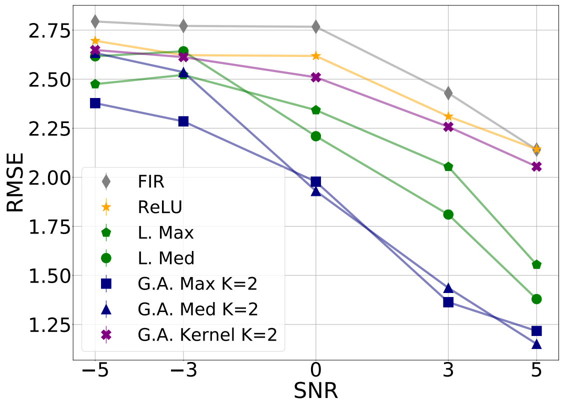


Figure 5.11: Robustness of the GCNNs and FIR graph filters for distributed regression as a function of the SNR. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. K denotes the filter order.

NOISE LEVEL

For each method, we considered the setup with the lowest filter order $K = 1$, thus, simulating a scenario when the communication resources are limited. The results in Figure 5.11 show the graph-adaptive and localized nonlinearities outperform or achieve comparable results to ReLU. The general trend shows an increase in performance when the SNR becomes larger, with a more significant increase for the graph-adaptive localized nonlinearities. The performance of the graph-adaptive kernel nonlinearity suffers in this scenario, as it requires higher filter orders compared to the rest. We suggest using higher orders in the latter case to fully exploit the kernel power.

5.5. CONCLUSION

In this chapter, we presented the numerical experiments carried out in this thesis and discussed the main findings.

In Section 5.2, we experimented with the source localization problem in both centralized and distributed scenarios. In the centralized setting, we saw our proposed graph-adaptive activation functions significantly outperform pointwise nonlinearities and preserve the benefits of localized activation functions, while bringing computational advantages. In the distributed scenario, we identified the max and median graph-adaptive activation functions outperform their localized counterparts, by being able to account for higher resolution neighborhoods.

In Section 5.3, we proposed a data-driven framework for addressing finite-time consensus with GCNNs. We exploited the link between consensus, graph convolutional filters, and GCNNs to propose a method that is readily distributable if the activation functions are properly chosen. Our results suggest: i) parametric activation functions should be employed when the distributed graph filters embedded into a non-parametric nonlinearity have limited discriminatory power – the latter is often linked to communication complexity (i.e., filter order); ii) better connected graphs facilitate learning the consensus function – our rationale is that the improved performance is because each node gets easier the information from all other nodes; iii) GCNNs generalize better to unseen graphs compared with FIR graph filters. These observations show the potential of GCNNs for finite-time consensus and demonstrate that GCNNs overcome by a margin FIR graph filters without requiring all nodes to know the entire graph structure.

In Section 5.4, we experimented with distributed regression. Our results highlight the benefits of graph-adaptive activation functions in distributed regression tasks, especially when the communication resources are restricted. We saw more significant performance increases in the case of the graph-adaptive nonlinearities when the noise levels decreased. Finally, we concluded that kernel graph-adaptive activation functions require higher filter orders than the rest. We suggested employing it in scenarios when this requirement can be met to take full advantage of the kernel power.

Overall, these numerical experiments support the importance of incorporating the signal-graph topology coupling into all the GCNN components. These findings also corroborate the benefits of accounting for this coupling within neighborhoods beyond one-hop when dealing with distributed processing tasks. Moreover, the results demonstrate the applicability of the proposed graph-adaptive activation functions in scenarios when the communication resources are limited. However, we have also seen their limitation in scenarios when the GCNN with a non-parametric nonlinearity, such as ReLU, has a high discriminatory power. While further analysis can shed more light on the performance of the proposed graph-adaptive nonlinearities when dealing with large-scale networks, these results, nevertheless, show the potential and benefits of exploiting the graph also in the nonlinear GCNN component.

6 Conclusion

In this chapter, we conclude the research carried out in this thesis and provide possible future work directions. Section 6.1 provides a summary of our research, while in Section 6.2 we answer the research questions posed in Chapter 1. Finally, in Section 6.3, we address possible future work directions that arise from the research presented in this thesis.

6.1. THESIS SUMMARY

In this thesis, we proposed a new family of *graph-adaptive* activation functions for GNNs that capture the graph topology while being *distributable*.

In Chapter 1, we motivated our research, introduced the research questions, and set the context for the subsequent chapters. In Chapter 2, we introduced the background material necessary for the research carried out in this thesis. We started with basic concepts from graph signal processing, followed by graph filters and their role in graph convolutions and GCNNs. Finally, we discussed the permutation equivariance property of GNCCs and defined different activation functions employed within the GCNNs. In Chapter 3, we provided an overview of the relevant literature for our research, covering GCNNs and their properties, activation functions, as well as the field of distributed signal processing over graphs.

Chapter 4 discussed our main contribution, the graph-adaptive activation functions, implemented by means of nonlinear operators or kernel functions. These activation functions incorporate the signal-graph topology coupling into all the GCNN components by combining nonlinearized features from neighboring nodes with a set of trainable parameters. These parameters adapt the information coming from neighborhoods of different resolutions to the task at hand, hence aiding learning. We also showed the proposed graph-adaptive activation functions preserve permutation equivariance.

In Chapter 5, we compared the graph-adaptive nonlinearities with GCNNs employing localized and pointwise nonlinearities in three different problems, based on both synthetic and real-world data: distributed classification in the context of source localization, distributed finite-time consensus, and distributed regression for signal denoising. Our proposed nonlinearities showed a comparable or even better performance than pointwise and other state-of-the-art localized nonlinearities.

6.2. ANSWER TO THE POSED RESEARCH QUESTIONS

In this section, we address the research questions posed in Chapter 1 and provide answers to them based on the research carried out in this thesis.

(RQ.1) *How can we embed the data-graph topology coupling in the nonlinear component of Graph Neural Networks?*

(RQ.2) *How can we develop activation functions that besides capturing the data-graph topology coupling are also distributable?*

To address these two research questions, we introduced in Chapter 4 the graph-adaptive localized activation functions. More specifically, we defined two types of activation functions: *graph-adaptive localized activation functions*, which are based on arbitrary nonlinear operators acting on the one-hop neighborhood of a node (Section 4.1) and *graph-adaptive kernel activation functions* (Section 4.2), which employ an arbitrary kernel function operating within the one-hop neighborhood of a node. As we discussed in Chapter 4, these proposed activation functions embed the data-graph topology coupling by shifting the nodal features obtained from graph filtering before local-nonlinearization in a form akin to graph convolutions. These nonlinear features are subsequently combined with a set of trainable parameters to weigh the information accordingly at different neighborhood resolutions. The resolution radius is a design parameter and allows adapting the GCNN nonlinear component to the task at hand. Moreover, addressing the second research question, we showed that even though the resolution —given by the shift order— can be arbitrarily large, the graph-adaptive activation functions operate only in the one-hop neighborhood of each node. Thus, they operate only on locally available information at each node and, therefore, lead to a localized and distributed implementation.

(RQ.3) *How can we employ our proposed graph-adaptive activation functions to address distributed processing tasks?*

To address this research question, we experimented in Chapter 5 with three different problems using both real-world and synthetic data. Concretely, we applied a GCNN with our proposed graph-adaptive activation functions for distributed classification in the context of source localization, distributed finite-time consensus, and distributed regression for signal denoising. We compared the performance with GCNNs employing pointwise and state-of-the-art localized nonlinearities. For source localization, our results highlighted the benefits of accounting for the graph topology during classification. Moreover, in the distributed scenario, the graph-adaptive max and median activation functions outperformed their localized versions, confirming the advantage of accounting for the signal-graph topology coupling within higher resolution neighborhoods. For the second numerical experiment, we proposed a data-driven framework for addressing finite-time consensus with GCNNs. We exploited the link between consensus, graph convolutional filters, and GCNNs to propose a method that is readily distributable. Our results demonstrated the potential of GCNNs for finite-time consensus and showed that GCNNs overcome by a margin FIR graph filters without requiring all nodes to know the entire graph structure. Furthermore, when experimenting with distributed regression in Section 5.4, our findings suggested the applicability of the proposed graph-adaptive localized activation functions in situations where the communication resources are limited. We also identified a better generalization ability in the case of graph-adaptive localized nonlinearities. Interestingly, the performance of the graph-adaptive kernel nonlinearity, contrary to the localized ones, suffered in the limited communication scenario, as it required higher filter orders than the rest. Thus, we suggested using higher orders in the latter case to fully exploit the power of the kernel.

6.3. FUTURE WORK

To conclude the research carried out in this thesis, we discuss possible future research directions. Concretely, future work can be carried out in three different directions: i) characterizing the stability of the proposed activation functions to perturbations in the input and topology; ii) performing learning distributively; iii) carrying out further analysis and identifying new interesting applications to further evaluate our proposed graph-adaptive activation functions.

TOPOLOGY PERTURBATIONS

The first possible future direction is to analyze the stability of our proposed graph-adaptive activation functions to input perturbations. This could be achieved through theoretical analysis or through extensive numerical experiments. Concretely, after training a GCNN model employing graph-adaptive activation functions, we would test it on data that has has been altered by the addition of different levels of noise. Thus, we would understand the stability and behavior of our approach when applied to data that has been modified since training. Moreover, we could analyze the stability of our proposed nonlinearities to topology perturbations. This scenario is similar to the former, but instead of modifying the data, we would modify the network topology. We briefly experimented with this scenario for the finite-time consensus problem in Section 5.3, but it would be, nevertheless, interesting to more thoroughly analyze this in a real-world dataset.

DISTRIBUTIVE LEARNING

A second interesting future work direction is performing the learning distributively to address the constantly increasing size of real-world networks. Since graph filters employed in graph convolutions, as well as our proposed graph-adaptive activation functions are distributive, we can deploy a fully graph-adaptive and distributive GCNN. Thus, instead of performing the learning by centrally storing the network, which might be infeasible when dealing with large real-world datasets, we can distribute parts of the network (such as, each node and its one-hop neighbors) over a large number of clients. Each client can independently compute an update to the model based on the nodes that it stores. Subsequently, each independent client's updates will be propagated to a central server that will aggregate them into a global model update. This technique is already employed in the literature under the name of federated learning [63–65].

EXPERIMENTAL FUTURE WORK

For the distributed finite-time consensus problem, additional future research directions can be identified. First, theoretical research can be carried out to investigate the limits of GCNN for finite-time consensus and link them with the graph spectrum. Second, it is worth investigating an asynchronous implementation since the latter has often shown superior performance compared with the synchronous one.

An interesting future application for the proposed graph-adaptive activation functions is the field of robot swarms. The work in [46] already focuses on learning decentralized controllers for the coordination of robot swarms employing GNNs. The authors envision the use of large-scale drone swarms in applications such as environmental surveillance or mapping. These are scenarios in which the communication resource are limited. The work in [46] points out the importance of incorporating information incoming from further away neighbors in the swarm, despite the local communication constraints. This scenario is ideal for the applicability of the proposed graph-adaptive activation function and our findings already highlight their benefits in situations with scarce communication resources.

REFERENCES

[1] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[2] Fernando Gama, Elvin Isufi, Geert Leus, and Alejandro Ribeiro. Graphs, convolutions, and neural networks. *arXiv preprint arXiv:2003.03777*, 2020.

[3] Aliaksei Sandryhaila, Soummya Kar, and José MF Moura. Finite-time distributed consensus through graph filters. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 1080–1084. IEEE, 2014.

[4] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

[5] Ali Jadbabaie, Jie Lin, and A Stephen Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on automatic control*, 48(6):988–1001, 2003.

[6] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[7] David I Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst.

[8] David I Shuman, Pierre Vandergheynst, Daniel Kressner, and Pascal Frossard. Distributed signal processing via chebyshev polynomial approximation. *IEEE Transactions on Signal and Information Processing over Networks*, 4(4):736–751, 2018.

[9] Elvin Isufi, Andreas Loukas, Andrea Simonetto, and Geert Leus. Autoregressive moving average graph filtering. *IEEE Transactions on Signal Processing*, 65(2):274–288, 2016.

[10] Santiago Segarra, Antonio G Marques, and Alejandro Ribeiro. Optimal graph-filter design and applications to distributed linear network operators. *IEEE Transactions on Signal Processing*, 65(15):4117–4131, 2017.

[11] Simone Scardapane, Steven Van Vaerenbergh, Danilo Comminiello, and Aurelio Uncini. Improving graph convolutional networks with non-parametric activation functions. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 872–876. IEEE, 2018.

[12] Luana Ruiz, Fernando Gama, Antonio García Marques, and Alejandro Ribeiro. Invariance-preserving localized activation functions for graph neural networks. *IEEE Transactions on Signal Processing*, 68:127–141, 2019.

[13] Fernando Gama, Alejandro Ribeiro, and Joan Bruna. Stability of graph scattering transforms. In *Advances in Neural Information Processing Systems*, pages 8036–8046, 2019.

[14] Morris H DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, 69(345):118–121, 1974.

[15] Reza Olfati-Saber, J Alex Fax, and Richard M Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, 2007.

[16] Lin Xiao and Stephen Boyd. Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1):65–78, 2004.

[17] Shreyas Sundaram and Christoforos N Hadjicostis. Distributed function calculation and consensus using linear iterative strategies. *IEEE journal on selected areas in communications*, 26(4):650–660, 2008.

[18] Sérgio Pequito, Stephen Kruzick, Soummya Kar, José MF Moura, and Pedro A Aguiar. Optimal design of distributed sensor networks for field reconstruction. In *21st European Signal Processing Conference (EUSIPCO 2013)*, pages 1–5. IEEE, 2013.

[19] IEEE International Workshop on MACHINE LEARNING FOR SIGNAL PROCESSING (MLSP). https://ieeemlsp.cc/, 2020. Accessed 10.06.2020.

[20] EUSIPCO 2020. https://eusipco2020.org/, 2020. Accessed 10.06.2020.

[21] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José MF Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

[22] Mario Coutino, Elvin Isufi, and Geert Leus. Advances in distributed graph filtering. *IEEE Transactions on Signal Processing*, 67(9):2320–2333, 2019.

[23] Mario Coutino, Elvin Isufi, Takanori Maehara, and Geert Leus. On the limits of finite-time distributed consensus through successive local linear operations. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pages 993–997. IEEE, 2018.

[24] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[25] Yves Chauvin and David E Rumelhart. *Backpropagation: theory, architectures, and applications*. Psychology press, 2013.

[26] Simone Scardapane, Steven Van Vaerenbergh, Simone Totaro, and Aurelio Uncini. Kafnets: kernel-based non-parametric activation functions for neural networks. *arXiv preprint arXiv:1707.04035*, 2017.

[27] Franco Scarselli, Sweah Liang Yong, Marco Gori, Markus Hagenbuchner, Ah Chung Tsoi, and Marco Maggini. Graph neural networks for ranking web pages. In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 666–672. IEEE, 2005.

[28] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[30] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and deep locally connected networks on graphs. In *ICLR*, pages 1–14, Banff, AB, 14-16 April 2014. Assoc. Comput. Linguistics.

