Graph Neural Networks With Convolutional ARMA Filters

Filippo Maria Bianchi, Daniele Grattarola, *Student Member, IEEE*, Lorenzo Livi, *Member, IEEE*, and Cesare Alippi, *Fellow, IEEE*

Abstract—Popular graph neural networks implement convolution operations on graphs based on polynomial spectral filters. In this paper, we propose a novel graph convolutional layer inspired by the auto-regressive moving average (ARMA) filter that, compared to polynomial ones, provides a more flexible frequency response, is more robust to noise, and better captures the global graph structure. We propose a graph neural network implementation of the ARMA filter with a recursive and distributed formulation, obtaining a convolutional layer that is efficient to train, localized in the node space, and can be transferred to new graphs at test time. We perform a spectral analysis to study the filtering effect of the proposed ARMA layer and report experiments on four downstream tasks: semi-supervised node classification, graph signal classification, graph classification, and graph regression. Results show that the proposed ARMA layer brings significant improvements over graph neural networks based on polynomial filters.

Index Terms—Geometric deep learning, graph filters, graph neural networks, graph theory, graph signal processing

1 Introduction

Graph Neural Networks (GNNs) are a class of models lying at the intersection between deep learning and methods for structured data, which perform inference on discrete objects (nodes) by accounting for arbitrary relationships (edges) among them [1], [2]. A GNN combines node features within local neighborhoods on the graph to learn node representations that can be directly mapped into categorical labels or real values [3], [4], or combined to generate graph embeddings for graph classification and regression [5], [6], [7], [8], [9].

The focus of this work is on GNNs that implement a convolution in the spectral domain with a non-linear trainable filter [10], [11]. Such a filter selectively shrinks or amplifies the Fourier coefficients of the graph signal (an instance of the node features) and then maps the node features to a new space. To avoid the expensive spectral decomposition and projection in the frequency domain, state-of-the-art GNNs implement graph filters as low-order polynomials that are learned directly in the node domain [12], [13], [14]. Polynomial filters have a finite impulse response and perform a weighted moving average filtering of graph signals on local node neighborhoods [15], allowing for fast distributed implementations such as those based on Chebyshev polynomials and Lanczos iterations [12], [16], [17]. Polynomial filters have limited modeling capabilities [18] and, due to their smoothness, cannot model sharp changes in the frequency response [15]. Crucially, polynomials with high degree are necessary to reach high-order neighborhoods, but they tend to be more computationally expensive and, most importantly, overfit the training data making the model sensitive to changes in the graph signal or the underlying graph structure. A more versatile class of filters is the family of Auto-Regressive Moving Average filters (ARMA) [19], which offer a larger variety of frequency responses and can account for higher-order neighborhoods compared to polynomial filters with the same number of parameters.

In this paper, we address the limitations of existing graph convolutional layers inspired by polynomial filters and propose a novel GNN convolutional layer based on ARMA filters. Our ARMA layer implements a non-linear and trainable graph filter that generalizes the convolutional layers based on polynomial filters and provides the GNN with enhanced modeling capability, thanks to a flexible design of the filter's frequency response. The ARMA layer captures global graph structures with fewer parameters, overcoming the limitations of GNNs based on high-order polynomial filters.

ARMA filters are not localized in node space and require to compute a matrix inversion, which is intractable in the context of GNNs. To address this issue, the proposed ARMA layer relies on a recursive formulation, which leads to a fast and distributed implementation that exploits efficient sparse operations on tensors. The resulting filters are

- *Filippo Maria Bianchi is with the Department of Mathematics and Statistics, UiT The Arctic University of Norway, 9019 Tromsø, Norway and also with the NORCE Norwegian Research Centre, 5008 Bergen, Norway. E-mail: filippo.m.bianchi@uit.no.*
- *Daniele Grattarola is with the Faculty of Informatics, Università della Svizzera Italiana, 6900 Lugano, Switzerland. E-mail: daniele.grattarola@usi.ch.*
- *Lorenzo Livi is with the Department of Computer Science and Mathematics, University of Manitoba, Winnipeg, MB R3T 2N2, Canada and also with the Department of Computer Science, University of Exeter, EX4 4PY Exeter, U.K. E-mail: L.Livi@exeter.ac.uk.*
- *Cesare Alippi is with the Faculty of Informatics, Università della Svizzera Italiana, 6900 Lugano, Switzerland and also with the Department of Electronics, Information, and Bioengineering, Politecnico di Milano, 20133 Milano, Italy. E-mail: cesare.alippi@polimi.it.*

Manuscript received 2 Dec. 2019; revised 20 Jan. 2021; accepted 24 Jan. 2021. Date of publication 26 Jan. 2021; date of current version 3 June 2022. (Corresponding author: Filippo Maria Bianchi.) Recommended for acceptance by I. Rish. Digital Object Identifier no. 10.1109/TPAMI.2021.3054830

not learned in the Fourier space induced by a given Laplacian, but are localized in the node space and are independent of the underlying graph structure. This allows our GNN to handle graphs with unseen topologies during the test phase of inductive inference tasks.

The performance of the proposed ARMA layer is evaluated on semi-supervised node classification, graph signal classification, graph classification, and graph regression tasks. Results show that a GNN equipped with ARMA layers outperforms GNNs with polynomial filters in every downstream task.

2 Background: Graph Spectral Filtering

We assume a graph with M nodes to be characterized by a symmetric adjacency matrix $\mathbf{A} \in \mathbb{R}^{M \times M}$ and refer to *graph signal* $\mathbf{X} \in \mathbb{R}^{M \times F}$ as the instance of all features (vectors in $\mathbb{R}^F$) associated with the graph nodes. Let $\mathbf{L} = \mathbf{I}_M - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ be the symmetrically normalized Laplacian of the graph (where $\mathbf{D}$ is the degree matrix), with spectral decomposition $\mathbf{L} = \sum_{m=1}^{M} \lambda_m \mathbf{u}_m \mathbf{u}_m^T$. A graph filter is an operator that modifies the components of $\mathbf{X}$ on the eigenvectors basis of $\mathbf{L}$, according to a frequency response h acting on each eigenvalue λ_m. The filtered graph signal reads

$$\begin{aligned} \bar{\mathbf{X}} &= \sum_{m=1}^{M} h(\lambda_m)\mathbf{u}_m\mathbf{u}_m^T\mathbf{X} \\ &= \mathbf{U}\,\mathrm{diag}[h(\lambda_1), \dots, h(\lambda_M)]\,\mathbf{U}^T\mathbf{X}. \end{aligned} \tag{1}$$

This formulation inspired the seminal work of Bruna *et al.* [10] that implemented spectral graph convolutions in a neural network. Their GNN learns end-to-end the parameters of a filter implemented as $h = \mathbf{B}\mathbf{c}$, where $\mathbf{B} \in \mathbb{R}^{M \times K}$ is a cubic B-spline basis and $\mathbf{c} \in \mathbb{R}^K$ is a vector of control parameters. Such filters are not localized, since the full projection on the eigenvectors yields paths of infinite length and the filter accounts for interactions of each node with the whole graph, rather than those limited to the node neighborhood. Since this contrasts with the local design of classic convolutional filters, a follow-up work [11] introduced a parametrization of the spectral filters with smooth coefficients to achieve spatial localization. However, the main issue with the spectral filtering in Eq. (1) is computational complexity: not only the eigendecomposition of $\mathbf{L}$ is computationally expensive, but a double product with $\mathbf{U}$ is computed whenever the filter is applied. Notably, $\mathbf{U}$ in Eq. (1) is full even when $\mathbf{L}$ is sparse. Finally, since these spectral filters depend on a specific Laplacian spectrum, they cannot be transferred to graphs with another structure. For this reason, this spectral GNN cannot be used in downstream tasks such as graph classification or graph regression, where each datum is a graph with a different topology.