[31] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *30th NeurIPS*, pages 3844–3858, Barcelona, Spain, 5-10 December 2016. Neural Inf. Process. Syst. Foundation.

[32] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[33] Fernando Gama, Antonio G. Marques, Geert Leus, and Alejandro Ribeiro. Convolutional neural network architectures for signals supported on graphs. *IEEE Transactions on Signal Processing*, 67(4):1034–1049, February 2019.

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

[35] Filippo Maria Bianchi, Daniele Grattarola, Cesare Alippi, and Lorenzo Livi. Graph neural networks with convolutional ARMA filters. *arXiv:1901.01343 [cs.LG]*, 2019.

[36] Elvin Isufi, Fernando Gama, and Alejandro Ribeiro. EdgeNets: Edge varying graph neural networks. *arXiv:2001.07620v1 [cs.LG]*, 21 January 2020.

[37] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018.

[38] Fernando Gama, Geert Leus, Antonio G Marques, and Alejandro Ribeiro. Convolutional neural networks via node-varying graph filters. In *2018 IEEE Data Science Workshop (DSW)*, pages 1–5. IEEE, 2018.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[40] Dongmian Zou and Gilad Lerman. Graph convolutional neural networks via scattering. *Applied and Computational Harmonic Analysis*, 2019.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[42] Pravin Chandra and Yogesh Singh. An activation function adapting training algorithm for sigmoidal feedforward networks. *Neurocomputing*, 61:429–437, 2004.

[43] Babak Zamanlooy and Mitra Mirhassani. Efficient vlsi implementation of neural networks with hyperbolic tangent activation function. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(1):39–48, 2013.

[44] Santiago Segarra, Antonio G Marques, Gonzalo R Arce, and Alejandro Ribeiro. Center-weighted median graph filters. In *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 336–340. IEEE, 2016.

[45] Santiago Segarra, Antonio G Marques, Gonzalo R Arce, and Alejandro Ribeiro. Design of weighted median graph filters. In *2017 IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 1–5. IEEE, 2017.

[46] Ekaterina Tolstaya, Fernando Gama, James Paulos, George Pappas, Vijay Kumar, and Alejandro Ribeiro. Learning decentralized controllers for robot swarms with graph neural networks. In *Conference on Robot Learning*, pages 671–682, 2020.

[47] Raymond Wagner, Véronique Delouille, and Richard Baraniuk. Distributed wavelet de-noising for sensor networks. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 373–379. IEEE, 2006.

[48] Carlos Guestrin, Peter Bodik, Romain Thibaux, Mark Paskin, and Samuel Madden. Distributed regression: an efficient framework for modeling sensor network data. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 1–10, 2004.

[49] Xuesong Shi, Hui Feng, Muyuan Zhai, Tao Yang, and Bo Hu. Infinite impulse response graph filters in wireless sensor networks. *IEEE Signal Processing Letters*, 22(8):1113–1117, 2015.

[50] Santiago Segarra, Antonio G Marques, and Alejandro Ribeiro. Distributed implementation of linear network operators using graph filters. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1406–1413. IEEE, 2015.

[51] Andreas Loukas, Andrea Simonetto, and Geert Leus. Distributed autoregressive moving average graph filters. *IEEE Signal Processing Letters*, 22(11):1931–1935, 2015.

[52] Mario Contino, Elvin Isufi, and Geert Leus. Distributed edge-variant graph filters. In *2017 IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 1–5. IEEE, 2017.

[53] Mario Coutino and Geert Leus. On distributed consensus by a cascade of generalized graph filters. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 46–50. IEEE, 2019.

[54] Zhan Gao, Elvin Isufi, and Alejandro Ribeiro. Stochastic graph neural networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 9080–9084. IEEE, 2020.

[55] Elvin Isufi, Andreas Loukas, Andrea Simonetto, and Geert Leus. Filtering random graph processes over random time-varying graphs. *IEEE Transactions on Signal Processing*, 65(16):4406–4421, 2017.

[56] Zhan Gao, Fernando Gama, and Alejandro Ribeiro. Wide and deep graph neural networks with distributed online learning. *arXiv preprint arXiv:2006.06376*, 2020.

[57] Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2508–2530, 2006.

[58] Thomas Sherson, Richard Heusdens, and W Bastiaan Kleijn. On the distributed method of multipliers for separable convex optimization problems. *IEEE Transactions on Signal and Information Processing over Networks*, 5(3):495–510, 2019.

[59] Ron Levie, Elvin Isufi, and Gitta Kutyniok. On the transferability of spectral graph filters. In *13th International Conference on Sampling Theory and Applications*, pages 1–5, Bordeaux, France, 8-12 July 2019. IEEE.

[60] Météo-France. Données horaires des 55 stations terrestres de la zone Large Molène sur un mois. `https://www.data.gouv.fr/`, 2014. Accessed 10.03.2020.

[61] Elvin Isufi, Andreas Loukas, Nathanael Perraudin, and Geert Leus. Forecasting time series with varma recursions on graphs. *IEEE Transactions on Sig. Proc.*, 67(18):4870–4885, 2019.

[62] Nathanaël Perraudin, Johan Paratte, David Shuman, Lionel Martin, Vassilis Kalofolias, Pierre Vandergheynst, and David K Hammond. Gspbox: A toolbox for signal processing on graphs. *arXiv preprint arXiv:1408.5781*, 2014.

[63] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[64] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[65] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

A

Graph-Adaptive Kernel Activation Functions: Additional Kernel Functions

Possible additional options for employing the kernel function $g(\cdot, \mathcal{N}_i)$ for the proposed graph-adaptive kernel activation functions [cf. (4.9)] are the following:

- **Linear Kernel Function**

$$k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y} + c \tag{A.1}$$

  where c is an optional constant.

- **Polynomial Kernel Function**

$$k(\mathbf{x}, \mathbf{y}) = (\alpha \mathbf{x}^\top \mathbf{y} + c)^d \tag{A.2}$$

  where the adjustable parameters are the slope α, the constant c and the degree of the polynomial d.

- **Exponential Kernel Function**

$$k(\mathbf{x}, \mathbf{y}) = exp(-\frac{||\mathbf{x} - \mathbf{y}||}{2\sigma^2}) \tag{A.3}$$

  where σ is an adjustable parameter.

B

Source Localization: Additional Results

Regarding the different probabilities for generating the SBM graph, the best results are achieved when the communities are highly intra-connected and clearly separated, which is the case of 0.8 intra-community probability and 0.1 inter-community probability. These results were reported in the main body of this thesis. We can observe a decrease in performance both when the connectivity within communities is lower, with a probability of 0.7, and when the communities become less clearly separated and the connectivity between them is higher, with a probability of 0.2. We report next these additional results.

B.1. ADDITIONAL RESULTS GCNN $F = 5$ FEATURES, $K = 5$ FILTER ORDER, AND SBM PROBABILITIES $p = 0.8$ AND $q = 0.2$

The results for the GCNN model consisting of two layers with $F = 5$ features per layer and filters of order $K = 5$ applied on the SBM graph generated using $p = 0.8$ intra-community and $q = 0.2$ inter-community probabilities are reported in Table B.1 and Figures B.1, B.2 and B.3.

Table B.1: Centralized Source Localization Test Accuracy. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

Activation Function	Average Accuracy	Standard Deviation
ReLU	51.6%	9.3
Max L. (1)	59.1%	3.7
Max L. (2)	**59.4**%	3.7
Max L. (3)	58.7%	4.4
Max G.A (1)	58.6%	5.3
Max G.A (2)	**59.5**%	4.4
Max G.A (3)	58.8%	3.7
Median L. (1)	60.1%	4.2
Median L. (2)	**62.1**%	4.2
Median L. (3)	60.5%	3.7
Median G.A (1)	60.1%	3.4
Median G.A (2)	**60.3**%	3.2
Median G.A (3)	60.1%	4.1
Kernel G.A. (1)	56.5%	5.2
Kernel G.A. (2)	**58.0**%	4.4
Kernel G.A. (3)	57.6%	3.9

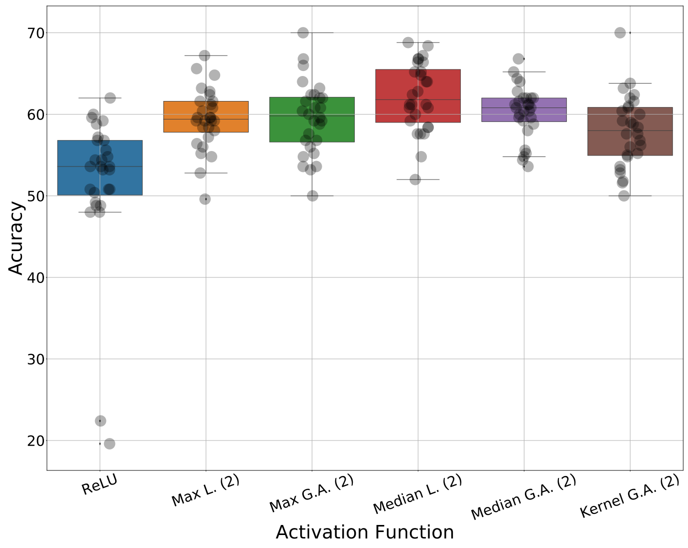


Figure B.1: Boxplots illustrating the performances of the different activation functions for the centralized source localization problem. For our proposed graph-adaptive and for the localized activation functions only the best performing order is shown. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

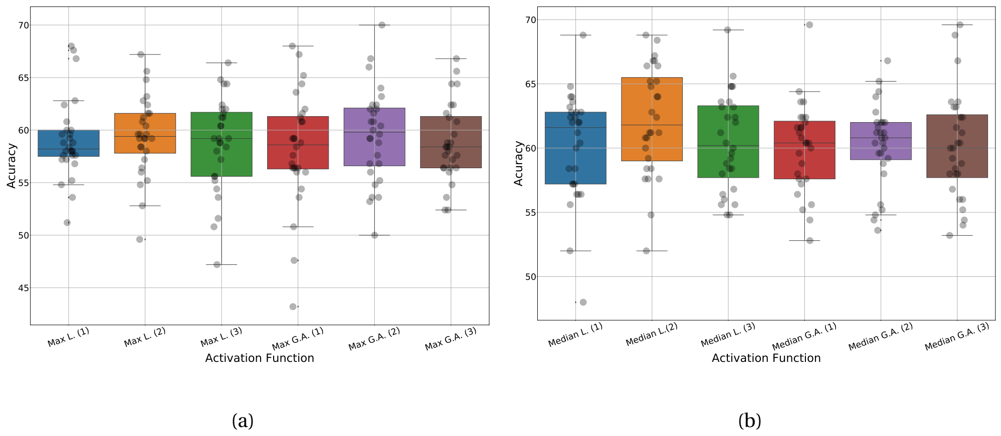


Figure B.2: Performances of the max activation functions [in (a)] and median activation functions [in (b)] for the centralized source localization problem. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

B.2. ADDITIONAL RESULTS GCNN $F = 5$ FEATURES, $K = 5$ FILTER ORDER, AND SBM PROBABILITIES $p = 0.7$ AND $q = 0.1$

The results for the GCNN model consisting of two layers with $F = 5$ features per layer and filter order of $K = 5$ applied on the SBM graph generated using $p = 0.7$ intra-community

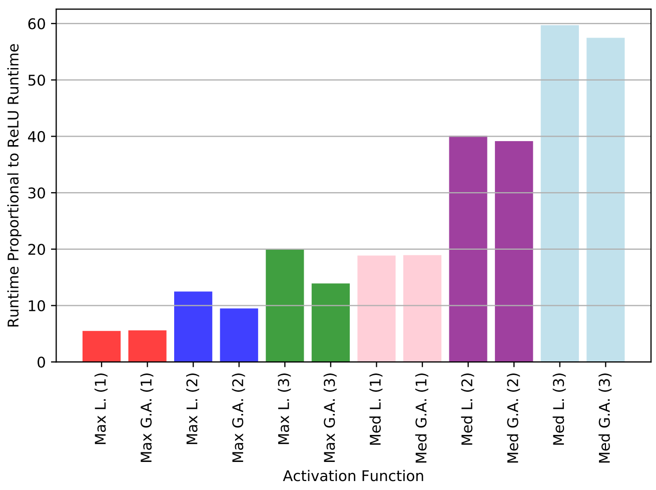


Figure B.3: Bar plots illustrating the runtime of the max and median graph-adaptive activation functions compared to the max and median localized activation functions for the centralized source localization problem. The runtimes are raported to ReLU runtime. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

and $q = 0.1$ inter-community probabilities are reported in Table B.2 and Figures B.4, B.5 and B.6.

Table B.2: Centralized Source Localization Test Accuracy. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

Activation Function	Average Accuracy	Standard Deviation
ReLU	65.5%	9.6
Max L. (1)	**74.5**%	4.5
Max L. (2)	73.7%	4.3
Max L. (3)	73.8%	4.3
Max G.A (1)	**73.6**%	3.8
Max G.A (2)	73.6%	3.1
Max G.A (3)	72.9%	4.1
Median L. (1)	75.5%	3.2
Median L. (2)	**76.0**%	3.1
Median L. (3)	75.9%	3.7
Median G.A (1)	**75.9**%	3.0
Median G.A (2)	75.1%	3.6
Median G.A (3)	75.4%	4.3
Kernel G.A. (1)	69.9%	4.7
Kernel G.A. (2)	**71.6**%	3.4
Kernel G.A. (3)	71.4%	3.5

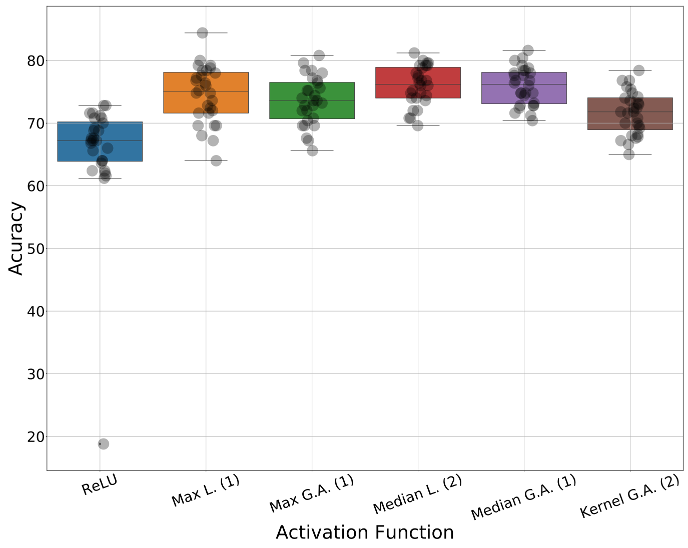
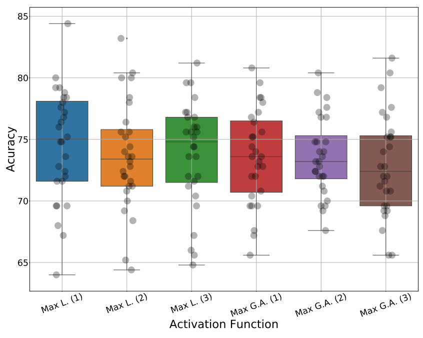
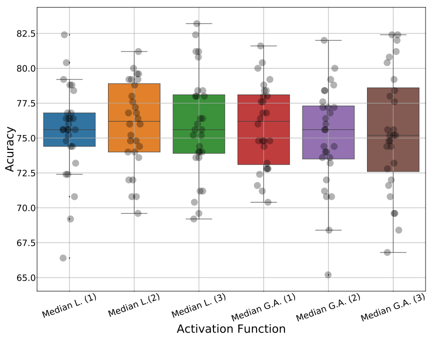


Figure B.4: Boxplots illustrating the test accuracy of the different activation functions for the centralized source localization problem. For our proposed graph-adaptive and for the localized activation functions only the best performing order is shown. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.



(a)



(b)

Figure B.5: Performances of the max activation functions [in (a)] and median activation functions [in (b)] for the centralized source localization problem. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

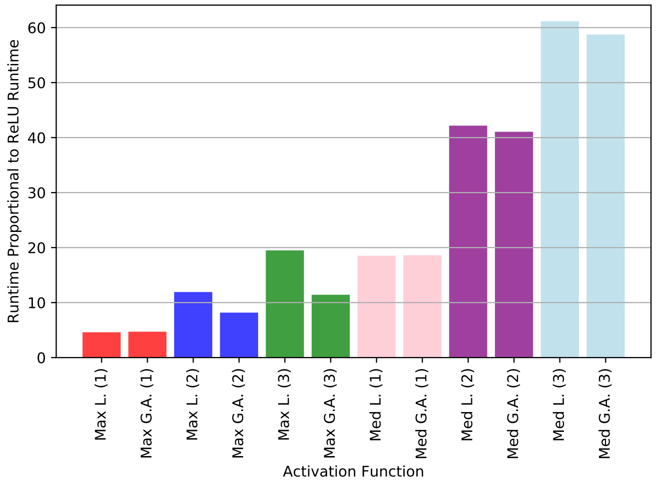


Figure B.6: Bar plots illustrating the runtime of the max and median graph-adaptive activation functions compared to the max and median localized activation functions for the centralized source localization problem. The runtimes are raported to ReLU runtime. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. Between brackets the filter order K is specified.

C

Distributed Regression: Weather Stations Information

Table C.1: Geographical coordinates of the weather stations considered in the distributed regression numerical experiments.

Station Name	Latitude	Longitude
ILE-DE-BREHAT	48°51’18N	3°00’12W
KERPERT	48°24’12N	3°08’48W
LOUARGAT	48°33’06N	3°22’36W
MERDRIGNAC	48°10’54N	2°24’36W
PLOUMANAC’H	48°49’30N	3°28’18W
PLOUGUENAST	48°15’54N	2°44’54W
POMMERIT-JAUDY	48°44’30N	3°15’00W
QUINTENIC	48°31’06N	2°25’12W
ROSTRENEN	48°13’48N	3°18’24W
SAINT-CAST-LE-G	48°38’36N	2°14’48W
ST BRIEUC	48°32’00N	2°51’06W
PTE DE PENMARCH	47°47’48N	4°22’24W
PLEYBER-CHRIST SA	48°30’00N	3°51’12W
PTE DU RAZ	48°02’18N	4°43’54W
ST-SEGAL S A	48°13’36N	4°05’48W
SIBIRIL S A	48°39’36N	4°04’36W
SPEZET	48°10’24N	3°43’42W
DINARD	48°35’18N	2°04’30W
GUERANDE	47°17’30N	2°25’48W
PTE DE CHEMOULIN	47°14’00N	2°17’54W
AURAY	47°39’30N	2°58’12W
BELLE ILE-LE TALUT	47°17’36N	3°13’06W
BIGNAN	47°53’00N	2°43’42W
ILE DE GROIX	47°39’06N	3°30’06W
PLEUCADEUC	47°45’54N	2°23’12W
PLOERMEL	47°57’00N	2°23’48W
PONTIVY	48°03’48N	2°56’42W
LORIENT-LANN BIHOUE	47°45’42N	3°26’06W
SARZEAU SA	47°30’42N	2°47’48W
VANNES-SENE	47°36’12N	2°42’48W
THEIX	47°38’24N	2°37’12W
NOIRMOUTIER EN	47°00’12N	2°15’24W

D
Distributed Regression: Additional Results

D.1. Additional Results for the GCNN with $F = 1$ Feature

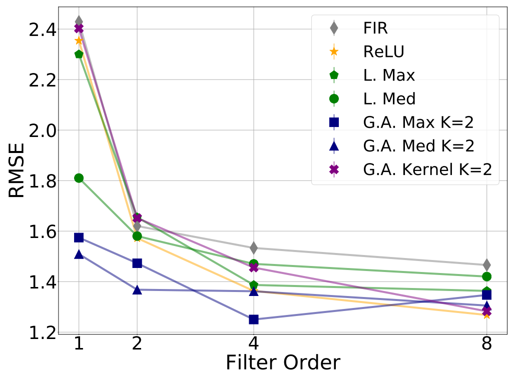


Figure D.1: Robustness of the GCNNs and FIR graph filters for distributed regression as a function of the SNR. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. K denotes the filter order.

D.2. Additional Results for the GCNN with $F = 2$ Features

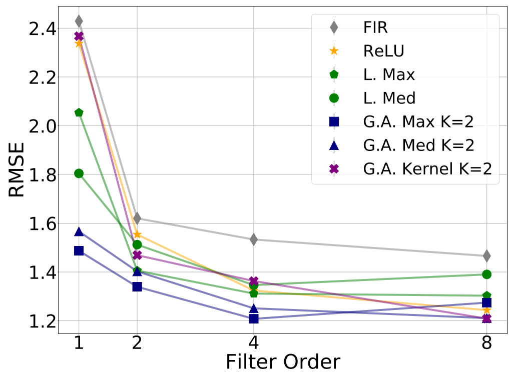


Figure D.2: Robustness of the GCNNs and FIR graph filters for distributed regression as a function of the SNR. L.: localized nonlinearities [cf. (2.24)]; G.A.: graph-adaptive nonlinearities [cf. (4.5) and (4.9)]. K denotes the filter order.

GRAPH-ADAPTIVE ACTIVATION FUNCTIONS FOR GRAPH NEURAL NETWORKS

*Bianca Iancu**, *Luana Ruiz*†, *Alejandro Ribeiro*† *and Elvin Isufi**

* Intelligent Systems Department, Delft University of Technology, Delft, The Netherlands
† Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, USA

ABSTRACT

Activation functions are crucial in graph neural networks (GNNs) as they allow defining a nonlinear family of functions to capture the relationship between the input graph data and their representations. This paper proposes activation functions for GNNs that not only adapt to the graph into the nonlinearity, but are also distributable. To incorporate the feature-topology coupling into all GNN components, nodal features are nonlinearized and combined with a set of trainable parameters in a form akin to graph convolutions. The latter leads to a graph-adaptive trainable nonlinear component of the GNN that can be implemented directly or via kernel transformations, therefore, enriching the class of functions to represent the network data. Whether in the direct or kernel form, we show permutation equivariance is always preserved. We also prove the subclass of graph-adaptive *max* activation functions are Lipschitz stable to input perturbations. Numerical experiments with distributed source localization, finite-time consensus, distributed regression, and recommender systems corroborate our findings and show improved performance compared with pointwise as well as state-of-the-art localized nonlinearities.

Index Terms— Activation functions; graph neural networks; graph signal processing; Lipschitz stability; permutation equivariance.

1. INTRODUCTION

Graph neural networks (GNNs) are parametric architectures suitable for learning a nonlinear mapping for data defined over graphs such as social, sensor, and biological network data [1,2]. By interweaving graph filters with pointwise nonlinearities, GNNs express the function map in a layered form and learn compositions of features that account for the data-topology coupling [3,4]. Another property GNNs inherit from graph filters is the distributed implementation [5–7]. Distributed computation facilitates scalability of computation and endows the system with robustness to failures of the processing unit. The latter is fundamental in applications involving consensus, optimization, and control [8–10].

Building on spectral graph theory, [11] defined graph convolutional neural networks by multiplying feature representations in the Laplacian eigenspace with trainable kernels. Subsequently, [3] used finite impulse response (FIR) graph filters to combine features in the vertex domain by means of a polynomial in the Laplacian matrix. The work in [4] follows the same idea but builds a polynomial filter in any graph representation matrix (e.g., adjacency, Laplacian). Differently from [11], [3,4] are also readily distributable architectures with appropriate choices of graph pooling (i.e., not altering the graph structure; e.g., zero-padding) and with pointwise activation functions. On the other hand, [12] builds a GNN with distributable autoregressive moving average graph filters [6], which capture a broader family of functions at the expense of computation cost. Parallel to these efforts, [13] proposes attention-like mechanisms to adapt the edge weights to the task at hand. More recently, the work in [14] showed all the above architectures are equivalent and fall under the framework of edge varying GNN (EdgeNet). Altogether, these works capture the data-graph coupling only linearly through graph filters, while they ignore the coupling in the nonlinear pointwise component (e.g., ReLU). To improve the representation power of GNNs, [15] proposed localized activation functions that account for the graph topology by operating on node neighborhoods of different resolutions. However, the latter accounts only for the graph and not the data-topology coupling, since it ignores the edge weights and the data propagation between neighbors. Localized activation functions are also not distributable beyond the one-hop neighborhood, hence missing multi-hop information between nodes.

To address these limitations, we put forward a new family of activation functions that adapt to the data-topology coupling in the surrounding of a node. The nodal features obtained from graph filtering are shifted prior to local-nonlinearization in a form akin to graph convolutions. These nonlinear features are subsequently combined with a set of trainable parameters to accordingly weight the information at different neighborhood resolutions. The resolution radius is a design parameter and allows adapting the GNN nonlinear component to the task at hand. Besides being graph-adaptive and distributable, these activation functions preserve two properties of theoretical interest for GNNs, namely permutation equivariance and Lipschitz stability to perturbations [16]. Concretely, our contribution is threefold.

1. We develop a new family of nonlinearities for GNNs that are graph-adaptive to the surrounding of a node and distributable. The first class [Def. 3] nonlinearizes shifted features in the surrounding of a node in their direct form. The second class [Def. 6] transforms the shifted features with graph-adaptive kernels prior to nonlinearization.
2. We prove that: (a) the proposed nonlinearities are permutation equivariant [Prop. 1], i.e. the output of the respective GNN architecture is agnostic to node labeling; (b) the *max* graph-adaptive nonlinearity is Lipschitz stable to input perturbations [Prop. 2].
3. We propose distributed GNN tasks with graph-adaptive nonlinearities for source localization, finite-time consensus, signal denoising, and rating prediction in recommender systems.

2. GRAPH NEURAL NETWORKS

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ of cardinality $|\mathcal{V}| = N$ and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ of cardinality $|\mathcal{E}| = M$. An edge is a tuple $e_{ij} = (i, j)$ connecting nodes i and j. The neighborhood of node i is the set of nodes $\mathcal{N}_i = \{j|(i,j) \in \mathcal{E}\}$ connected to i. Associated to $\mathcal{G}$ is the graph shift operator (GSO) matrix $\mathbf{S} \in \mathbb{R}^{N\times N}$, whose sparsity pattern matches the graph structure. That is, entry (i, j) satisfies $[\mathbf{S}]_{i,j} = s_{i,j} \neq 0$ only if $i = j$ or $(i, j) \in \mathcal{E}$. Commonly used GSOs include the adjacency matrix, the graph Laplacian, and their normalized and translated forms.

On the vertices of $\mathcal{G}$, we define a graph signal $\mathbf{x} \in \mathbb{R}^N$ whose ith component is the value at node i. We consider applications where graph signals are processed in a *distributed* fashion. A typical example is in sensor networks without access to a centralized processing unit and where each sensor communicates only with its neighbor sensors.

Graph convolution. A graph convolution is defined as a graph filter $\mathbf{H}(\mathbf{S})$ that can be written as a polynomial of the GSO $\mathbf{S}$ [7]. For an input signal $\mathbf{x}$ and filter coefficients $\mathbf{h} = [h_0, \dots, h_K]^\top$, the output $\mathbf{y} \in \mathbb{R}^N$ of the graph convolutional filter is computed as

$$\mathbf{y} = \mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x}. \tag{1}$$

Due to the locality of $\mathbf{S}$, graph convolutions can be run distributively. When building the output $\mathbf{y}$, we need to compute the terms $\mathbf{S}\mathbf{x}, \dots, \mathbf{S}^K\mathbf{x}$. Since $\mathbf{S}$ is local, operation $\mathbf{S}\mathbf{x}$ requires one-hop node exchanges and so, by writing $\mathbf{S}^k\mathbf{x} = \mathbf{S}(\mathbf{S}^{k-1}\mathbf{x}) = \mathbf{S}\mathbf{x}^{(k-1)}$, node i can compute signal $\mathbf{x}^{(k)}$ through exchange of previous shifted information $\mathbf{x}^{(k-1)}$ with its neighbors. This recursion allows for distributed communications and computational cost of order $\mathcal{O}(MK)$, while the trainable parameters defining (1) are of order $\mathcal{O}(K)$ [7].

Graph convolutional neural networks (GCNNs). We consider a GCNN of L graph convolutional layers followed by a shared fully connected layer per node. Each convolutional layer comprises a bank of graph filters [cf. (1)] and a nonlinearity. At layer l, the GCNN takes as input F_{l-1} features $\{\mathbf{x}_{l-1}^g\}_{g=1}^{F_{l-1}}$ from layer $(l-1)$ and produces F_l output features $\{\mathbf{x}_l^f\}_{f=1}^{F}$. Each input feature $\mathbf{x}_{l-1}^g$ is processed by a parallel bank of F_l graph filters $\{\mathbf{H}_l^{fg}(\mathbf{S})\}_f$. The filter outputs are aggregated over the input index g to yield the fth convolved feature

$$\mathbf{z}_l^f = \sum_{g=1}^{F_{l-1}} \mathbf{H}_l^{fg}(\mathbf{S})\mathbf{x}_{l-1}^g = \sum_{g=1}^{F_{l-1}} \sum_{k=0}^{K} h_{kl}^{fg} \mathbf{S}^k \mathbf{x}_{l-1}^g, \text{ for } f=1,\ldots,F_l. \tag{2}$$

The convolved feature $\mathbf{z}_l^f$ is subsequently passed through an activation function $\sigma(\cdot)$ to obtain the fth convolutional layer output

$$\mathbf{x}_l^f = \sigma(\mathbf{z}_l^f), \quad \text{for } f=1,\ldots,F. \tag{3}$$

The output features of the last convolutional layer L, $\mathbf{x}_L^1,\ldots,\mathbf{x}_L^{F_L}$, represent the final convolutional features. These features are interpreted as a collection of F_L graph signals, where on node i we have the $F_L \times 1$ feature vector $\boldsymbol{\chi}_{Li} = [x_{Li}^1,\ldots,x_{Li}^{F_L}]^\top$. Each node locally combines the features $\boldsymbol{\chi}_{Li}$ with a one-layer perceptron to obtain the output

$$\tilde{\mathbf{y}} = \mathbf{H}_{\text{FC}}\boldsymbol{\chi}_{Li} \tag{4}$$

where matrix $\mathbf{H}_{\text{FC}} \in \mathbb{R}^{F_o \times F_L}$ maps the F_L convolutional features to the F_o output features (e.g., the number of classes). The parameters in $\mathbf{H}_{\text{FC}}$ are shared among nodes to keep the number of trainable parameters independent of the graph dimensions (i.e. N and M), but only dependent on the filter order and the number of features and layers.