2.1 GNNs Based on Polynomial Filters and Limitations

The desired filter response $h(\lambda)$ can be approximated by a polynomial of order K

$$h_{\mathrm{POLY}}(\lambda) = \sum_{k=0}^{K} w_k \lambda^k, \tag{2}$$

which performs a weighted moving average of the graph signal [15]. These filters overcome important limitations of the spectral formulation, as they avoid the eigendecomposition and their parameters are independent of the Laplacian spectrum [20]. Polynomial filters are localized in the node space, since the output at each node in the filtered signal is a linear combination of the nodes with their K-hop neighborhoods.

The order of the polynomial K is assumed to be small and independent of the number M of nodes in the graph.

To express polynomial filters in the node space, we first recall that the kth power of any diagonalizable matrix, such as the Laplacian, can be computed by taking the power of its eigenvalues, i.e., $\mathbf{L}^k = \mathbf{U}\,\mathrm{diag}[\lambda_1^k, \dots, \lambda_M^k]\,\mathbf{U}^T$. It follows that the filtering operation becomes

$$\begin{aligned} \bar{\mathbf{X}} &= \big(w_0\mathbf{I} + w_1\mathbf{L} + w_2\mathbf{L}^2 + \cdots + w_K\mathbf{L}^K\big)\mathbf{X} \\ &= \sum_{k=0}^{K} w_k \mathbf{L}^k \mathbf{X}. \end{aligned} \tag{3}$$

Eqs. (2) and (3) represent a generic polynomial filter. Among the existing classes of polynomials, Chebyshev polynomials are often used in signal processing as they attenuate unwanted oscillations around the cut-off frequencies [21], which, in our case, are the eigenvalues of the Laplacian. Fast localized GNN filters can approximate the desired filter response by means of the Chebyshev expansion $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ [12], resulting in convolutional layers that perform the filtering operation

$$\bar{\mathbf{X}} = \sigma\left(\sum_{k=0}^{K-1} T_k(\hat{\mathbf{L}})\mathbf{X}\mathbf{W}_k\right), \tag{4}$$

where $\hat{\mathbf{L}} = 2\mathbf{L}/\lambda_{\max} - \mathbf{I}$, $\sigma(\cdot)$ is a non-linear activation (e.g., a sigmoid or a ReLU function), and $\mathbf{W}_k \in \mathbb{R}^{F_{\mathrm{in}} \times F_{\mathrm{out}}}$ are the k trainable weight matrices that map the node features from $\mathbb{R}^{F_{\mathrm{in}}}$ to $\mathbb{R}^{F_{\mathrm{out}}}$.

The output of a k-degree polynomial filter is a linear combination of the input within each vertex's k-hop neighborhood. Since the input beyond the k-hop neighborhood has no impact on the output of the filtering operation, to capture larger structures on the graph it is necessary to adopt high-degree polynomials. However, high-degree polynomials have poor interpolatory and extrapolatory performance since they overfit the known graph frequencies, i.e., the eigenvalues of the Laplacian. This hampers the GNN's generalization capability as it becomes sensitive to noise and small changes in the graph topology. Moreover, evaluating a polynomial with a high degree is computationally expensive both during training and inference [18]. Finally, since polynomials are very smooth, they cannot model filter responses with sharp changes.

A particular first-order polynomial filter has been proposed by [13] for semi-supervised node classification. In their GNN model, called Graph Convolutional Network (GCN), the convolutional layer is a simplified version of a Chebyshev filter, obtained from Eq. (4) by considering $K = 1$ and by setting $\mathbf{W} = \mathbf{W}_0 = -\mathbf{W}_1$

$$\bar{\mathbf{X}} = \sigma\big(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}\big). \tag{5}$$

Additionally, $\hat{\mathbf{L}}$ is replaced by $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$, with $\tilde{\mathbf{A}} = \mathbf{A} + \gamma\mathbf{I}_M$ (usually, $\gamma = 1$). The modified adjacency matrix $\hat{\mathbf{A}}$ contains self-loops that compensate for the removal of the term of order 0 in the polynomial, by ensuring that a node is part of its first-order neighborhood and that its features are preserved (to some extent) after convolution. Higher-order neighborhoods can be reached by stacking multiple GCN layers. On one hand, GCNs reduce overfitting and the heavy computational load of Chebyshev filters with high-order polynomials. On the other hand, since each GCN layer performs a Laplacian smoothing, after few convolutions the node features becomes too smoothed over the graph [22] and the initial node features are lost.

3 Rational Filters for Graph Signals

An ARMA filter can approximate well any desired filter response $h(\lambda)$ thanks to a rational design that, compared to polynomial filters, can model a larger variety of filter shapes [15]. The filter response of an ARMA filter of order K, denoted in the following as $\mathrm{ARMA_K}$, reads

$$h_{\mathrm{ARMA}_K}(\lambda) = \frac{\sum_{k=0}^{K-1} p_k \lambda^k}{1 + \sum_{k=1}^{K} q_k \lambda^k}, \tag{6}$$

which translates to the following filtering relation in the node space

$$\bar{\mathbf{X}} = \left(\mathbf{I} + \sum_{k=1}^{K} q_k \mathbf{L}^k\right)^{-1} \left(\sum_{k=0}^{K-1} p_k \mathbf{L}^k\right)\mathbf{X}. \tag{7}$$

Different orders ($\leq K$) of the numerator and denominator in Eq. (6) are trivially obtained by setting some coefficients to 0. Notice that by setting $q_k = 0$, for every k, one recovers a polynomial filter, which is considered as the MA term of the model. The inclusion of the additional AR term encoded by these coefficients makes the ARMA model robust to noise and allows to capture longer dynamics on the graph since $\bar{\mathbf{x}}$ depends, in turn, on several steps of propagation of the node features. This is the key to capturing longer dependencies and more global structures on the graph, compared to a polynomial filter with the same degree.

The matrix inversion in Eq. (7) is slow to compute and yields a dense matrix that prevents us from using sparse multiplications to implement the GNN. In this paper, we follow a straightforward approach to avoid computing the inverse, which can be easily extended to a neural network implementation. Specifically, we approximate the effect of an $\mathrm{ARMA_1}$ filter by iterating, until convergence, the first-order recursion

$$\bar{\mathbf{X}}^{(t+1)} = a\mathbf{M}\bar{\mathbf{X}}^{(t)} + b\mathbf{X}, \tag{8}$$

where

$$\mathbf{M} = \frac{1}{2}(\lambda_{\max} - \lambda_{\min})\mathbf{I} - \mathbf{L}. \tag{9}$$

The recursion in Eq. (8) is adopted in graph signal processing to apply a low-pass filter on a graph signal [18], [23], but it is also equivalent to the recurrent update used in Label Propagation [24] and Personalized Page Rank [25] to propagate information on a graph by means of a random walk with a restart probability.