By grouping all learnable parameters into the set $\mathcal{H} = \{\mathbf{h}_l^{fg}; \mathbf{H}_{\text{FC}}\}_{lfg}$, we can consider the GCNN as a map $\boldsymbol{\Phi}(\cdot)$ that takes as input a graph signal $\mathbf{x}$, a GSO $\mathbf{S}$, and a set of parameters $\mathcal{H}$ to produce the output

$$\boldsymbol{\Phi}(\mathbf{x};\mathbf{S};\mathcal{H}) := \tilde{\mathbf{y}}. \tag{5}$$

The output (5) is computed for a training set $\mathcal{T} = \{(\mathbf{x},\mathbf{y})\}$ of $|\mathcal{T}|$ pairs, where $\mathbf{y}$ are the target representations.

Activation functions. The activation function $\sigma(\cdot)$ in (3) can be any of the conventional pointwise activation functions, such as ReLU ($\sigma(x) = \max(0,x)$), or a localized activation function [15]. Differently from the pointwise, localized activation functions consider the features at the neighborhood of each node i in the nonlinear GCNN component [15]. For a graph signal feature $\mathbf{x}$ the localized activation function is based on two local operators, namely:

- *local max operator*, $\max(\mathbf{S},\mathbf{x})$, whose output is a graph signal $\mathbf{z}$ with ith entry being the maximum value of the signal in the neighborhood, i.e., $z_i = [\max(\mathbf{S},\mathbf{x})]_i = \max\big(\{x_j : v_j \in \mathcal{N}_i\}\big)$;
- *local median operator*, $\text{med}(\mathbf{S},\mathbf{x})$, whose output is a graph signal $\mathbf{z}$ with ith entry being the median value of the signal in the neighborhood, i.e., $z_i = [\text{med}(\mathbf{S},\mathbf{x})]_i = \text{med}\big(\{x_j : v_j \in \mathcal{N}_i\}\big)$.

For simplicity, we denote both local operators with the generic local function $f(\mathbf{S},\mathbf{x})$. Then, the localized activation function is defined as

$$\sigma(\mathbf{x}) = \beta\text{ReLU}(\mathbf{x}) + \sum_{k=1}^{K} h_{\sigma k} f(\mathbf{S}^k,\mathbf{x}). \tag{6}$$

where $f(\mathbf{S}^k,\mathbf{x})$ applies the local activation function to the signal values of the k-hop neighbors and parameters β and $\mathbf{h}_\sigma = [h_{\sigma 1},\ldots,h_{\sigma K}]^\top$ are *learned* [15]. A GCNN with localized activation functions can thus be written as the map $\boldsymbol{\Phi}(\mathbf{x};\mathbf{S};\mathcal{H})$ with parameters $\mathcal{H} = \{\mathbf{h}_l^{fg}; \mathbf{h}_{\sigma l}^f; \mathbf{H}_{\text{FC}}\}_{lfg}$.

As it follows from (6), localized activation functions ignore the edge weights and require information from the non-immediate k-hop neighbors, which makes them not distributable. Hence, in distributed settings, the order K in (6) is limited to one. To address this limitation, we propose two new activation functions based on local operators and kernel functions to account for the graph structure and be distributable.

3. GRAPH-ADAPTIVE ACTIVATION FUNCTIONS

In this section, we first define the *graph-adaptive localized activation functions*, which are based on arbitrary nonlinear operators acting on the one-hop neighborhood of a node (Section 3.1). Then, we define the *graph-adaptive kernel activation functions* (Section 3.2). Finally, we prove the proposed nonlinearities are permutation equivariant and stable to input perturbations (Section 3.3).

3.1. Graph-Adaptive Localized Activation Functions

To start, let us first define the basic building block for graph-adaptive activation functions: the *shifted localized operator* (SLO).

Definition 1 (Shifted Localized Operator)**.** Let $\mathcal{G}$ be an N-node graph with shift operator $\mathbf{S}$, $\mathbf{x}$ a signal, and $\mathbf{S}^k\mathbf{x}$ the kth shifted signal. Consider an arbitrary nonlinear localized function $f(\cdot,\mathcal{N}_i) : \mathbb{R}^N \to \mathbb{R}^N$, which at node i computes the local nonlinear operation $[f(\mathbf{x},\mathcal{N}_i)]_i = f(\{x_j\}_{j\in\mathcal{N}_i})$. For this choice of $f(\cdot,\mathcal{N}_i)$, the k-hop shifted localized operator maps input $\mathbf{x}$ to output $\mathbf{z} \in \mathbb{R}^N$ as

$$z_i = [f(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i = f(\{[\mathbf{S}^k\mathbf{x}]_j : j \in \mathcal{N}_i\}). \tag{7}$$

That is, the SLO shifts the signal k times to obtain $\mathbf{S}^k\mathbf{x}$, and then replaces the value of this signal at each node i by a nonlinear aggregation $f(\cdot,\mathcal{N}_i)$ of the signal values within the one-hop neighborhood of i. The SLO utilizes information locally available at each node to account for the signal-topology coupling for nodes that are k-hops away. We can now define *graph-adaptive nonlinear graph filters* as follows.

Definition 2 (Shifted Localized Graph Filter)**.** Consider the shifted localized operator induced by an arbitrary nonlinear localized function $f(\cdot,\mathcal{N}_i)$ [cf. Def. 1], and let $\mathbf{h}_\sigma = [h_{\sigma 1},\ldots,h_{\sigma K}]^\top$ be a vector of parameters. The output of the shifted localized graph filter applied to signal $\mathbf{x}$, w.r.t. the shift operator $\mathbf{S}$, is the signal $\mathbf{z} \in \mathbb{R}^N$ with ith entry

$$z_i = \sum_{k=1}^{K} h_{\sigma k}[f(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i. \tag{8}$$

Definition 2 implies the output of a shifted localized graph filter is a linear combination of the SLOs $f(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)$ at different resolutions. Hence, shifted localized graph filters inherit the localization property of SLOs, as they incorporate the graph structure up to K hops away accessing only neighboring information. These nonlinear filters can be employed to define *graph-adaptive localized activation functions*.

Definition 3 (Graph-Adaptive Localized Activation Function)**.** Consider a scalar β and vector $\mathbf{h}_\sigma = [h_{\sigma l1}^f,\ldots,h_{\sigma lK}^f]^\top$ of learnable parameters. At layer l, the graph-adaptive localized activation function maps the linear features $\mathbf{z}_l^f$ [cf. 2] to the output features $\mathbf{x}_l^f$ following the recursion

$$[\mathbf{x}_l^f]_i = \beta\text{ReLU}([\mathbf{z}_l^f]_i) + \sum_{k=1}^{K} h_{\sigma lk}^f [f(\mathbf{S}^k\mathbf{z}_l^f,\mathcal{N}_i)]_i. \tag{9}$$

Definition 3 combines the pointwise ReLU nonlinearity and the shifted localized graph filters [cf. Def. 2] into a single graph-adaptive localized nonlinearity for GNNs. The latter is distributable and localized because, even though the resolution —given by the shift order— can be arbitrarily large, the SLO $f(\cdot,\mathcal{N}_i)$ [cf. Def. 1] operates only in the one-hop neighborhood. In Section 4, we evaluate this activation function for $f(\cdot,\mathcal{N}_i)$ being the max and median, leading to the *graph-adaptive max and median activation function*, respectively.

3.2. Graph-Adaptive Kernel Activation Functions

The graph-adaptive kernel activation functions replace the localized nonlinear function $f(\cdot,\mathcal{N}_i)$ by a localized *kernel* to enrich the representation power. Let $\mathbf{x}_i^{(k)} \in \mathbb{R}^{|\mathcal{N}_i|}$ denote the vector containing $|\mathcal{N}_i|$ copies of the k shifted signal at node i, $[\mathbf{S}^k\mathbf{x}]_i$, i.e. $\mathbf{x}_i^{(k)} = \mathbf{1}_{|\mathcal{N}_i|} \otimes [\mathbf{S}^k\mathbf{x}]_i$ where $\mathbf{1}_{|\mathcal{N}_i|}$ is the vector of ones of dimension $|\mathcal{N}_i|$ and $\otimes$ is the Kronecker operator. Additionally, consider the vector containing the values at neighbors

$j \in \mathcal{N}_i$ of the kth shifted signal $\mathbf{S}^k\mathbf{x}$, i.e. $\mathbf{x}^{(k)}_{j\in\mathcal{N}_i} = [\mathbf{S}^k\mathbf{x}]_{j\in\mathcal{N}_i}$. With this notation in place, we define a graph kernel operator as follows.

Definition 4 (Kernel Operator)**.** Let $\mathcal{G}$ be an N-node graph with shift operator $\mathbf{S}$, $\mathbf{x}$ a signal, and $\mathbf{S}^k\mathbf{x}$ the kth shifted signal. Consider an arbitrary kernel function $g(\cdot,\mathcal{N}_i) : \mathbb{R}^{|\mathcal{N}_i|} \to \mathbb{R}^{|\mathcal{N}_i|}$, which at node i computes the nonlinear local operation $[g(\mathbf{x},\mathcal{N}_i)]_i = g(\widetilde{\mathbf{x}}_i, \mathbf{x}_{j\in\mathcal{N}_i})$, where $\widetilde{\mathbf{x}}_i = \mathbf{1}_{|\mathcal{N}_i|} \otimes [\mathbf{x}]_i$ is a vector of dimensionality $|\mathcal{N}_i|$ containing copies of signal $\mathbf{x}$ at node i. The k-hop shifted kernel operator mapping from $\mathbf{x}$ to $\mathbf{z} \in \mathbb{R}^N$ has the entries

$$z_i = [g(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i := g(\mathbf{x}^{(k)}_i, \mathbf{x}^{(k)}_{j\in\mathcal{N}_i}). \tag{10}$$

Definition 4 shows the kernel operator first shifts the input signal $\mathbf{x}$ as $\mathbf{S}^k\mathbf{x}$ and then replaces the signal value at each i by the kernel value $g(\cdot,\mathcal{N}_i)$ in the one-hop neighborhood of i. Thus, the kernel operator employs only local information at each node to account for the signal-topology coupling up to k-hops away from a node. For the kernel function $g(\cdot,\mathcal{N}_i)$ we will employ the Gaussian kernel

$$g(x,y) = \exp\left(-||\mathbf{x}-\mathbf{y}||^2/2\gamma^2\right), \tag{11}$$

where scalar γ is tunable. We can now define *kernel graph filters*.

Definition 5 (Kernel Graph Filter)**.** Consider a kernel operator [cf. 4] with kernel function $g(\cdot,\mathcal{N}_i)$ and let $\mathbf{h}_\sigma = [h_{\sigma 1},\ldots,h_{\sigma K}]^\top$ be a vector of parameters. The output of the kernel graph filter applied to signal $\mathbf{x}$, w.r.t. the shift operator $\mathbf{S}$, is the signal $\mathbf{z} \in \mathbb{R}^N$ with ith entry

$$z_i = \sum_{k=1}^{K} h_{\sigma k}[g(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i. \tag{12}$$

Definition 5 implies the output of the kernel graph filter is a linear combination of the kernel operator applied to each k-shifted signal $\mathbf{S}^k\mathbf{x}$ for $1 \le k \le K$. Kernel graph filters thus preserve the localization properties of kernel operators, i.e. they account for the topology of the graph up to K-hops away accessing only information in the one-hop neighborhood. These kernel graph filters can be further employed to define the *graph-adaptive kernel activation function* as follows.

Definition 6 (Graph-Adaptive Kernel Activation Function)**.** Consider a scalar β and vector $\mathbf{h}_\sigma = [h^f_{\sigma l1},...,h^f_{\sigma lK}]^\top$ of learnable parameters. At layer l, the graph-adaptive kernel activation function maps the linear features $\mathbf{z}^f_l$ [cf. 2] to the output features $\mathbf{x}^f_l$ following the recursion

$$[\mathbf{x}^f_l]_i = \beta \text{ReLU}([\mathbf{z}^f_l]_i) + \sum_{k=1}^{K} h^f_{\sigma lk}[g(\mathbf{S}^k\mathbf{z}^f_l,\mathcal{N}_i)]_i. \tag{13}$$

Definition 6 combines the pointwise ReLU and kernel graph filters [cf. Def. 5] into a single graph-adaptive kernel activation function. This activation function is distributable and localized because, even though the resolution —given by the shift order— can be arbitrarily large, the kernel $g(\cdot,\mathcal{N}_i)$ operates only in the one-hop neighborhood.

In both proposed activation functions, coefficients $\{\beta,\mathbf{h}_\sigma\}$ are trainable, meaning these nonlinearities adapt the multi-hop resolution weights to the task at hand. Because these coefficients are shared among nodes, the number of parameters to learn for a graph-adaptive activation function is independent of the graph size. This allows GCNNs to *scale*. Note that even though the nonlinear functions $f(\cdot,\mathcal{N}_i)$ or the kernel functions $g(\cdot,\mathcal{N}_i)$ act only on the one-hop neighborhood, they are applied to the shifted signals $\mathbf{S}^k\mathbf{x}$, therefore they account for the feature-graph coupling (up to K-hops away) in a nonlinear fashion. This is an advantage over traditional GCNNs with pointwise nonlinearities, in which the graph topology is only incorporated through linear encodings generated by graph convolutions.

Definitions 3 and 6 implement *fully graph-adaptive* GCNNs that, at each layer, apply a graph convolution followed by a graph-adaptive activation function. The distributed GCNN is given by the map

$$\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H},\mathcal{W}) := \tilde{\mathbf{y}}. \tag{14}$$

The GCNN output now depends on both the coefficients $\mathcal{H}$ [cf. (5)] and on the nonlinear activation functions coefficients $\mathcal{W} = \{\mathbf{h}^f_{\sigma l}\}_{lf} \cup \{\beta\}$.

3.3. Properties of Graph-Adaptive Nonlinearities

A key property GCNNs with pointwise activation functions inherit from graph convolutions is *permutation equivariance* [15]. The output of a GCNN is invariant to node relabeling and, more importantly, GCNNs exploit graph symmetries to generalize learned representations to different graph signals that share some of these symmetries. Herein, we show that permutation equivariance also holds for graph-adaptive nonlinearities. We will also discuss a property that is specific to the graph-adaptive localized max activation: Lipschitz stability to input perturbations.