Following the derivation in [18, Theorems 1 and 2], we analyze the frequency response of an $\mathrm{ARMA_1}$ filter from the convergence of Eq. (8)

$$\bar{\mathbf{X}} = \lim_{t\to\infty}\left[(a\mathbf{M})^t\bar{\mathbf{X}}^{(0)} + b\sum_{i=0}^{t}(a\mathbf{M})^i\mathbf{X}\right]. \tag{10}$$

The eigenvectors of $\mathbf{M}$ and $\mathbf{L}$ are the same, while the eigenvalues are related as follows: $\mu_m = (\lambda_{\max} - \lambda_{\min})/2 - \lambda_m$, where μ_m and λ_m represent the mth eigenvalue of $\mathbf{M}$ and $\mathbf{L}$, respectively. Since $\mu_m \in [-1, 1]$, for $|a| < 1$ the first term of Eq. (10), $(a\mathbf{M})^t$, goes to zero when $t \to \infty$, regardless of the initial point $\bar{\mathbf{X}}^{(0)}$. The second term, $b\sum_{i=0}^{t}(a\mathbf{M})^i$, is a geometric series that converges to the matrix $b(\mathbf{I} - a\mathbf{M})^{-1}$, with eigenvalues $b/(1 - a\mu_m)$. It follows that the frequency response of the $\mathrm{ARMA_1}$ filter is

$$h_{\mathrm{ARMA}_1}(\mu_m) = \frac{b}{1 - a\mu_m}. \tag{11}$$

By summing K $\mathrm{ARMA_1}$ filters, it is possible to recover the analytical form of the $\mathrm{ARMA_K}$ filter in Eq. (7). The resulting filtering operation is

$$\bar{\mathbf{X}} = \sum_{k=1}^{K}\sum_{m=1}^{M}\frac{b_k}{1 - a_k\mu_m}\mathbf{u}_m\mathbf{u}_m^T\mathbf{X}, \tag{12}$$

with

$$h_{\mathrm{ARMA}_K}(\mu_m) = \sum_{k=1}^{K}\frac{b_k}{1 - a_k\mu_m}. \tag{13}$$

4 The ARMA Neural Network Layer

In graph signal processing, the filter coefficients a and b in Eq. (8) are optimized with linear regression to reproduce a desired filter response $h^*(\lambda)$, which must be provided *a priori* by the designer [18]. Here, we consider a machine learning approach that does not require to specify the target response $h^*(\lambda)$ but in which the parameters are learned end-to-end from the data by optimizing a task-dependent loss function. Importantly, we also introduce non-linearities to enhance the representation capability of the filter response that can be learned.

The proposed neural network formulation of the $\mathrm{ARMA_1}$ filter implements the recursive update of Eq. (8) with a *Graph Convolutional Skip* (GCS) layer, defined as

$$\bar{\mathbf{X}}^{(t+1)} = \sigma\Big(\tilde{\mathbf{L}}\bar{\mathbf{X}}^{(t)}\mathbf{W} + \mathbf{X}\mathbf{V}\Big), \tag{14}$$

where $\sigma(\cdot)$ is a non-linearity such as ReLU, sigmoid, or hyperbolic tangent (*tanh*), $\mathbf{X}$ are the initial node features, and $\mathbf{W} \in \mathbb{R}^{F_{\mathrm{out}} \times F_{\mathrm{out}}}$ and $\mathbf{V} \in \mathbb{R}^{F_{\mathrm{in}} \times F_{\mathrm{out}}}$ are trainable parameters. The modified Laplacian matrix $\tilde{\mathbf{L}}$ is defined by setting $\lambda_{\min} = 0$ and $\lambda_{\max} = 2$ in Eq. (9), which gives $\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{L}$. This is a reasonable simplification since the spectrum of $\mathbf{L}$ lies in [0,2] and the trainable parameters $\mathbf{W}$ and $\mathbf{V}$ can compensate for the small offset introduced. The unfolded recursion in

Eq. (14) corresponds to a stack of GCS layers that share the same parameters.

Each GCS layer is localized in the node space, as it performs a filtering operation that depends on local exchanges among neighboring nodes and, through the skip connection, also on the initial node features $\mathbf{X}$. The computational complexity of the GCS layer is linear in the number of edges (both in time and space) since the layer can be efficiently implemented as a sparse multiplication between $\tilde{\mathbf{L}}$ and $\bar{\mathbf{X}}^{(t)}$.

The neural network formulation of an ARMA_1 filter is obtained by iterating Eq. (14) until convergence, i.e., until $\|\bar{\mathbf{X}}^{(T+1)} - \bar{\mathbf{X}}^{(T)}\| < \epsilon$, where ϵ is a small positive constant and T is the convergence time. The convergence of the update in Eq. (14), which draws a connection to the original recursive formulation of the ARMA_1 filter, is guaranteed by Theorem 1.

Theorem 1. *It is sufficient that $\|\mathbf{W}\|_2 < 1$ and that $\sigma(\cdot)$ is a non-expansive map for Eq. (14) to converge to a unique fixed point, regardless of the initial state $\bar{\mathbf{X}}^{(0)}$.*

Proof. Let $\bar{\mathbf{X}}_a^{(0)}$ and $\bar{\mathbf{X}}_b^{(0)}$ be two different initial states and $\|\mathbf{W}\|_2 < 1$. After applying Eq. (14) for $t+1$ steps, we obtain states $\mathbf{X}_a^{(t+1)}$ and $\bar{\mathbf{X}}_b^{(t+1)}$. If the non-linearity $\sigma(\cdot)$ is a non-expansive map, such as the ReLU function, the following inequality holds:

$$\begin{aligned}
&\left\|\bar{\mathbf{X}}_a^{(t+1)} - \bar{\mathbf{X}}_b^{(t+1)}\right\|_2 \\
&= \left\|\sigma\left(\tilde{\mathbf{L}}\bar{\mathbf{X}}_a^{(t)}\mathbf{W} + \mathbf{X}\mathbf{V}\right) - \sigma\left(\tilde{\mathbf{L}}\bar{\mathbf{X}}_b^{(t)}\mathbf{W} + \mathbf{X}\mathbf{V}\right)\right\|_2 \\
&\leq \left\|\tilde{\mathbf{L}}\bar{\mathbf{X}}_a^{(t)}\mathbf{W} + \mathbf{X}\mathbf{V} - \tilde{\mathbf{L}}\bar{\mathbf{X}}_b^{(t)}\mathbf{W} - \mathbf{X}\mathbf{V}\right\|_2 \\
&= \left\|\tilde{\mathbf{L}}\bar{\mathbf{X}}_a^{(t)}\mathbf{W} - \tilde{\mathbf{L}}\bar{\mathbf{X}}_b^{(t)}\mathbf{W}\right\|_2 \\
&\leq \|\tilde{\mathbf{L}}\|_2 \|\mathbf{W}\|_2 \left\|\bar{\mathbf{X}}_a^{(t)} - \bar{\mathbf{X}}_b^{(t)}\right\|_2.
\end{aligned} \tag{15}$$

If the non-linearity $\sigma(\cdot)$ is also a squashing function (e.g., sigmoid or *tanh*), then the first inequality in (15) is strict.

Since the largest singular value of $\tilde{\mathbf{L}}$ is ≤ 1 by definition, it follows that $\|\tilde{\mathbf{L}}\|_2 \|\mathbf{W}\|_2 < 1$ and, therefore, (15) implies that Eq. (14) is a contraction mapping. The convergence to a unique fixed point and, thus, the inconsequentiality of the initial state, follow by the Banach fixed-point theorem [26]. □

From Theorem 1 it follows that it is possible to choose an arbitrary $\epsilon > 0$ for which

$$\exists T_\epsilon < \infty \text{ s.t. } \left\|\bar{\mathbf{X}}^{(t+1)} - \bar{\mathbf{X}}^{(t)}\right\|_2 \leq \epsilon, \forall t \geq T_\epsilon.$$

Therefore, we can easily implement a stopping criterion for the iteration, which is met in finite time.

Similar to the formulation of the ARMA filter in Eq. (12), the output of the ARMA_K convolutional layer is obtained by combining the outputs of K parallel stacks of GCS layers.

4.1 Implementation

Each GCS stack k may require a different and possibly high number of iterations T_k to converge, depending on the value of the node features $\mathbf{X}$ and the weight matrices $\mathbf{W}_k$ and $\mathbf{V}_k$. This makes the implementation of the neural network cumbersome, because the computational graph is dynamic and changes every time the weight matrices are updated with gradient descent during training. Moreover, to train the parameters with backpropagation through time the neural network must be unfolded many times if T_k is large, introducing a high computational cost and the vanishing gradient issue [27].

One solution is to follow the approach of Reservoir Computing, where the weight matrices $\mathbf{W}_k$ and $\mathbf{V}_k$ in each stack are randomly initialized and left untrained [28], [29]. We notice that the random weights initialization guarantees that the K GCS stacks implement different filtering operations. To compensate for the lack of training, high-dimensional features are exploited to generate rich latent representations that disentangle the factors of variations in the data [30]. However, randomized architectures with high-dimensional feature spaces are memory inefficient and computationally expensive at inference time.