Permutation equivariance. Consider the graph convolutional filter $\mathbf{H}(\mathbf{S})$ [cf. (1)] and let $\mathbf{P}$ be an $N \times N$ permutation matrix satisfying $\mathbf{P}^\mathsf{T}\mathbf{P} = \mathbf{P}\mathbf{P}^\mathsf{T} = \mathbf{I}$. If we permute the GSO $\mathbf{S}$ and input $\mathbf{x}$ respectively as $\mathbf{S}' = \mathbf{P}^\mathsf{T}\mathbf{S}\mathbf{P}$ and $\mathbf{x}' = \mathbf{P}^\mathsf{T}\mathbf{x}$, we get the corresponding graph convolution output

$$\mathbf{y}' = \mathbf{H}(\mathbf{S}')\mathbf{x}' = \mathbf{H}(\mathbf{P}^\mathsf{T}\mathbf{S}\mathbf{P})\mathbf{P}^\mathsf{T}\mathbf{x} = \mathbf{P}^\mathsf{T}\mathbf{H}(\mathbf{S})\mathbf{P}\mathbf{P}^\mathsf{T}\mathbf{x} = \mathbf{P}^\mathsf{T}\mathbf{y}\,. \tag{15}$$

Because pointwise activation functions are scalar and by definition permutation equivariant, (15) implies GCNNs with pointwise nonlinearities are invariant to node relabelings. For GCNNs with graph-adaptive activation functions, it is then desirable to retain this property. This is guaranteed by the following proposition.

Proposition 1 (Permutation equivariance)**.** Consider a graph signal $\mathbf{x}$ defined on an N-node graph $\mathcal{G}$ with GSO $\mathbf{S}$. Let $\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H},\mathcal{W})$ be the output of a GCNN with graph-adaptive activation functions [cf. (14)] and let $\mathbf{P}$ be an $N \times N$ permutation matrix. The GNN $\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H},\mathcal{W})$ satisfies

$$\mathbf{\Phi}(\mathbf{P}^\top\mathbf{x};\mathbf{P}^\top\mathbf{S}\mathbf{P},\mathcal{H},\mathcal{W}) = \mathbf{P}^\top\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H},\mathcal{W}) \tag{16}$$

i.e., GNNs with graph-adaptive activation functions are permutation equivariant.

Proof. For the proof, we refer the reader to the Appendix. □

Lipschitz stability. In addition to permutation equivariance, the *graph-adaptive max nonlinearity* is Lipschitz stable to input perturbations with respect to the infinity norm $\|\cdot\|_\infty$ as stated in the following proposition.

Proposition 2 (Lipschitz stability)**.** Let $\mathcal{G}$ be a graph with GSO $\mathbf{S}$. Assume that $\mathbf{S}$ is normalized by its largest eigenvalue so that its spectral norm $\rho(\mathbf{S})$ is unitary. Let $\mathbf{x}$ be a graph signal and let $\tilde{\mathbf{x}}$ be a perturbation of $\mathbf{x}$. The output of the graph-adaptive max activation function

$$[\mathbf{z}]_i = \beta \text{ReLU}([\mathbf{x}]_i) + \sum_{k=1}^{K} h_{\sigma k}[\max(\mathbf{S}^k\mathbf{x},\mathcal{N}_i)]_i \tag{17}$$

with coefficients $|h_{\sigma k}| \le C$ is Lipschitz stable to input perturbations in the infinity norm $\|\cdot\|_\infty$. That is, there exists a constant L_σ such that

$$\|\tilde{\mathbf{z}} - \mathbf{z}\|_\infty \le L_\sigma \|\tilde{\mathbf{x}} - \mathbf{x}\|_\infty \tag{18}$$

where $L_\sigma = |\beta| + KC\max_k \|\mathbf{S}^k\|_\infty$.

Proof. For the proof, we refer the reader to the Appendix. □

Proposition 2 implies the graph-adaptive max activation is Lipschitz stable *at each node*. Lipschitz stability is crucial to make learning more robust. For instance, in classification problems a GNN with graph-adaptive max nonlinearities will more likely classify correctly a perturbed signal $\tilde{\mathbf{x}}$ than a GNN with non-Lipschitz activation functions. The Lipschitz constant depends on the coefficient β, the number of filter taps K, the weights $h_{\sigma k}$ (through C), and the graph (through $\max_k \|\mathbf{S}^k\|_\infty$). While we may not have full control over $\max_k \|\mathbf{S}^k\|_\infty$, β and K are design parameters, and so is the maximum value of the coefficients $h_{\sigma k}$. The Lipschitz constant of graph-adaptive max nonlinearities is thus *tunable*. This represents an advantage compared to conventional pointwise activation functions, which are stable but have fixed Lipschitz constants.

4. NUMERICAL EXPERIMENTS

We evaluate the performance of six activation functions that include: ReLU, localized activation functions (max and median) [15], and our proposed graph-adaptive localized (max and median) and kernel activation functions. Our goal is to highlight the benefits and limitations of the different nonlinearities in applications requiring distributed computations with both synthetic and real data. To train the GCNNs we used the ADAM optimizer with learning rate 10^{-3} and forgetting factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$. As the GSO, we employ the adjacency matrix normalized by the maximum eigenvalue. For the graph-adaptive kernel nonlinearity, we set the parameter γ in (11) to $\gamma = 0.1$

4.1. Source Localization

We consider a diffusion process over a graph of $N = 40$ nodes divided into $C = 4$ communities. The goal is to determine the source community of a given diffused signal locally at a selected node; hence, the distributed implementation is essential. The graph is an undirected stochastic block model (SBM) with intra- and inter-community probabilities $p = 0.8$ and $q = 0.1$, respectively. The graph signals are defined as Kronecker deltas $\boldsymbol{\delta}_c \in \mathbb{R}^N$ centered at a source node c and diffused at a timestamp $t \in [0, 30]$, i.e. $\mathbf{x}_t = \mathbf{S}^t \boldsymbol{\delta}_c$. We choose as source node c each of the 40 nodes, thus generating a data set consisting of 1200 graph signals. We split these samples into training, validation, and test set respectively as 80%, 10%, and 10%. We simulate 10 different graphs and generate 10 different splits per graph. The training and testing are performed for the highest connected node for each community, resulting in four nodes. Training is performed for 400 epochs with a batch size of 100 samples.

Table 1 shows the classification accuracy for different number of features, $F \in \{2, 4, 8\}$. For the graph-adaptive nonlinearities, we carried out the experiments with resolutions $K = 1$ and $K = 2$. We only report the results for the better performing filter order, as the rest were comparable to the localized nonlinearities from [15]. We observe both the localized nonlinearities and the proposed graph-adaptive nonlinearities significantly outperform ReLU, with a difference in classification accuracy of at least 14%. This result highlights the benefits of accounting for the graph topology during classification. Moreover, the graph-adaptive max and median activation functions outperform their localized versions, confirming the advantage of accounting for further away data-graph coupling. The max nonlinearities achieve a higher accuracy than medians in both the localized and graph-adaptive localized nonlinearities. This result could be caused by the fact that the median will overall smooth the signal, hence undermining some local variations important for classification. Additionally, this could also explain the lower performance of the graph-adaptive kernel nonlinearities compared to the localized nonlinearities, which might be affected by the possible redundancies in the extra information coming from neighbors.

4.2. Distributed Finite-Time Consensus

Distributed finite-time consensus aims to achieve consensus among all nodes in finite-time, by accessing only local information at each node. We consider learning the distributed consensus function in a data-driven fashion over an undirected SBM graph with $N = 100$ nodes divided into $C = 5$ communities with intra- and inter-community probabilities $p = 0.8$ and $q = 0.1$, respectively. The graph signals are generated from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. We generate 2500 samples and split them into 80%, 10%, 10% training, validation, and test sets, respectively. We average the performance across 10 different graph realizations and 10 different data splits for each graph. We consider a two layer GCNN with $F = 32$ features per layer followed by a per-node fully connected layer. We employ various number of filter orders $K \in \{20, 25, 30, 35\}$. Training is performed for 400 epochs with batch size 100. The evaluation metric is the RMSE.

Figure 1a shows the RMSE as a function of the filter order for the different nonlinearities. All GCNNs achieve a lower RMSE compared with the FIR graph filter. For the lowest order $K = 20$, ReLU yields a worse RMSE than the localized and graph-adaptive nonlinearities. Once

Table 1: Source Localization Test Accuracy. L.: localized nonlinearities [cf. 6]; G.A.: graph-adaptive nonlinearities [cf. (9) and (13)]. Between brackets the filter order K is specified.

Nonlinearity/ F	2	4	8
ReLU	$\mathbf{47.9}(\pm\mathbf{12.1})\%$	$44.9(\pm 15.6)\%$	$47.2(\pm 15.5)\%$
Max L.	$64.5(\pm 8.0)\%$	$69.7(\pm 8.6)\%$	$\mathbf{72.2}(\pm\mathbf{7.7})\%$
Max G.A. (2)	$64.9(\pm 7.6)\%$	$69.2(\pm 7.0)\%$	$\mathbf{73.9}(\pm\mathbf{6.8})\%$
Median L.	$61.6(\pm 7.4)\%$	$65.1(\pm 8.3)\%$	$\mathbf{69.6}(\pm\mathbf{7.2})\%$
Median G.A. (2)	$65.4(\pm 7.5)\%$	$65.6(\pm 7.6)\%$	$\mathbf{71.3}(\pm\mathbf{7.1})\%$
Kernel G.A. (1)	$58.6(\pm 9.5)\%$	$57.4(\pm 10.2)\%$	$\mathbf{61.9}(\pm\mathbf{10.7})\%$

the filter order increases, and thus the degrees of freedom, adding a parametric nonlinearity seems to be less beneficial because the network has enough degrees of freedom in the filter to model the consensus function. We also experiment with the robustness of the different models to link losses by removing graph edges with different probabilities, following the random edge sampling model of [17]. For each method, we considered the best performing setup. From the trained graph $\mathcal{G}$, we randomly removed edges with probabilities in the interval $[0.025, 0.15]$. The results are shown in Figure 1b, averaged across 10 realizations. Although all models deteriorate when the link losses increase, graph-adaptive nonlinearities handle the stochasticity better. The kernel nonlinearity seems to be the most sensitive as its performance reaches those of the other graph-adaptive alternatives.

4.3. Distributed Regression

We perform distributed regression using the Molene dataset, which contains hourly temperature measurements of $N = 32$ stations over $T = 744$ hours recorded in January 2014 in the area of Brest (France). Using the node (station) coordinates, we generate a weighted geometric graph using a ten nearest neighbor approach proposed in [18]. We consider as graph signals the measurements taken at different timestamps $t \in T$. Thus, our data set consists of 744 graph signals. On top of the original signals we add zero-mean noise with a signal-to-noise ratio (SNR) of 3 dB. These noisy signals are split into 80%, 10%, 10% training, validation, and test sets, respectively. Our goal is to train a GCNN for removing the noise distributively. We employ a GCNN with one layer and a varying number of features $F \in \{1, 2, 4\}$ and filter orders $K \in \{1, 2, 4, 8\}$. We perform the training for 500 epochs with a batch size of 100 samples. We employ RMSE as the evaluation metric. The final results are averaged across 20 different splits of the data set.

Figure 1c shows the RSME as a function of the filter order for the different nonlinearities. Across all GCNNs the best performance was achieved for the highest number of features, four, so we only report the results for this setup. In the other setups, the performances were comparable. All GCNNs perform better than the FIR, but the difference is more significant for the lowest filter order $K = 1$, especially in the case of graph-adaptive localized nonlinearities. This finding suggests their applicability in situations where the communication resources are limited. To further address this hypothesis, we experimented with different levels of noise added to the data. For each method, we considered the setup with the lowest filter order $K = 1$. The results in Figure 1d show that the graph-adaptive and localized nonlinearities outperform or achieve comparable results to ReLU. The general trend shows an increase in performance when the SNR becomes larger, with a more significant increase for the graph-adaptive localized nonlinearities. The performance of the graph-adaptive kernel nonlinearity suffers in this scenario, as it requires higher filter orders compared to the rest. We suggest using higher orders in the latter case to fully exploit the kernel power.

4.4. Recommender Systems

We implement a GNN-based recommender system by considering a $U \times M$ rating matrix $\mathbf{R}$, containing 100,000 ratings given by $U = 943$ users to $M = 1682$ movies in the MovieLens 100k dataset [19]. The entries $[\mathbf{R}]_{um}$ are the ratings between 1 and 5 if user u has rated movie m, and 0 otherwise. We interpret the rows of $\mathbf{R}$, i.e. the user rating vec-

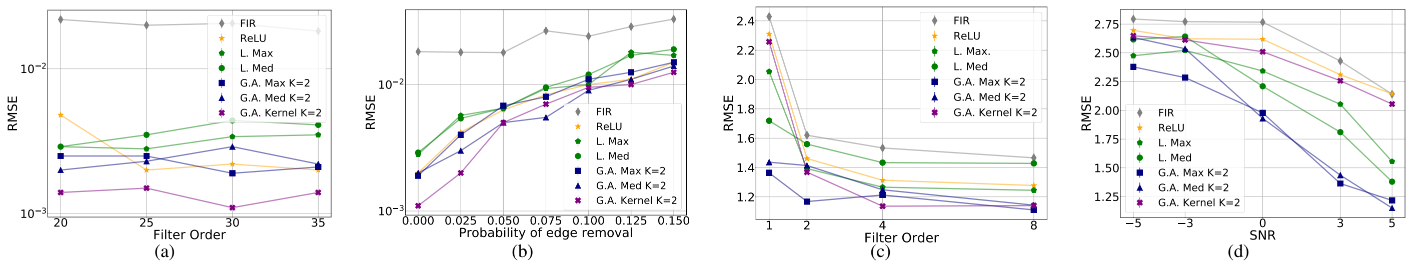


Fig. 1. (a) Root mean square error (RMSE) of the GCNNs and FIR graph filters for distributed finite-time consensus as a function of filter order. (b) Robustness of the GCNNs and FIR graph filters for distributed finite-time consensus as a function of link-loss probabilities. (c) RMSE of the GCNNs and FIR graph filters for distributed regression as a function of filter order. (d) Robustness of the GCNNs and FIR graph filters for distributed regression as a function of the SNR. L.: localized nonlinearities [cf. 6]; G.A.: graph-adaptive nonlinearities [cf. (9) and (13)]. K denotes the filter order.

Table 2: Average test RMSE over five train-test splits for the movies Toy Story, Contact and Return of the Jedi. L.: localized nonlinearities [cf. 6]; G.A.: graph-adaptive nonlinearities [cf. (9) and (13)].

Nonlinearity	Toy Story	Contact	Return of the Jedi
ReLU	$0.976(\pm 0.158)$	$1.022(\pm 0.042)$	$1.018(\pm 0.177)$
Max L.	$0.999(\pm 0.166)$	$1.028(\pm 0.029)$	$1.001(\pm 0.162)$
Max G.A.	$\mathbf{0.968(\pm 0.168)}$	$\mathbf{1.018(\pm 0.038)}$	$\mathbf{0.998(\pm 0.172)}$
Median L.	$0.987(\pm 0.156)$	$1.039(\pm 0.055)$	$1.020(\pm 0.177)$
Median G.A.	$0.989(\pm 0.160)$	$1.020(\pm 0.038)$	$1.021(\pm 0.181)$
Kernel G.A.	$0.977(\pm 0.152)$	$1.021(\pm 0.037)$	$1.014(\pm 0.177)$

tors $\mathbf{r}_u$, as graph signals on a M-node movie similarity network. The graph signals are split into 90% as training and 10% as test set, and the movie similarity network is built by computing pairwise correlations between movie rating vectors (i.e. columns of $\mathbf{R}$) containing only ratings from users in the training set. The GNN is trained to predict user ratings to a movie m. This is achieved by "zero-ing" out the ratings to movie m in the input graph signals $\mathbf{r}_u$, feeding them to the GNN to generate the rating prediction $[\overline{\mathbf{r}_u}]_m$, and minimizing the smooth ℓ_1 loss $|[\mathbf{r}_u]_m - [\overline{\mathbf{r}_u}]_m|$. We consider three graph-adaptive GNNs employing the one-hop max, one-hop median, and one-hop kernel graph-adaptive nonlinearities to highlight the impact of immediate neighboring information, hence making the recommendation more localized over items. They are compared with GNNs containing ReLU activations and the one-hop max and median activations from [15]. All GNNs consist of $L = 1$ layer, $F = 32$ features using graph convolutional filters banks with $K = 5$ filter taps each. We train all GNNs over 30 epochs and in batches of 5 for the movies Toy Story, Contact, and Return of the Jedi. The average test RMSEs over five random train-test splits for each movie are reported in Table 2.