A second approach, considered in this work, is to drop the requirement of convergence altogether and fix the number of iterations to a constant value T, so that $T_k = T$ in each GCS stack k. In this way, we obtain a GNN that is easy to implement, fast to train and evaluate, and not affected by stability issues. Notably, the constraint $\|\mathbf{W}\|_2 < 1$ of Theorem 1 can be relaxed by adding to the loss function an L_2 weight decay regularization term.

Even by stacking a small number T of GCS layers, we expect the GNN to learn a large variety of node representations thanks to the non-linearity and the trainable parameters [31]. As non-linearity we adopt the ReLU function that, compared to the squashing non-linearities, improves training efficiency by facilitating the gradient flow [32].

Given the limited number of iterations, the initial state $\bar{\mathbf{X}}^{(0)}$ now influences the final representation $\bar{\mathbf{X}}^{(T)}$. A natural choice is to initialize the state with $\bar{\mathbf{X}}^{(0)} = \mathbf{0} \in \mathbb{R}^{M \times F_\mathrm{out}}$ or with a linear transformation of the node features $\bar{\mathbf{X}}^{(0)} = \mathbf{X}\mathbf{W}^{(0)}$, where $\mathbf{W}^{(0)} \in \mathbb{R}^{F_\mathrm{in} \times F_\mathrm{out}}$ replaces $\mathbf{W}$ in the first layer of the stack. We adopted the latter initialization so that the node features are propagated also by the first GCS layer. We also note that it is possible to set $\mathbf{W}^{(0)} = \mathbf{V}$ to reduce the number of trainable parameters.

The output of the ARMA_K convolutional layer is computed as

$$\bar{\mathbf{X}} = \frac{1}{K}\sum_{k=1}^{K} \bar{\mathbf{X}}_k^{(T)}, \tag{16}$$

where $\bar{\mathbf{X}}_k^{(T)}$ is the output of the last GCS layer in the kth stack. Fig. 1 depicts a scheme of the proposed ARMA graph convolutional layer.

To encourage each GCS stack to learn a filtering operation with a response different from the other stacks, we apply stochastic dropout to the skip connections $\mathbf{X}\mathbf{V}_k$ in each GCS layer. This leads to learning a heterogeneous set of features that, when combined to form the output of the ARMA_K layer, yield powerful and expressive node

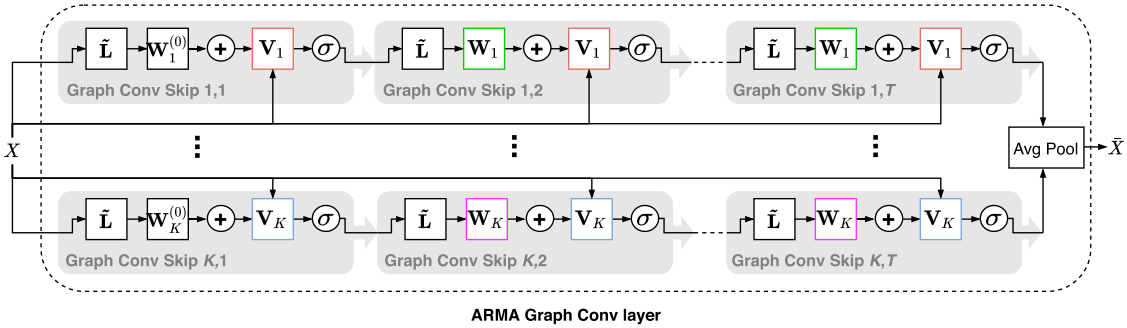


Fig. 1. The ARMA convolutional layer. Same color indicates that the weights are shared.

representations. We notice that the parameter sharing in each layer of the GCS stack endows the GNN with a strong regularization that helps to prevent overfitting and greatly reduces the model complexity, in terms of the number of trainable parameters. Finally, since the GCS stacks are independent of each other, the computation of an ARMA layer can be distributed across multiple processing units.

4.2 Properties and Relationship With Other Approaches

Contrarily to filters defined directly in the spectral domain [10], ARMA filters do not explicitly depend on the eigenvectors and the eigenvalues of $\mathbf{L}$, making them robust to perturbations in the underlying graph structure. For this reason, as formally proven for generic rational filters [33], the proposed ARMA filters are transferable, i.e., they can be applied to graphs with different topologies not seen during training.

The skip connections in our architecture allow stacking many GCS layers without the risk of over-smoothing the node features. Due to the weight sharing, the ARMA architecture has similarities with the recurrent neural networks with residual connections used to process sequential data [34].

Similarly to GNNs operating directly in the node domain [3], [35], each GCS layer computes the filtered signal $\bar{\mathbf{x}}_i^{(t+1)}$ at vertex i as a combination of signals $\mathbf{x}_j^{(t)}$ in its 1-hop neighborhood, $j \in \mathcal{N}(i)$. Such a commutative aggregation solves the problem of undefined vertex ordering and varying neighborhood sizes, making the proposed operator permutation equivariant.

The skip connections in ARMA inject in each GCS layer t of the stack the initial node features $\mathbf{X}$. This is different from a skip connection that either takes the output of the previous layer $\mathbf{X}^{(t-1)}$ as input [8], [36], or connects all the layers in a GNN stack directly to the output [37].

The ARMA layer can naturally deal with a time-varying topology and graph signals [38], [39] by replacing the constant term $\mathbf{X}$ in Eq. (14) with a time-dependent input $\mathbf{X}^{(t)}$.

Finally, we discuss the relationship between the proposed ARMA GNN and CayleyNets [40], a GNN architecture that also approximates the effect of a rational filter. Specifically, the filtering operation of a Cayley polynomial in the node space is

$$\bar{\mathbf{X}} = w_0\mathbf{X} + 2\mathrm{Re}\left\{\sum_{k=1}^{K} w_k(\mathbf{L}+i\mathbf{I})^k(\mathbf{L}-i\mathbf{I})^{-k}\right\}\mathbf{X}. \tag{17}$$

To approximate the matrix inversion in Eq. (17) with a sequence of differentiable operations, CayleyNets adopt a fixed number T of Jacobi iterations. In practice, the Jacobi iterations approximate each term $(\mathbf{L}+i\mathbf{I})(\mathbf{L}-i\mathbf{I})^{-1}$ as a polynomial of order T with fixed coefficients. Therefore, the resulting filtering operation performed by a CayleyNet assumes the form

$$\bar{\mathbf{X}} \approx \sigma\left(w_0\mathbf{X} + 2\mathrm{Re}\left\{\sum_{k=1}^{K} w_k\left(\sum_{t=1}^{T}\hat{\mathbf{L}}^t\right)^k\right\}\mathbf{X}\right), \tag{18}$$

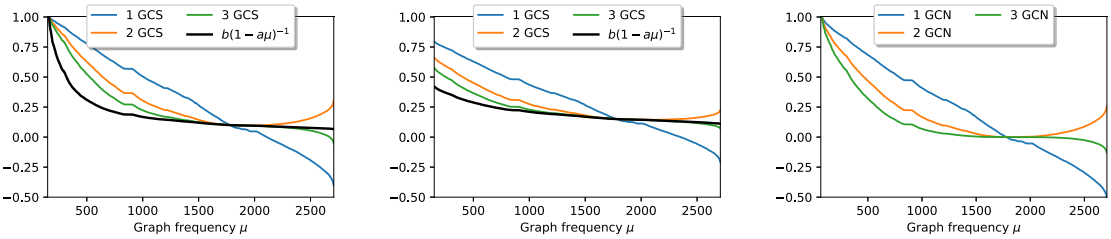
where $\hat{\mathbf{L}}$ is an operator with the same sparsity pattern of $\mathbf{L}$. We note that Eqs. (17) and (18) slightly simplify the original formulation presented by Levie *et al.* [40], but allow us to better understand what type of operation is actually performed by the CayleyNet. Specifically, Eq. (18) implements a polynomial filter of order KT, such as the one in Eq. (3).

For this reason, CayleyNets share strong similarities with the Chebyshev filter in Eq. (4), as it uses a (high-order) polynomial to propagate the node features on the graph for KT hops before applying the non-linearity. On the other hand, each of the K parallel stacks in the proposed ARMA layer propagates the current node representations $\bar{\mathbf{X}}^{(t)}$ only for 1 hop and combines them with the node features $\mathbf{X}$ before applying the non-linearity.

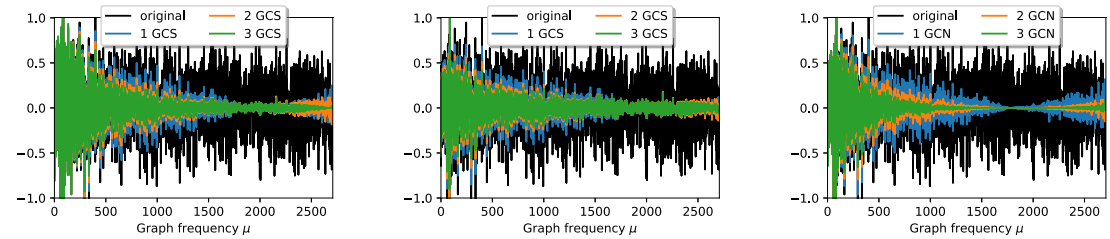
5 Spectral Analysis of the ARMA Layer

In this section we show how the proposed ARMA layer can implement filtering operations with a large variety of frequency responses. The filter response of the ARMA filter derived in Section 3 cannot be exploited to analyze our GNN formulation, due to the presence of non-linearities. Therefore, we first recall that a filter changes the components of a graph signal $\mathbf{X}$ on the eigenbase induced by $\mathbf{L}$ (which is the same as the one induced by $\tilde{\mathbf{L}}$, according to Sylvester's theorem). By referring to Eq. (1), $\mathbf{X}$ is first projected on the eigenspace of $\mathbf{L}$ by $\mathbf{U}^T$, then the filter $h(\lambda_m)$ changes the value of the component of $\mathbf{X}$ on each eigenvector $\mathbf{u}_m$, finally $\mathbf{U}^T$ maps back to the node space. By left-multiplying $\mathbf{U}^T$ in Eq. (1) we obtain

$$\begin{aligned}\mathbf{U}^T\bar{\mathbf{X}} &= \mathrm{diag}[h(\lambda_1),\ldots,h(\lambda_M)]\,\mathbf{U}^T\mathbf{X},\\ \sum_{m=1}^{M}\mathbf{u}_m^T\bar{\mathbf{X}} &= \sum_{m=1}^{M}h(\lambda_m)\mathbf{u}_m^T\mathbf{X}.\end{aligned} \tag{19}$$



(a) $\tilde{h}$ in the 1st GCS stack (b) $\tilde{h}$ in the 2nd GCS stack (c) $\tilde{h}$ in a GCN stack



(d) Comp. of $\mathbf{X}$ and $\bar{\mathbf{X}}$ in the 1st stack (e) Comp. of $\mathbf{X}$ and $\bar{\mathbf{X}}$ in the 2nd stack (f) Comp. of $\mathbf{X}$ and $\bar{\mathbf{X}}$ in a GCN stack

Fig. 2. In (a, b), the empirical filter responses of two GCS stacks for $T = 1, 2, 3$; the black lines indicate the analytical response of an ARMA_1 filter with similar parameters. In (c), the empirical response of a GCN with $T = 1, 2, 3$ layers. In (d, e), the original components of the input graph signal $\mathbf{X}$ (in black), and the components of the graph signal $\bar{\mathbf{X}}$ processed by two GCS stacks for $T = 1, 2, 3$ (in color). In (f), the components of $\bar{\mathbf{X}}$ processed by a GCN with $T = 1, 2, 3$ layers.

When $\bar{\mathbf{X}}$ is the output of the ARMA layer, the term $\mathbf{U}^T\bar{\mathbf{X}}$ defines how the original components, $\mathbf{U}^T\mathbf{X}$, are changed by the GNN. Therefore, we can compute numerically the unknown filter response of the ARMA layer as the ratio between $\mathbf{U}^T\bar{\mathbf{X}}$ and $\mathbf{U}^T\mathbf{X}$. We define the *empirical filter response* $\tilde{h}$ as

$$\tilde{h}_m = \frac{F_{\text{in}}}{F_{\text{out}}} \frac{\sum_{f=1}^{F_{\text{out}}} \mathbf{u}_m^T \bar{\mathbf{x}}_f}{\sum_{f=1}^{F_{\text{in}}} \mathbf{u}_m^T \mathbf{x}_f}, \tag{20}$$

where $\bar{\mathbf{x}}_f$ is column f of the output $\bar{\mathbf{X}}_k$, $\mathbf{x}_f$ is column f of the graph signal $\mathbf{X}$, and $\mathbf{u}_m$ is an eigenvector of $\mathbf{L}$.

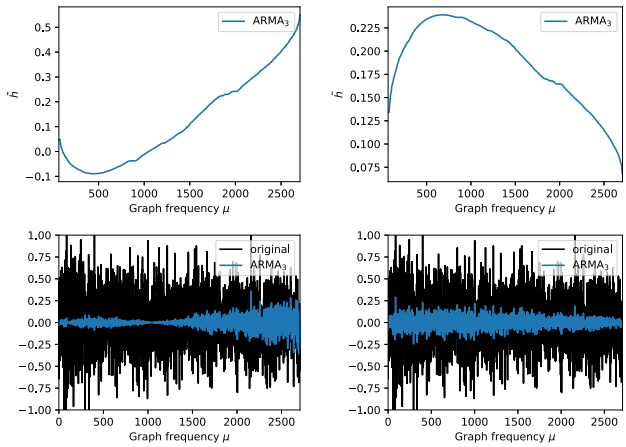
The empirical filter response allows us to analyze the type of filtering implemented by an ARMA layer. We start by comparing the recursion in Eq. (8), which converges to an ARMA_1 filter with response $\{h_{\mathrm{ARMA}_1}(\mu_m)\}_{m=1}^M$ according to Eq. (11), with the empirical response $\{\tilde{h}_{m,k}\}_{m=1}^M$ of the kth GCS stack. To facilitate the interpretation of the results, we set the number of output features of the GCS layer to $F_{\text{out}} = 1$ by letting $\mathbf{W} = a$ and $\mathbf{V} = b\mathbf{1}_{F_{\text{in}}}$ in Eq. (14). Notice that we are keeping the notation consistent with Eq. (8), where a and b are the parameters of the ARMA_1 filter. In the following we consider the graph and the node features from the Cora citation network. We remark that the examples in this section are not related to the results on the semi-supervised node classification task presented in Section 6 and any other dataset could have been used instead of Cora.

Figs. 2a and 2b show the empirical responses $\tilde{h}_1$ and $\tilde{h}_2$ of two different GCS stacks, when varying the number of layers T. As T increases, $\tilde{h}_1$ and $\tilde{h}_2$ become more similar to the analytical responses of the ARMA_1 filters, depicted as a black line in the two figures. This supports our claim that $\tilde{h}$ can estimate the unknown response of the GNN filtering operation.

Figs. 2d and 2e show how the two GCS stacks modify the components of $\mathbf{X}$ on the Fourier basis. In particular, we depict in black the components $\mathbf{u}_m^T\mathbf{X}$, $m = 1, \ldots, M$ associated with each graph frequency μ_m. In colors, we depict the components $\mathbf{u}_m^T\bar{\mathbf{X}}$, which show how much the GCS stacks filter the components associated with each frequency. The responses and the signal components in Figs. 2a and 2d are obtained for $a = 0.99$ and $b = 0.1$, while in Figs. 2b and 2e for $a = 0.7$ and $b = 0.15$.