We observe the graph-adaptive max activation function outperforms the other nonlinearities for all three movies. In particular, the graph-adaptive max fares better than both the ReLU and its localized counterpart. The graph-adaptive median also outperforms the localized median for the movie Contact, and achieves comparable performance for the other movies. As for the graph-adaptive kernel activation, it performs similarly to the ReLU and does not provide much of an improvement.

5. CONCLUSIONS

We proposed a new family of *graph-adaptive* activation functions for GNNs that capture the graph topology while also being *distributable*. These activation functions incorporate the data-topology coupling into all the GNN components by combining nonlinearized features from neighboring nodes with a set of trainable parameters. These parameters adapt the information coming from neighborhoods of different resolutions to the task at hand, hence aiding learning. The proposed graph-adaptive activation functions preserve permutation equivariance and the graph adaptive max activation function is Lipschitz stable to input perturbations. Graph-adaptive nonlinearities were compared to GCNNs employing localized and pointwise nonlinearities in four different problems based on both synthetic and real-world data, showing an improved performance compared to pointwise and other state-of-the-art localized nonlinearities. Future work will be on two fronts: characterizing the stability of the proposed activation functions to perturbations in the topology and performing learning distributively.

APPENDIX

Proof of Prop. 1. Let $\mathbf{S}' = \mathbf{P}^\top \mathbf{S}\mathbf{P}$ be the graph permutation and $\mathbf{x}' = \mathbf{P}^\top \mathbf{x}$ the permuted signal. From (15), the output of the graph convolution is equivariant to the action of $\mathbf{P}$. Hence, we only need to prove permutation equivariance of the graph-adaptive activation functions in (9) and (13). We write their output as the signal $\mathbf{z}$ with entries

$$[\mathbf{z}]_i = \beta \text{ReLU}([\mathbf{x}]_i) + \sum_{k=1}^{K} h_{\sigma k}[g(\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)]_i \tag{19}$$

where $g(\cdot, \mathcal{N}_i)$ denotes either a shifted localized operator [cf. Def 1] or a kernel operator [cf. Def. 4]. Applying the activation functions in (19) to the permuted signal $\mathbf{x}'$, we obtain

$$[\mathbf{z}']_i = \beta \text{ReLU}([\mathbf{P}^\top \mathbf{x}]_i) + \sum_{k=1}^{K} h_{\sigma k}[g((\mathbf{P}^\top \mathbf{S}\mathbf{P})^k \mathbf{P}^\top \mathbf{x}, \mathcal{N}_i)]_i. \tag{20}$$

Since the ReLU activation function is pointwise, it is permutation equivariant, i.e. $\text{ReLU}(\mathbf{P}^\top\mathbf{x}) = \mathbf{P}^\top\text{ReLU}(\mathbf{x})$. We then focus on the second term of the sum, where we observe that $(\mathbf{P}^\top\mathbf{S}\mathbf{P})^k = \mathbf{P}^\top\mathbf{S}\mathbf{P}\mathbf{P}^\top\mathbf{S}\mathbf{P}...\mathbf{P}^\top\mathbf{S}\mathbf{P} = \mathbf{P}^\top\mathbf{S}^k\mathbf{P}$, which implies $(\mathbf{P}^\top\mathbf{S}\mathbf{P})^k\mathbf{P}^\top\mathbf{x} = \mathbf{P}^\top\mathbf{S}^k\mathbf{x}$. We can rewrite $\mathbf{z}'$ as

$$[\mathbf{z}']_i = \beta[\mathbf{P}^\top \text{ReLU}(\mathbf{x})]_i + \sum_{k=1}^{K} h_{\sigma k}[g(\mathbf{P}^\top \mathbf{S}^k \mathbf{x}, \mathcal{N}_i)]_i. \tag{21}$$

Because function $g(\cdot, \mathcal{N}_i)$ is localized, it acts on the one-hop neighborhoods of each node, which are preserved under node relabelings. Therefore, $g(\cdot, \mathcal{N}_i)$ is permutation equivariant and (21) becomes

$$\begin{aligned}[\mathbf{z}']_i &= \beta[\mathbf{P}^\top \text{ReLU}(\mathbf{x})]_i + \sum_{k=1}^{K} h_{\sigma k}[\mathbf{P}^\top g(\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)]_i \\ &= [\mathbf{P}^\top \beta \text{ReLU}(\mathbf{x})]_i + \left[\mathbf{P}^\top \sum_{k=1}^{K} h_{\sigma k} g(\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)\right]_i.\end{aligned}$$

Therefore $\mathbf{z}' = \mathbf{P}^\top\mathbf{z}$ and, hence, GNNs with graph-adaptive activation functions are permutation equivariant. □

Proof of Prop. 2. Let $\tilde{\mathbf{x}}$ be a perturbed input with ith entry $[\tilde{\mathbf{x}}]_i = [\mathbf{x}]_i + \epsilon_i$. Denoting by $\tilde{\mathbf{z}}$ the output obtained by applying the graph-adaptive max activation function to $\tilde{\mathbf{x}}$, we can write

$$\begin{aligned}\|[\tilde{\mathbf{z}}]_i - [\mathbf{z}]_i\| &\le \|\beta\left(\text{ReLU}([\tilde{\mathbf{x}}]_i) - \text{ReLU}([\mathbf{x}]_i)\right)\| \\ &+ \left\|\sum_{k=1}^{K} h_{\sigma k}\left([\max(\mathbf{S}^k\tilde{\mathbf{x}}, \mathcal{N}_i)]_i - [\max(\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)]_i\right)\right\|\end{aligned} \tag{22}$$

which is obtained by grouping terms and applying the triangle inequality. The ReLU activation is Lipschitz stable with constant one [20], and so

$$\|\beta\left(\mathrm{ReLU}([\tilde{\mathbf{x}}]_i) - \mathrm{ReLU}([\mathbf{x}]_i)\right)\| \leq |\beta|\, \|[\tilde{\mathbf{x}}]_i - [\mathbf{x}]_i\| = |\beta|\, \|[\tilde{\mathbf{x}} - \mathbf{x}]_i\|\,. \tag{23}$$

For the second part of the sum in (22), we have

$$\begin{aligned}&\left\|\sum_{k=1}^{K} h_{\sigma k}\left([\max(\mathbf{S}^k\tilde{\mathbf{x}}, \mathcal{N}_i)]_i - [\max(\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)]_i\right)\right\| \\ &\leq \sum_{k=1}^{K} |h_{\sigma k}| \left\|[\max(\mathbf{S}^k\tilde{\mathbf{x}}, \mathcal{N}_i)]_i - [\max(\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)]_i\right\|\end{aligned}$$

which follows from the Cauchy-Schwarz inequality. Observe that, for any two functions $f(\cdot)$ and $g(\cdot)$, we can write the inequality $\max(f) = \max(f - g + g) \leq \max(f - g) + \max(g)$, and so

$$\begin{aligned}&\sum_{k=1}^{K} |h_{\sigma k}| \left\|[\max(\mathbf{S}^k\tilde{\mathbf{x}}, \mathcal{N}_i)]_i - [\max(\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)]_i\right\| \\ &\leq \sum_{k=1}^{K} |h_{\sigma k}| \|[\max(\mathbf{S}^k(\tilde{\mathbf{x}} - \mathbf{x}), \mathcal{N}_i)]_i\|.\end{aligned}$$

We proceed by noting that

$$\begin{aligned}\|[\max(\mathbf{S}^k(\tilde{\mathbf{x}} - \mathbf{x}), \mathcal{N}_i)]_i\| &\leq \|\max_i [\mathbf{S}^k(\tilde{\mathbf{x}} - \mathbf{x})]_i\| \\ &\leq \max_i \|[\mathbf{S}^k(\tilde{\mathbf{x}} - \mathbf{x})]_i\| = \|\mathbf{S}^k(\tilde{\mathbf{x}} - \mathbf{x})\|_\infty\end{aligned}$$

which allows us to write

$$\begin{aligned}&\left\|\sum_{k=1}^{K} h_{\sigma k}\left([\max(\mathbf{S}^k\tilde{\mathbf{x}}, \mathcal{N}_i)]_i - [\max(\mathbf{S}^k\mathbf{x}, \mathcal{N}_i)]_i\right)\right\| \\ &\leq \sum_{k=1}^{K} |h_{\sigma k}| \|\mathbf{S}^k(\tilde{\mathbf{x}} - \mathbf{x})\|_\infty \leq \sum_{k=1}^{K} |h_{\sigma k}| \|\mathbf{S}^k\|_\infty \|\tilde{\mathbf{x}} - \mathbf{x}\|_\infty \\ &\leq KC \max_k \|\mathbf{S}^k\|_\infty \|\tilde{\mathbf{x}} - \mathbf{x}\|_\infty.\end{aligned} \tag{24}$$

Putting (23) and (24) together, we can write

$$\|[\tilde{\mathbf{z}} - \mathbf{z}]_i\| = \|[\tilde{\mathbf{z}}]_i - [\mathbf{z}]_i\| \leq |\beta|\, \|[\tilde{\mathbf{x}} - \mathbf{x}]_i\| + KC \max_k \|\mathbf{S}^k\|_\infty \|\tilde{\mathbf{x}} - \mathbf{x}\|_\infty.$$

Since this is true for all i and from the definition of $\|\cdot\|_\infty$, we conclude

$$\|\tilde{\mathbf{z}} - \mathbf{z}\|_\infty = \leq \left(|\beta| + KC \max_k \|\mathbf{S}^k\|_\infty\right) \|\tilde{\mathbf{x}} - \mathbf{x}\|_\infty$$

which completes the proof. Note that $\|\mathbf{S}^k\|_\infty \geq \rho(\mathbf{S})^k = 1$ for all k with $\lim_{k\to\infty} \|\mathbf{S}^k\|_\infty = \rho(\mathbf{S})^k = 1$, so there exists K_0 such that, for all $k > K_0$, $\|\mathbf{S}^k\|_\infty \leq \max_k \|\mathbf{S}^k\|_\infty$ with $\max_k \|\mathbf{S}^k\|_\infty = \|\mathbf{S}^{K_0}\|_\infty$. □

6. REFERENCES

[1] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[2] F. Gama, E. Isufi, G. Leus, and A. Ribeiro, "Graphs, convolutions, and neural networks," *arXiv:2003.03777*, 2020.

[3] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *30th Conf. Neural Inform. Process. Syst.*, Barcelona, Spain, 5-10 Dec. 2016, pp. 3844–3858, Neural Inf. Process. Syst. Foundation.

[4] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural network architectures for signals supported on graphs," *IEEE Trans. Sig. Proc.*, vol. 67, no. 4, pp. 1034–1049, Feb. 2019.

[5] D. I. Shuman, P. Vandergheynst, D. Kressner, and P. Frossard, "Distributed signal processing via chebyshev polynomial approximation," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 4, no. 4, pp. 736–751, 2018.

[6] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Autoregressive moving average graph filtering," *IEEE Trans. Sig. Proc.*, vol. 65, no. 2, pp. 274–288, 2016.

[7] S. Segarra, A. G. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," *IEEE Trans. Sig. Proc.*, vol. 65, no. 15, pp. 4117–4131, Aug. 2017.

[8] Aliaksei Sandryhaila, Soummya Kar, and José MF Moura, "Finite-time distributed consensus through graph filters," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 1080–1084.

[9] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE transactions on automatic control*, vol. 31, no. 9, pp. 803–812, 1986.

[10] Ali Jadbabaie, Jie Lin, and A Stephen Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on automatic control*, vol. 48, no. 6, pp. 988–1001, 2003.

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *2nd Int. Conf. Learning Representations*, Banff, AB, 14-16 Apr. 2014, pp. 1–14, Assoc. Comput. Linguistics.

[12] F. M. Bianchi, D. Grattarola, C. Alippi, and L. Livi, "Graph neural networks with convolutional ARMA filters," *arXiv:1901.01343 [cs.LG]*, 2019.

[13] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Int. Conf. Learning Representations 2018*, Vancouver, BC, 30 Apr.-3 May 2018, pp. 1–12, Assoc. Comput. Linguistics.

[14] E. Isufi, F. Gama, and A. Ribeiro, "EdgeNets: Edge varying graph neural networks," *arXiv:2001.07620v1 [cs.LG]*, 21 Jan. 2020.

[15] L. Ruiz, F. Gama, A. G. Marques, and A. Ribeiro, "Invariance-preserving localized activation functions for graph neural networks," *IEEE Trans. Sig. Proc.*, vol. 68, no. 1, pp. 127–141, Jan. 2020.

[16] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *arXiv:1905.04497v2 [cs.LG]*, 4 Sep. 2019.

[17] Elvin Isufi, Andreas Loukas, Andrea Simonetto, and Geert Leus, "Filtering random graph processes over random time-varying graphs," *IEEE Transactions on Signal Processing*, vol. 65, no. 16, pp. 4406–4421, 2017.

[18] Elvin Isufi, Andreas Loukas, Nathanael Perraudin, and Geert Leus, "Forecasting time series with varma recursions on graphs," *IEEE Transactions on Sig. Proc.*, vol. 67, no. 18, pp. 4870–4885, 2019.

[19] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," *ACM Trans. Interactive Intell. Syst.*, vol. 5, no. 4, pp. 19:(1–19), Jan. 2016.

[20] T. Wiatowski and H. Bölcskei, "A mathematical theory of deep convolutional neural networks for feature extraction," *IEEE Transactions on Inf. Theory*, vol. 64, no. 3, pp. 1845–1866, 2017.

Towards Finite-Time Consensus with Graph Convolutional Neural Networks

Bianca Iancu and Elvin Isufi
Intelligent Systems Department, Delft University of Technology, Delft, The Netherlands

***Abstract*—This work proposes a learning framework for distributed finite-time consensus with graph convolutional neural networks (GCNNs). Consensus is a central problem in distributed and adaptive optimisation, signal processing, and control. We leverage the link between finite-time consensus and graph filters, and between graph filters and GCNNs to study the potential of a readily distributed architecture for reaching consensus. We have found GCNNs outperform classical graph filters for distributed consensus and generalize better to unseen topologies such as distributed networks affected by link losses.**

***Index Terms*—Finite-time consensus, graph convolutions, graph signal processing, graph neural networks.**

I. Introduction

Distributed average consensus is a fundamental problem in signal processing, sensor networks, and multi-agent control [1]–[7]. A first approach to reach consensus is through distributed iterative solvers, such as randomized gossip [8] or methods of multipliers [9]. These methods reach consensus at steady-state and their convergence rate is dominated by the network topology. A more recent direction considers reaching consensus within a finite number of iterations and frames this problem as a graph filtering operation [10].