In Fig. 2c, we show the empirical response resulting from a stack of GCNs. As also highlighted in recent work [41], [42], the filtering obtained by stacking one or more GCNs has the undesired effect of symmetrically amplifying the lowest and also the highest frequencies of the spectrum. This is due to the GCN filter response, which is $(1-\lambda)^T$ in the linear case and can assume negative values when T is odd. The effect is mitigated by summing $\gamma\mathbf{I}_M$ to the adjacency matrix, which adds self-loops with weight γ and shrinks the spectral domain of the graph filter. For high values of γ, the GCN acts more as a low-pass filter that prevents high-frequency oscillations. This is due to the self-loops that limit the spread of information across the graph and the communication between neighbors. However, even after adding $\gamma\mathbf{I}_M$, GCN cuts almost completely the medium frequencies and then amplifies again the higher ones, as shown in Fig. 2f.

A stack of GCNs lacks flexibility in implementing different filtering operations, as the only degree of freedom to modify a GCN's response consists of *manually* tuning the hyperparameter γ to shrink the spectrum. On the other hand, different GCS stacks can generate heterogeneous filter responses, depending on the value of the trainable parameters in each stack. This is what provides powerful modeling capability to the proposed ARMA layer, which can learn a large variety of filter responses that selectively shrink or amplify the Fourier components of the graph by combining K GCS stacks.



(a) High-pass filter (b) Band-pass filter

Fig. 3. While each GCS stack behaves as a low-pass filter, an ARMA layer with $K = 3$ can implement filters of different shapes. The ARMA layer in (a) implements a high-pass filtering operation that dampens low frequencies. The ARMA layer in (b) implements a band-pass filtering operation that mostly allows medium frequencies.

Similarly to an ARMA_1 filter, each GCS stack behaves as a low-pass filter that gradually dampens the Fourier components as their frequency increases. However, we recall that high-pass and band-pass filters can be obtained as a linear combination of low-pass filters [43]. To show this behavior in practice, in Fig. 3 we report the empirical filter responses and modified Fourier components obtained with two different ARMA_K filters, for $K = 3$.

6 Experiments

We consider four downstream tasks: node classification, graph signal classification, graph classification, and graph regression. Our experiments focus on comparing the proposed ARMA layer with GNN layers based on polynomial filters, namely Chebyshev [12] and GCN [13], and CayleyNets [40] that, like ARMA, are based on rational spectral filters. As additional baselines, we also include Graph Attention Networks (GAT) [44], GraphSAGE [8], and Graph Isomorphism Networks (GIN) [45]. The comparison with these methods helps to frame the proposed ARMA GNN within the current state of the art. We also mention that other GNNs with graph convolutional filters related to our method have appeared while our work was under review [46], [47], [48], [49].

To ensure a fair and meaningful evaluation, we compare the performance obtained with a fixed GNN architecture, where we only change only the graph convolutional layers. In particular, we fixed the GNN capacity (number of hidden units), used the same splits in each dataset, and the same training and evaluation procedures. Finally, in all experiments we used the same polynomial order K for polynomial/rational filters, or a stack of K layers for GCN, GAT, GIN, and GraphSAGE layers. The details of every dataset considered in the experiments and the optimal hyperparameters for each model are deferred to Section 7.

Public implementations of the ARMA layer are available in the open-source GNN libraries Spektral [50] (TensorFlow/Keras) and PyTorch Geometric [51] (PyTorch).

TABLE 1
Node Classification Accuracy

Method	Cora	Citeseer	Pubmed	PPI
GAT	83.1 ± 0.6	70.9 ± 0.6	78.5 ± 0.3	81.3 ± 0.1
GraphSAGE	73.7 ± 1.8	65.9 ± 0.9	78.5 ± 0.6	70.0 ± 0.0
GIN	75.1 ± 1.7	63.1 ± 2.0	77.1 ± 0.7	78.1 ± 2.6
GCN	81.5 ± 0.4	70.1 ± 0.7	$\mathbf{79.0 \pm 0.5}$	80.8 ± 0.1
Chebyshev	79.5 ± 1.2	70.1 ± 0.8	74.4 ± 1.1	86.4 ± 0.1
CayleyNet	81.2 ± 1.2	67.1 ± 2.4	75.6 ± 3.6	84.9 ± 1.2
ARMA	$\mathbf{83.4 \pm 0.6}$	$\mathbf{72.5 \pm 0.4}$	78.9 ± 0.3	$\mathbf{90.5 \pm 0.3}$

6.1 Node Classification

First, we consider transductive node classification on three citation networks: Cora, Citeseer, and Pubmed. The input is a single graph described by an adjacency matrix $\mathbf{A} \in \mathbb{R}^{M \times M}$, the node features $\mathbf{X} \in \mathbb{R}^{M \times F_\mathrm{in}}$, and the labels $\mathbf{y}_l \in \mathbb{R}^{M_l}$ of a subset of nodes $M_l \subset M$. The targets are the labels $\mathbf{y}_u \in \mathbb{R}^{M_u}$ of the unlabelled nodes. The node features are sparse bag-of-words vectors representing text documents. The binary undirected edges in $\mathbf{A}$ indicate citation links between documents. The models are trained using 20 labels per document class ($\mathbf{y}_l$) and the performance is evaluated as classification accuracy on $\mathbf{y}_u$.

Second, we perform inductive node classification on the protein-protein interaction (PPI) network dataset. The dataset consists of 20 graphs used for training, 2 for validation, and 2 for testing. Contrarily to the transductive setting, the testing graphs (and the associated node features) are not observed during training. Additionally, each node can belong to more than one class (multi-label classification).

We use a 2-layers GNN with 16 hidden units for the citation networks and 64 units for PPI. In the citation networks high dropout rates and L_2-norm regularization are exploited to prevent overfitting. Table 1 reports the classification accuracy obtained by a GNN equipped with different graph convolutional layers.

Transductive node classification is a semi-supervised task that demands using a simple model with strong regularization to avoid overfitting on the few labels available. This is the key of GCN's success when compared to more complex filters, such as Chebyshev. Thanks to its flexible formulation, the proposed ARMA layer can implement the right degree of complexity and performs well on each task. On the other hand, since the PPI dataset is larger and more labels are available during training, less regularization is required and the more complex models are advantaged. This is reflected by the better performance achieved by Chebyshev filters and CayleyNets, compared to GCN. On PPI, ARMA significantly outperforms every other model, due to its powerful modeling capability that allows learning filter responses with different shapes. Since each layer in GAT, GraphSAGE, and GIN combines the features of a node only with those from its 1st order neighborhood, similarly to a GCN, these architectures need to stack more layers to reach higher-order neighborhoods and suffer from the same over-smoothing issue.

We notice that the optimal depth T of the ARMA layer reported in Table 6 is low in every dataset. We argue that a reason is the small average shortest path in the graphs (see

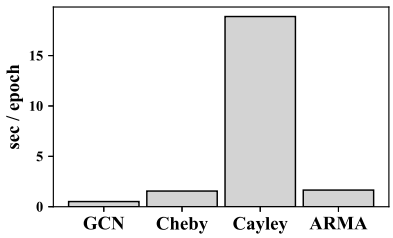


Fig. 4. Training times on the PPI dataset, obtained with an Nvidia Titan Xp GPU.

Table 5). Indeed, most nodes in the graphs can be reached with only a few propagation steps, which is not surprising since many real networks are small-world [52].

Fig. 4 shows the training times of the GNN model configured with GCN, Chebyshev, CayleyNet, and ARMA layers. The ARMA layer exploits sparse operations that are linear in the number of nodes in $\mathbf{L}$ and can be trained in a time comparable to a Chebyshev filter. On the other hand, CayleyNet is slower than other methods, due to the complex formulation based on the Jacobi iterations that results in a high order polynomial.

6.2 Graph Signal Classification

In this task, N different graph signals $\mathbf{X}_n \in \mathbb{R}^{M \times F_{\text{in}}}, n = 1, \ldots, N$, defined on the same graph with adjacency matrix $\mathbf{A} \in \mathbb{R}^{M \times M}$, must be mapped to labels $y_1, \ldots, y_N$. We perform these experiments following the same setting of [12] for the MNIST and 20news datasets.

MNIST. To emulate a classic CNNs operating on a regular 2D grid, an 8-NN graph is defined on the 784 pixels of the MNIST images. To determine if a vertex v_j belongs to the neighborhood $\mathcal{N}(v_i)$ of a vertex v_i, we compute the euclidean distance between the 2D coordinates of pixels (vertices) i and j. The elements in $\mathbf{A}$ are

$$a_{ij} = \begin{cases} 1 & \text{if } v_j \in \mathcal{N}(v_i); \\ 0 & \text{otherwise.} \end{cases} \tag{21}$$

Each graph signal is a vectorized image $\mathbf{x} \in \mathbb{R}^{784}$. The architecture is a GNN(32)-P(4)-GNN(64)-P(4)-FC(512)-$\text{FC}_{\text{Softmax}}$(10), where GNN($n$) indicates a GNN layer with n filters, P(s) a pooling operation with stride s, and FC(u) a fully connected layer with u units (the last FC has a Softmax activation). Pooling is implemented by a hierarchical spectral clustering algorithm (GRACLUS) [53], which maps the graph signal $\bar{\mathbf{x}}^{(l)}$ at layer l into a new node feature space $\mathbf{x}^{(l+1)} \in \mathbb{R}^{M_{l+1} \times F_{l+1}}$.

Table 2 reports the results obtained by using GCN, Chebyshev, CayleyNet, or ARMA. The results are averaged over 10 runs and show that ARMA achieves a slightly higher, and almost perfect, accuracy compared to Chebyshev and CayleyNet, while the performance of GCN is significantly lower. Similarly to the PPI experiment, the larger amount of data allows more powerful architectures to be trained more precisely and to achieve better performance compared to the simpler GCN.

20news. The dataset consists of 18,846 documents divided into 20 classes. Each graph signal is a document represented by a bag-of-words of the 10^4 most frequent words in the corpus, embedded via Word2vec [54]. The underlying graph of 10^4 nodes is defined by a 16-NN adjacency matrix built as in Eq. (21), with the difference that the vertex neighborhoods are computed from the euclidean distance between the embeddings vectors rather than the pixel coordinates. We report results obtained with a single convolutional layer (GCN, Chebyshev, CayleyNet, or ARMA), followed by global average pooling and Softmax. As in [12], we use 32 channels for Chebyshev. Instead, for GCN, CayleyNet, and ARMA, better results are obtained with only 16 filters. The classification accuracy reported in Table 2 shows that ARMA significantly outperforms every other model also on this dataset.

TABLE 2
Graph Signal Classification Accuracy

GNN layer	MNIST	20news
GCN	98.48 ± 0.2	65.45 ± 0.2
Chebyshev	99.14 ± 0.1	68.24 ± 0.2
CayleyNet	99.18 ± 0.1	68.84 ± 0.3
ARMA	$\mathbf{99.20 \pm 0.1}$	$\mathbf{70.02 \pm 0.1}$

For this experiment we used a particular configuration of the ARMA layer with $K = 1$ and $T = 1$ (see Table 12), which is equivalent to a GCN with a skip connection. The skip connection allows to weight differently the contribution of the original node feature, compared to the features of the neighbors. It is important to notice that, contrary to other downstream tasks, the 20news graph is generated from the similarity of word embeddings. Such an artificial graph always links an embedding vector to its first 16 neighbors. We argue that, for some words, the links might be not very relevant and using a skip connection allows weighting them less.

Similarly to the node classification datasets, the average shortest path in the 20news graph is low (see Table 11). On the other hand, the MNIST graph has a much higher diameter, due to its regular structure with very localized connectivity. This could explain why the optimal depth T of the ARMA layer is larger for MNIST than for any other task (see Table 12), as several steps are necessary to mix the node features on the graph.

6.3 Graph Classification

In this task, the ith datum is a graph represented by a pair $\{\mathbf{A}_i, \mathbf{X}_i\}, i = 1, \ldots N$, where $\mathbf{A}_i \in \mathbb{R}^{M_i \times M_i}$ is an adjacency matrix with M_i nodes, and $\mathbf{X}_i \in \mathbb{R}^{M_i \times F}$ are the node features. Each sample must be classified with a label y_i. We test the models on five different datasets. We use node degree, clustering coefficients, and node labels as additional node features. For each dataset we adopt a fixed network architecture GNN-GNN-GNN-AvgPool-$\text{FC}_{\text{Softmax}}$, where AvgPool indicates a global average pooling layer. We compute the model performance with nested 10-fold cross-validation repeated for 10 runs, using 10 percent of the training set in each fold for early stopping. Table 3 reports the average accuracy and includes the results obtained also by using GAT, GraphSAGE, and GIN as convolutional layers. The GNN equipped with the proposed ARMA layer achieves the highest mean accuracy compared the polynomial filters (Chebyshev and GCN). Compared to CayleyNets, which are also based on a rational filter implementation, ARMA

TABLE 3
Graph Classification Accuracy

Method	Enzymes	Proteins	D&D	MUTAG	BHard
GAT	51.7±4.3	72.3±3.1	70.9±4.0	87.3±5.3	30.1±0.7
GraphSAGE	60.3±7.1	70.2±3.9	73.6±4.1	85.7±4.7	71.8±1.0
GIN	45.7±7.7	71.4±4.5	71.2±5.4	86.3±9.1	72.1±1.1
GCN	53.0±5.3	71.0±2.7	74.7±3.8	85.7±6.6	71.9±1.2
Chebyshev	57.9±2.6	72.1±3.5	73.7±3.7	82.6±5.2	71.3±1.2
CayleyNet	43.1±10.7	65.6±5.7	70.3±11.6	87.8±10.0	70.7±2.4
ARMA	**60.6**±7.2	**73.7**±3.4	**77.6**±2.7	**91.5**±4.2	**74.1**±0.5

TABLE 4
Graph Regression Mean Squared Error

Property	GCN	Chebyshev	CayleyNet	ARMA
mu	0.445±0.007	0.433±0.003	0.442±0.009	**0.394**±0.005
alpha	0.141±0.016	0.171±0.008	0.118±0.005	**0.098**±0.005
HOMO	0.371±0.030	0.391±0.012	0.336±0.007	**0.326**±0.010
LUMO	0.584±0.051	0.528±0.005	0.679±0.148	**0.508**±0.011
gap	0.650±0.070	0.565±0.015	0.758±0.106	**0.552**±0.013
R2	0.132±0.005	0.294±0.022	0.185±0.043	**0.119**±0.019
ZPVE	0.349±0.022	0.358±0.001	0.555±0.174	**0.338**±0.001
U0_atom	0.064±0.003	0.126±0.017	1.493±1.414	**0.053**±0.004
Cv	0.192±0.012	0.215±0.010	0.184±0.009	**0.163**±0.007

achieves not only a higher mean accuracy but also a lower standard deviation. These empirical results indicate that our implementation is robust and confirm the transferability of the proposed ARMA layer, discussed in Section 4.