The first work to formalize finite-time consensus through graph filters is [11]. This work uses the finite impulse response (FIR) graph filters and designs the filter coefficients by relying on the graph spectrum. Conditions on when the latter is feasible are further analyzed in [12], [13]. A main limitation of these theoretical contributions is that the filter coefficients depend on the specific eigenvalues of the graph Laplacian matrix. The cost of computing the eigendecomposition limits also their applicability to graphs of small dimensions. The designed filters suffer also from numerical issues due to the finite-precision of the eigenvalues. Besides the theoretical insights, the practical benefit of these works is to approximate better consensus in a finite number of iterations compared with the other distributed solvers. The fastest converging filter is the edge varying graph filter [14], which differently from FIRs exploits also nodes' locality and sparsity to enhance the degrees of freedom. However, the edge varying graph filter requires a fixed labeling in both design and implementation phase and the graph structure to be fixed; both assumptions that might be infeasible in practical distributed settings or when the topology changes slightly (e.g. nodes and links that fail).

In this paper, we address distributed finite-time consensus as a learning problem on graphs. We employ a distributed graph convolutional neural network (GCNN) to learn the consensus function in a data-driven fashion. GCNNs can be thought of as extending to graphs conventional CNNs, where the spatial convolutional filters are substituted by graph convolutional filters [15], [16]. By having the FIR graph filter as their integral part, GCNNs link directly to finite-time consensus if the activation functions leave unaffected the distributed implementation. The coupling *graph filter-activation function* also facilitates the transferability of GCNNs to graphs that deviate slightly from the ones they were trained on [17]. The main research question we address is *how do GCNNs behave for distributed finite-time consensus.* Our preliminary results show the potential of the GCNNs to outperform FIRs for reaching consensus. The improved performance is sensitive to the activation function and to the graph topology. Parametric activation functions should be employed when the GCNN with non-parametric ones (e.g., ReLU) has limited discriminatory power or when the communication complexity is limited. Also, better connected graphs yield a better performance. Finally, we observed GCNNs are more robust than FIRs in reaching consensus over graphs affected by link losses.

This paper is organized as follows. Section II recalls some background material about graph signal processing and distributed consensus with graph filters. Section III details the architecture and nonlinearities under study. The numerical experiments are reported in Section IV, while the paper conclusions in Section V.

II. Background

We start with some basic concepts from graph signal processing and then we continue with graph filters and their link to distributed consensus.

A. Graph signal processing

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V} = \{v_1, v_2, ..., v_N\}$ of cardinality $|\mathcal{V}| = N$ and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ of cardinality $|\mathcal{E}| = M$. An edge is a tuple $e_{ij} = (v_i, v_j)$ connecting nodes v_i and v_j. The neighborhood of node v_i is the set of nodes connected to v_i, i.e., $\mathcal{N}_i = \{v_j | (v_i, v_j) \in \mathcal{E}\}$. Associated with $\mathcal{G}$ is an $N \times N$ matrix $\mathbf{S}$, named the graph shift operator (GSO) matrix, whose sparsity pattern matches the graph structure. The entry (i, j) of $\mathbf{S}$ is $[\mathbf{S}]_{i,j} = s_{i,j} \neq 0$, if $i = j$ or $(i, j) \in \mathcal{E}$. Commonly used GSOs are the adjacency matrix $\mathbf{A}$, the graph Laplacian matrix $\mathbf{L}$ (for undirected graphs), or their normalized and translated forms.

On the vertices of $\mathcal{G}$, we define a graph signal $\mathbf{x} = [x_1, x_2, ..., x_N]^\top \in \mathbb{R}^N$, whose i-th component x_i is the signal value of node v_i. The GSO can be used to represent the signal in the graph spectral domain. For this, consider the eigendecomposition $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}$ with eigenvectors $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_N]$ and eigenvalues $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_N)$. The graph Fourier transform (GFT) of $\mathbf{x}$ is defined as $\hat{\mathbf{x}} = \mathbf{U}^{-1}\mathbf{x}$, where $\hat{x}_i$ quantifies how much eigenvector $\mathbf{u}_i$ contributes to the variation of signal $\mathbf{x}$ over the graph [10], [18]. As we shall see in the sequel, this Fourier decomposition plays a role in approaching consensus from a spectral perspective. For completeness, the inverse GFT is $\mathbf{x} = \mathbf{U}\hat{\mathbf{x}}$ and the eigenvalues $\mathbf{\Lambda}$ are referred to as the graph frequencies.

B. Consensus as graph signal filtering

Consider the signal $\mathbf{x}$ and the consensus version $\bar{\mathbf{x}} = \bar{x}\mathbf{1}$, with $\bar{x}$ being the mean of $\mathbf{x}$ and $\mathbf{1}$ the vector of all ones. We can think of $\bar{\mathbf{x}}$ as a signal whose GFT coefficients $\hat{\bar{\mathbf{x}}}$ are such that the combined eigenvectors yield the DC component. For the GSO being the graph Laplacian $\mathbf{S} = \mathbf{L}$, this is straightforward since eigenvector $\mathbf{u}_1$ associated to the smallest eigenvalue $\lambda_1 = 0$ is constant, i.e., $\mathbf{u}_1 = 1/\sqrt{N}\mathbf{1}$. Only the first coefficient $\hat{\bar{x}}_1$ is necessary to represent the consensus signal, while all other coefficients can be null, $\hat{\bar{x}}_2 = \ldots = \hat{\bar{x}}_N = 0$. For $\mathbf{S}$ being the adjacency matrix or any other graph representation matrix that does not have a constant eigenvector, vector $\hat{\bar{\mathbf{x}}}$ will have more than one entry (if not all) non-zero to represent the constant signal.

We can think of consensus as a graph filter that takes an heterogeneous graph signal $\mathbf{x}$ and filters it to return the constant mean signal $\bar{\mathbf{x}} = \bar{x}\mathbf{1}$ over the nodes [11]. A graph filter matrix $\mathbf{H}(\mathbf{S})$ w.r.t. the GSO $\mathbf{S}$ is defined as the polynomial matrix of order K

$$\mathbf{H}(\mathbf{S}) = \sum_{k=0}^{K} h_k \mathbf{S}^k \tag{1}$$

that takes as input a graph signal $\mathbf{x}$ to return the output signal $\mathbf{y} = \mathbf{H}(\mathbf{S})\mathbf{x}$. Vector $\mathbf{h} = [h_0, \ldots, h_K]^\top$ contains the $K+1$ filter coefficients. Exploiting the GFT, we can write the input-output graph filtering relation as $\hat{\mathbf{y}} = \mathbf{H}(\mathbf{\Lambda})\hat{\mathbf{x}}$, where the diagonal matrix $\mathbf{H}(\mathbf{\Lambda}) = \sum_{k=0}^{K} h_k \mathbf{\Lambda}^k$ contains the filter frequency response on the main diagonal. Reaching consensus with graph filters of the form in (1) accounts for learning the filter parameters $\mathbf{h}$ such that the signal is low-pass filtered to pass only the DC component.

Another advantage of (1) is its readily distributed implementation. In building the output $\mathbf{y}$, we need to compute the terms $\mathbf{S}\mathbf{x}, \mathbf{S}^2\mathbf{x}, \ldots, \mathbf{S}^K\mathbf{x}$. By exploiting the recursion $\mathbf{S}^k\mathbf{x} = \mathbf{S}(\mathbf{S}^{k-1}\mathbf{x}) = \mathbf{S}\mathbf{x}^{(k-1)}$, node i can compute the shifted signal $\mathbf{x}^{(k)}$ by exchanging previous shift information $\mathbf{x}^{(k-1)}$ with its direct one-hop neighbors $\mathcal{N}_i$, since the shift operator is local. This recursive implementation allows for a distributed communication and computational cost of order $\mathcal{O}(MK)$ [12].

The main benefit of (1) is that, under appropriate conditions on the spectrum of $\mathbf{S}$ [13], coefficients $\mathbf{h}$ can be designed to achieve exact finite-time consensus in at most $K = N$ iterations [11], [19]. However, their applicability is limited to simple (cyclic or star) graphs, since these approaches require high numerical precision of the eigenvalues. An approach to tackle the numerical issues is to consider a different graph filter in (1), such as ARMA [20], node varying [12], or edge varying [21]. Of particular interest is the so-called edge varying graph filter [21], which substitutes scalars h_k with $N \times N$ coefficient matrices $\mathbf{H}_k$ in which entry (i, j) is the coefficient applied to edge e_{ij}. In this case, finite-time consensus can be approximated with higher accuracy compared with (1), but the graph structure and its labeling should be fixed. The latter is also practically non-transferable to a slightly different graph, such as a graph affected by link losses.

Employing instead a GCNN with filters of the form in (1) does not require the GSO eigendecomposition, a fixed labeling, and it is better transferable to unseen graphs than the linear graph filter [17].

III. Methods

In this section, we first detail the GCNN architecture and the activation functions under study. Then, we discuss two properties of the GCNN, namely, the permutation equivariance and transference to unseen graphs and their suitability to distributed consensus.

Architecture. We consider a GCNN composed of L graph convolutional layers followed by a *per node* fully connected layer –Figure 1. Each graph convolutional layer comprises a bank of graph filters [cf. (1)] and a nonlinearity. At layer l, the GCNN takes F input features $\{\mathbf{x}_{l-1}^g\}_{g=1}^F$ and produces other F output features $\{\mathbf{x}_l^f\}_{f=1}^F$. Each input feature $\mathbf{x}_{l-1}^g$ is treated as graph signals and processed by a parallel bank of F graph filters $\{\mathbf{H}_l^{fg}\}_f$ of the form (1). The filter outputs are then aggregated over the input index g to yield the f-th intermediate feature

$$\mathbf{z}_l^f = \sum_{g=1}^{F} \mathbf{H}_l^{fg}(\mathbf{S})\mathbf{x}_{l-1}^g = \sum_{g=1}^{F}\sum_{k=0}^{K} h_{kl}^{fg}\mathbf{S}^k\mathbf{x}_{l-1}^g, \text{ for } f \in \{1, \ldots, F\}. \tag{2}$$

The intermediate feature $\mathbf{z}_l^f$ is another graph signal whose i-th entry $[\mathbf{z}_l^f]_i$ is associated with node v_i. The latter is subsequently passed through an activation function $\sigma(\cdot)$ to yield the f-th output of the l-th convolutional layer

$$\mathbf{x}_l^f = \sigma(\mathbf{z}_l^f), \quad \text{for } f \in \{1, \ldots, F\}. \tag{3}$$

Layer l is characterized by the F^2 coefficient vectors $\mathbf{h}_l^{fg} = [h_{0l}^{fg}, \ldots, h_{Kl}^{fg}]^\top$ of filters $\mathbf{H}_l^{fg}(\mathbf{S})$ in (2). Remark the number of input and output features do not need to be the same, but we assume so to ease notation.

The input feature of layer $l = 1$ is the graph signal $\mathbf{x}_0 := \mathbf{x}$ for which we want to reach consensus. The output features of layer L, $\mathbf{x}_L^1, \ldots, \mathbf{x}_L^F$, represent the final convolutional features. The latter can also be seen as a collection of F graph signals, where on node v_i we have the $F \times 1$ feature vector $\boldsymbol{\chi}_{Li} =$

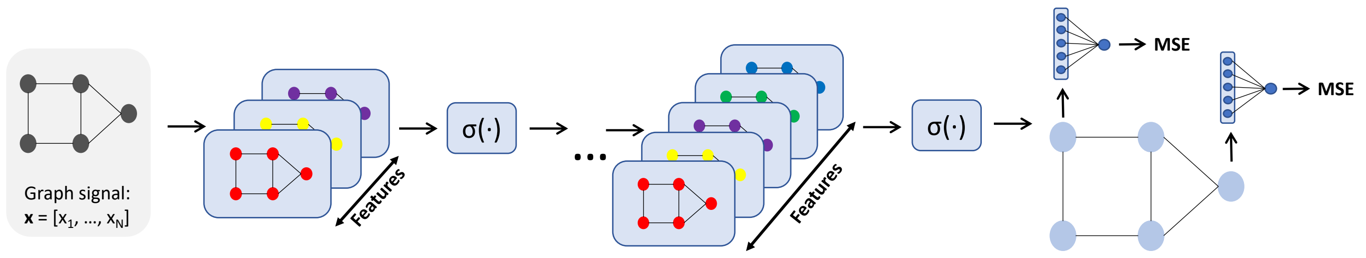


Fig. 1: Distributed GCNN architecture for finite-time consensus. The input is a graph signal $\mathbf{x}$, which is filtered by a filter bank of F FIR graph filters [cf. (1)] and then passed through and activation function $\sigma(\cdot)$. This forms a graph convolutional layer, which is cascaded L times. The final convolutional features are concatenated per node and passed to a *per-node* fully connected layer to compute the final output. This output is used during training the minimze the mean squared error (MSE).

$[x_{Li}^1, \ldots, x_{Li}^F]^\top$. Each node locally combines the features $\boldsymbol{\chi}_{Li}$ with a one-layer perceptron[1] to build the final scalar output

$$y_i = \mathbf{h}_{\text{FC}}^\top \boldsymbol{\chi}_{Li} \tag{4}$$

where $\mathbf{h}_{\text{FC}} = [h_1, \ldots, h_F]^\top$ is the $F \times 1$ vector of parameters in the local fully connected layer. Vector $\mathbf{h}_{\text{FC}}$ is shared among all nodes to keep the number of trainable parameters independent from the graph dimensions.

Activation functions. If the activation functions in the convolutional layers were local, the GCNN would be readily distributable. In fact, all filters $\{\mathbf{H}_l^{fg}(\mathbf{S})\}_{fgl}$ are distributable, as discussed in Section II-B. The last fully connected layer leaves unaffected the distributed implementation since it is local over the nodes. In this work, we study the effect of three activation functions for distributed consensus: the pointwise ReLU, the pointwise kernel [22], and the local max [23].

ReLU: The rectified linear unit is pointwise on each scalar entry x_i of the feature vector $\mathbf{x}$ and it is defined as

$$\sigma(\mathbf{x}) = \max(\mathbf{0}, \mathbf{x}). \tag{5}$$

Kernel: The pointwise kernel activation function considers a one-dimensional dictionary $\mathbf{d} = [d_1, \ldots, d_D]^\top$ of D atoms sampled uniformly around zero. Any scalar feature x_i of node v_i is combined with all elements of $\mathbf{d}$ to build the parametric nonlinear features

$$\sigma(x_i) = \sum_{j=1}^{D} h_j \kappa(x_i, d_j) \tag{6}$$

where $\mathbf{h}_\sigma = [h_1, \ldots, h_D]^\top$ is a $D \times 1$ vector of trainable parameters and $\kappa(x_i, d_j)$ is a one-dimensional kernel between feature value x_i and dictionary atom d_j. Following [22], we employ the Gaussian kernel function $\kappa(x_i, d_j) = \exp\big(-\gamma(x_i - d_j)^2\big)$, where γ is a tuneable parameter.

Max local: Differently from the above two, the max local activation function is not pointwise at node v_i but takes into account also feature values at neighboring nodes $\mathcal{N}_i$. Let $\mathbf{x}$ be an $N \times 1$ graph signal feature on which we want to apply the max local activation function. Then, the output of a local max operator $\max(\mathbf{S}, \mathbf{x})$ applied to signal $\mathbf{x}$ is another graph signal $\mathbf{z}$ whose i-th entry z_i is the maximum value in the neighboorhood, i.e., $z_i = [\max(\mathbf{S}, \mathbf{x})]_i = \max\big(\{x_j : v_j \in \mathcal{N}_i\}\big)$. The max local activation function for the feature signal $\mathbf{x}$ builds the parametric features

$$\sigma(\mathbf{x}) = h_0 \max(\mathbf{0}, \mathbf{x}) + h_1 \max(\mathbf{S}, \mathbf{x}). \tag{7}$$

with trainable parameters $\mathbf{h}_\sigma = [h_0, h_1]^\top$. The ReLU term nonlinearizes also the node features. In [23], the authors extended (7) to a neighborhood of order K. This choice, however, is not distributable and we shall not discuss it further.

The above activation functions leave unaffected the communication and computational costs of the GCNN, which remain governed by the cost of running all graph filters [cf.(1)]. For an architecture of F features per layer and L graph convolutional layers the cost is of order at most $\mathcal{O}(F^2LMK)$.

Parameter training. If the ReLU nonlinearity is used, the total number of parameters of the GCNN is $F^2(L-1)(K+1)+F(K+1)+F$. This divides as: *i)* $F(K+1)$ parameters for the F filters in the first graph convolutional layer; *ii)* $F^2(K+1)(L-1)$ for the parameters of the F^2 filters in the remaining $L-1$ graph convolutional layers; and *iii)* F parameters in the final fully-connected layer. Instead, if the kernel or the max local activation functions are used, we should also consider the parameters in $\mathbf{h}_\sigma$. This adds DL or $2L$ parameters for the kernel or the max local activation function, respectively.

By grouping all parameters into set $\mathcal{H} = \{\mathbf{h}_l^{fg}; \mathbf{h}_{\sigma l}; \mathbf{h}_{\text{FC}}\}_{lfg}$, we can consider the GCNN as a mapping $\boldsymbol{\Phi}(\cdot)$ that takes as input a graph signal $\mathbf{x}$, a GSO $\mathbf{S}$, and a set of parameters $\mathcal{H}$ to produce the output

$$\boldsymbol{\Phi}(\mathbf{x}; \mathbf{S}; \mathcal{H}) := \tilde{\mathbf{y}}. \tag{8}$$

The output (8) is computed for a training set $\mathcal{T} = \{(\mathbf{x}_r, \mathbf{y}_r)\}$ of $|\mathcal{T}| = R$ pairs, where the input $\mathbf{x}_r$ is a graph signal and $\mathbf{y}_r$ is the vector containing the consensus signal $\bar{x}_r$ for all nodes; i.e., $\mathbf{y}_r = \bar{x}_r \mathbf{1}$. The goal of the GCNN is to learn the distributed averaging function from examples in $\mathcal{T}$ and extrapolating it to unseen graph signals $\mathbf{x} \notin \mathcal{T}$.

As a loss function, we considered the averaged mean squared error (MSE) between the GCNN output $\tilde{\mathbf{y}}_r$ and the

[1]Each node can also consider a local multi-layer perceptron to combine the features in $\boldsymbol{\chi}_{Li}$.

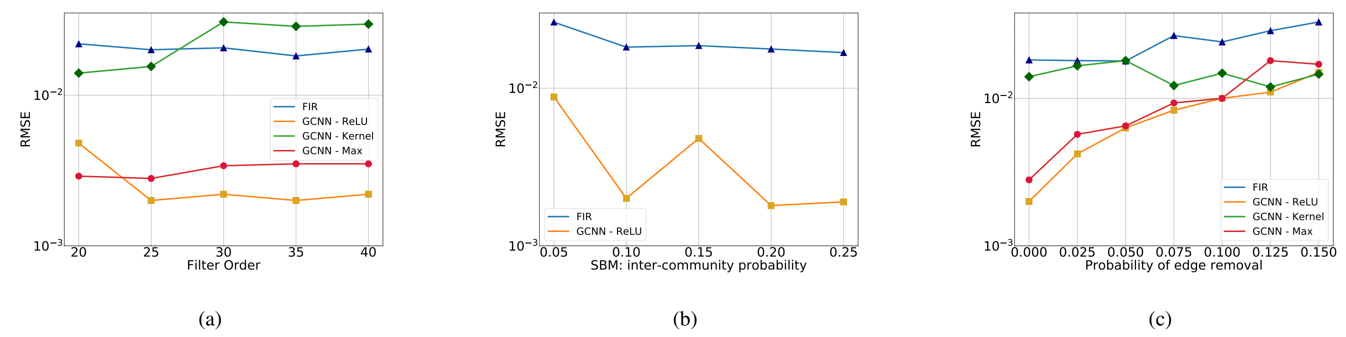


Fig. 2: Root mean square error (RMSE) of the GCNN and FIR graph filters for reaching finite-time consensus. (a) Comparison of different filter orders (iterations) and nonlinearities. (b) Performance of the FIR and GCNN with ReLU nonlinearity as a function of the graph connectivity. (c) Robustness of the different models as a function of link loss probability.

label $\mathbf{y}_r$; i.e.,

$$\mathcal{L} = \frac{1}{R}\sum_{r=1}^{R} \|\mathbf{y}_r - \tilde{\mathbf{y}}_r\|_2^2. \tag{9}$$

This loss is minimized w.r.t. parameters in $\mathcal{H}$ using standard backpropagation with stochastic gradient descent or any other preferred descent method.

Equivariance and transference. The coupling *graph filter-activation function* embodies the GCNN with two important properties, namely, permutation equivariance and transference to unseen graphs. Permutation equivariance implies that the processing of a graph signal with the GCNN is independent of node labeling. This is satisfactory for distributed consensus because we would like to train the GCNN on a graph $\mathcal{G}$ and deploy it on any permuted version of $\mathcal{G}$. Although permutation equivariance restricts the family of activation functions, pointwise nonlinearities and the max local nonlinearity are permutation equivariant [23].

The transference of the GCNN to unseen graphs is essential for distributed consensus since in practical scenarios communication links are prone to perturbations. Transference properties of the graph filters [cf. (1)] and of the GCNN [cf. (8)] are recently linked with their ability to be robust to perturbations [17], [24]. Next, we investigate this property for consensus and observe that GCNNs have a better transference to unseen graphs compared with the FIR filter (1).

IV. Numerical Results

We evaluate the impact of the three activation functions, ReLU (5), kernel (6), and localized max (7), for the GCNN architecture (8) and compare their performance with the FIR graph filter (1). Our goal is to highlight the benefits and limitations of the different activation functions as well as provide preliminary insights on the GCNN behavior when employed for distributed consensus. In specific, the research questions we aim to answer are:

RQ.1 What is the impact of the activation function and filter order on the GCNN?

RQ.2 What is the impact of the graph connectivity when learning the GCNN consensus function?

RQ.3 How do different activation functions behave when the GCNN is deployed on different graphs?

Setup. We considered an undirected stochastic block model (SBM) graph of $N = 100$ nodes divided into $C = 5$ communities with intra- and inter-community probabilities $p = 0.8$ and $q = 0.1$, respectively. The graph signals are generated from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. We generated 2500 samples and split them into 80%, 10%, 10% training, validation and test sets, respectively. We averaged the performance across 10 different graph realizations and 10 different data splits for each graph. The GSO is the normalized adjacency matrix $\mathbf{S} = \mathbf{A}/\lambda_{\max}(\mathbf{A})$, where $\lambda_{\max}(\mathbf{A})$ is the maximum eigenvalue of the adjacency matrix[2]. The considered architecture is a two layer GCNN with $F = 32$ features per layer followed by a per-node fully connected layer. To train the parameters, we considered the ADAM optimizer with learning rate 0.001 and forgetting factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for 400 epochs and batch size of 100 samples. For the kernel activation function, we considered the same parameters as in [22].

Non-linearity and filter order. We analyzed the three activation functions in Section III and filter orders in the set $K \in \{20, 25, 30, 35, 40\}$. Since for consensus we want the filters to approximate a strongly low-pass transfer function, low filter orders (e.g., $K \in \{1, \dots, 5\}$ as used for classification) significantly affect the performance. From Fig. 2a, we see the ReLU and the local max activation functions achieve a significantly lower root MSE compared with the kernel activation function but also with the FIR graph filter. The local max performs better than ReLU only for the lowest order $K = 20$, which goes in line with the classification results in [23]. When the filter order K increases, hence the degrees of freedom, adding a parametric nonlinearity is a disadvantage. In fact, the kernel activation function (6) has $D = 20$ extra

[2]We also experimented with the Laplacian as GSO but its performance was consistently worse compared with the normalized adjacency matrix.

parameters per layer and yields a worse performance compared with the local max which has two additional parameters. These observations suggest that parametric activation functions should be preferred when a GCNN architecture with non-parametric ones has a low discriminatory power or when the communication cost is limited.

Graph connectivity. To analyze the impact of the graph connectivity when learning the consensus function, we evaluated the inter-community edge formation probability in the interval $q \in [0.05, 0.25]$. In Fig. 2b, we compare directly the ReLU nonlinearity for $K = 25$ with the FIR graph filter since it was the best performing architecture. For both methods, we observe a lower RMSE when the communities are better connected. This finding is intuitively satisfying for distributed consensus, as the better connected the communities the easier nodes get information from further away neighbors.

Robustness. In this last experiment, we analyze the robustness of the different methods when transferred to graphs affected by link losses. For each method, we considered the best performing order. From the trained graph $\mathcal{G}$, we randomly removed edges with probabilities in the interval $[0.025, 0.15]$. Fig. 2 illustrates the performance averaged over 10 additional realizations. All GCNN models outperform the FIR. It is, however, remarkable that the kernel activation function is quite robust to link losses compared with the rest. We attribute this behavior to the increased degrees of freedom, which trade performance with robustness.

V. Conclusions and Future Work

We proposed a data-driven framework for addressing finite-time consensus with GCNNs. We exploited the link between consensus, graph convolutional filters, and GCNNs to propose a method that is readily distributable if the activation functions are properly chosen and the multilayer perceptron is applied per node. Our preliminary results suggest: $i)$ parametric activation functions should be employed when the distributed graph filters embedded into a non-parametric nonlinearity have limited discriminatory power –the latter is often linked to communication complexity (i.e., filter order); $ii)$ better connected graphs facilitate learning the consensus function –our rationale is the improved performance is because each node gets easier the information from all other nodes; $iii)$ GCNNs generalize better to unseen graphs compared with FIR graph filters. These preliminary observations show the potential of the GCNNs for finite-time consensus rather than being conclusive. Three interesting research directions should be addressed in future work. First, theoretical research is needed to investigate the limits of GCNN for finite-time consensus and link them with the graph spectrum. Second, extensive results in different graphs are needed to validate our observations. Third, it is worth investigating an asynchronous implementation since the latter has often shown superior performance compared with the synchronous one. This work, nevertheless, shows GCNNs overcome by a margin FIR graph filters for finite-time consensus.

References

[1] M. H. DeGroot, "Reaching a consensus," *Journal of the American Statistical Association*, vol. 69, no. 345, pp. 118–121, 1974.

[2] J. Tsitsiklis, D. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE transactions on automatic control*, vol. 31, no. 9, pp. 803–812, 1986.

[3] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on automatic control*, vol. 48, no. 6, pp. 988–1001, 2003.

[4] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[5] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[6] S. Sundaram and C. N. Hadjicostis, "Distributed function calculation and consensus using linear iterative strategies," *IEEE journal on selected areas in communications*, vol. 26, no. 4, pp. 650–660, 2008.

[7] S. Pequito, S. Kruzick, S. Kar, J. M. Moura, and A. P. Aguiar, "Optimal design of distributed sensor networks for field reconstruction," in *21st European Signal Processing Conference (EUSIPCO 2013)*. IEEE, 2013, pp. 1–5.

[8] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2508–2530, 2006.

[9] T. Sherson, R. Heusdens, and W. B. Kleijn, "On the distributed method of multipliers for separable convex optimization problems," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 3, pp. 495–510, 2019.

[10] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," vol. 30, no. 3, pp. 83–98, May 2013.

[11] A. Sandryhaila, S. Kar, and J. M. Moura, "Finite-time distributed consensus through graph filters," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1080–1084.

[12] S. Segarra, A. G. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," vol. 65, no. 15, pp. 4117–4131, Aug. 2017.

[13] M. Coutino, E. Isufi, T. Maehara, and G. Leus, "On the limits of finite-time distributed consensus through successive local linear operations," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2018, pp. 993–997.

[14] M. Coutino, E. Isufi, and G. Leus, "Advances in distributed graph filtering," *IEEE Transactions on Signal Processing*, vol. 67, no. 9, pp. 2320–2333, 2019.

[15] F. Gama, A. G. Marques, G. Leus, and A. Ribeiro, "Convolutional neural network architectures for signals supported on graphs," vol. 67, no. 4, pp. 1034–1049, Feb. 2019.

[16] E. Isufi, F. Gama, and A. Ribeiro, "Edgenets: Edge varying graph neural networks," *arXiv preprint arXiv:2001.07620*, 2020.

[17] F. Gama, J. Bruna, and A. Ribeiro, "Stability properties of graph neural networks," *arXiv:1905.04497v2 [cs.LG]*, 4 Sep. 2019. [Online]. Available: http://arxiv.org/abs/1905.04497

[18] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," vol. 61, no. 7, pp. 1644–1656, Apr. 2013.

[19] S. Sundaram and C. N. Hadjicostis, "Finite-time distributed consensus in graphs with time-invariant topologies," in *2007 American Control Conference*. IEEE, 2007, pp. 711–716.

[20] E. Isufi, A. Loukas, A. Simonetto, and G. Leus, "Autoregressive moving average graph filtering," vol. 65, no. 2, pp. 274–288, Jan. 2017.

[21] M. Coutino, E. Isufi, and G. Leus, "Advances in distributed graph filtering," vol. 67, no. 9, pp. 2320–2333, May 2019.

[22] S. Scardapane, S. Van Vaerenbergh, D. Comminiello, and A. Uncini, "Improving graph convolutional networks with non-parametric activation functions," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, 2018, pp. 872–876.

[23] L. Ruiz, F. Gama, A. G. Marques, and A. Ribeiro, "Invariance-preserving localized activation functions for graph neural networks," *arXiv preprint arXiv:1903.12575*, 2019.

[24] R. Levie, E. Isufi, and G. Kutyniok, "On the transferability of spectral graph filters," in *13th Int. Conf. Sampling Theory Applications*. Bordeaux, France: IEEE, 8-12 Jul. 2019, pp. 1–5.