6.4 Graph Regression

This task is similar to graph classification, with the difference that the target output y_i is now a real value, rather than a discrete class label. We consider the QM9 chemical database [55], which contains more than 130,000 molecular graphs. The nodes represent heavy atoms and the undirected edges the atomic bonds between them. Nodes have discrete attributes indicating one of four possible elements. The regression task consists of predicting a given chemical property of a molecule given its graph representation. As for graph classification, we evaluate the performance on the 80-10-10 train-validation-test splits of the nested 10-folds. The network architecture adopted to predict each property is GNN(64)-AvgPool-FC(128). We report in Table 4 the mean squared error (MSE) averaged over 10 independent runs, relative to the prediction of 9 molecular properties. It can be noticed that each model achieves a very low standard deviation. One reason is the very large amount of training data, which allows the GNN to learn a configuration that generalizes well. Contrarily to the previous tasks, here there is not a clear winner among GCN, Chebyshev, and CayleyNet, since each of them performs better than the others on some tasks. On the other hand, ARMA always achieves the lowest MSE in predicting each molecular property.

7 Experimental Details

7.1 Node Classification

Table 5 reports for each node classification dataset the number of nodes, number of edges, number of node attributes (size of the node feature vectors), average shortest path of the graph (Avg. SP), and the number of classes that each node can be assigned to. The three citation networks (Cora, Citeseer, and Pubmed) are taken from https://github.com/tkipf/gcn/raw/master/gcn/data/, while the PPI dataset is taken from http://snap.stanford.edu/graphsage/.

TABLE 5
Summary of the Node Classification Datasets

Dataset	Nodes	Edges	Node attr.	Avg. SP	Node classes
Cora	2708	5429	1433	5.87±1.52	7 (single label)
Citeseer	3327	9228	3703	6.31±2.00	6 (single label)
Pubmed	19717	88651	500	6.34±1.22	3 (single label)
PPI	56944	818716	50	2.76±0.56	121 (multi-label)

TABLE 6
Hyperparameters for Node Classification

Dataset	L_2 reg.	p_{drop}	lr	GCN	Cheby.	Cayley		ARMA	
				L	K	K	T	K	T
Cora	5e-4	0.75	0.01	1	2	1	5	2	1
Citeseer	5e-4	0.75	0.01	1	3	1	5	3	1
Pubmed	5e-4	0.25	0.01	1	3	2	5	1	1
PPI	0.0	0.25	0.01	2	3	3	5	3	2

TABLE 7
Summary of the Graph Regression Dataset

Samples	Avg. nodes	Avg. edges	Node attr.
133,885	8.79	27.61	1

Table 6 describes the optimal hyperparameters used in GCN, Chebyshev, CayleyNet, and ARMA for each node classification dataset. For all GNN, we report the L_2 regularization weight, the learning rate (lr) and dropout probability (p_{drop}). For GCN, we report the number of stacked graph convolutions (L). For Chebyshev, we report the polynomial order (K). For CayleyNet, we report the polynomial order (K) and the number of Jacobi iterations (T). For ARMA, we report the number of GCS stacks (K) and the stacks' depth (T). Additionally, we configured the MLP in GIN with 2 hidden layers and trained the parameter ϵ, while for GraphSAGE we used the *max* aggregator, to differentiate more its behavior from GCN and GIN. Finally, GAT is configured with 8 attention heads and the same number of layers L as GCN.

Each model is trained for 2,000 epochs with early stopping (based on the validation accuracy) at 50 epochs. We used full-batch training, i.e., in each epoch the weights are updated one time, according to a single batch that includes all the training data.

7.2 Graph Regression

The QM9 dataset used for graph regression is available at http://quantum-machine.org/datasets/, and its statistics are reported in Table 7.

The hyperparameters are reported in Table 8. Only for this task, CayleyNets use only 3 Jacobi iterations, since with

TABLE 8
Hyperparameters for Graph Classification and Graph Regression

Dataset	GCN	Cheby.	Cayley		ARMA		
	L	K	K	T	p_{drop}	K	T
QM9	3	3	3	3	0.75	3	3

TABLE 9
Summary of the Graph Classification Datasets

Dataset	Samples	Classes	Avg. nodes	Avg. edges	Node attr.	Node labels
Bench-hard	1,800	3	148.32	572.32	–	yes
Enzymes	600	6	32.63	62.14	18	no
Proteins	1,113	2	39.06	72.82	1	no
D&D	1,178	2	284.32	715.66	–	yes
MUTAG	188	2	17.93	19.79	–	yes

TABLE 10
Hyperparameters for Graph Classification and Graph Regression

Dataset	GCN	Cheby	Cayley		ARMA		
	L	K	K	T	p_{drop}	K	T
Bench-hard	2	2	2	10	0.4	1	2
Enzymes	2	2	2	10	0.6	2	2
Proteins	4	4	4	10	0.6	4	4
D&D	4	4	4	10	0.0	4	4
MUTAG	4	4	4	10	0.0	4	4

more iterations we experienced numerical errors and the loss quickly diverged. All models are trained for 1,000 epochs with early stopping at 50 epochs, using the Adam optimizer with learning rate 10^{-3}. We used batch size 64 and no L_2 regularization.

7.3 Graph Classification

The datasets Enzymes, Proteins, D&D, and MUTAG are taken from the Benchmark Data Sets for Graph Kernels https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets, while the dataset Bench-hard is taken from https://github.com/FilippoMB/Benchmark_dataset_for_graph_classification. The statistics of each graph classification dataset are summarized in Table 9.

For all methods, we use a fixed architecture composed of three GNN layers, each with 32 output units, ReLU activations, and L_2 regularization with a factor of 10^{-4}. All models are trained to convergence with Adam, using a learning rate of 10^{-3}, batch size of 32, and patience of 50 epochs. We summarize in Table 10 the hyperparameters used for ARMA, Chebyshev, and CayleyNets on the different datasets.

7.4 Graph Signal Classification

To generate the datasets we used the code available at github.com/mdeff/cnn_graph. The models are trained for 20 epochs on each dataset. We used batches of size 32 for MNIST and 128 for 20news. In the 20news dataset, the word embeddedings have size 200.

TABLE 11
Summary of the Graph Signal Classification Datasets

Dataset	Nodes	Edges	Avg. SP	Class	Train	Val	Test
MNIST	784	5,928	12.36±5.45	10	55,000	5,000	10,000
20news	10,000	249,944	4.21±0.94	20	10,168	7,071	7,071

TABLE 12
Hyperparameters for Graph Signal Classification

Dataset	L_2 reg.	lr	p_{drop}	GCN	Cheby.	Cayley		ARMA	
				L	K	K	T	K	T
MNIST	5e-4	1e-3	0.5	3	25	12	11	5	10
20news	1e-3	1e-3	0.7	1	5	5	10	1	1

Table 11 reports, for each graph signal classification dataset: the number of nodes and edges of the graph and the average shortest path (Avg. SP), the number of classes each graph signal can be assigned to, and the number of graph signals in the training, validation, and test set. Table 12 reports the optimal hyperparameters configuration for each model.

8 Conclusion

This paper introduced the ARMA layer, a novel graph convolutional layer based on a rational graph filter. The ARMA layer models more expressive filter responses and can account for larger neighborhoods compared to GNN layers based on polynomial filters of the same order. Our ARMA layer consists of parallel stacks of recurrent operations, which approximate a graph filter with an arbitrary order K by means of efficient sparse tensor multiplications. We reported a spectral analysis of our neural network implementation, which provides valuable insights into the proposed method and shows that our ARMA layer can implement a large variety of filter responses. The experiments showed that the proposed ARMA layer outperforms existing GNN architectures, including those based on polynomial filters and other more complex models, on a large variety of graph machine learning tasks.

Acknowledgments

The work of Daniele Grattarola was supported by the Swiss National Science Foundation's Grant 200021/172671. The authors gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. The work of Lorenzo Livi was supported by the Canada Research Chairs program. The authors would also like to thank the anonymous reviewers for their precious suggestions, which helped us to improve our work.

References

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

[2] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," 2018, *arXiv: 1806.01261*.

[3] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[4] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *Proc. Int. Conf. Learn. Representations*, 2019.

[5] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 701–710.

[6] D. K. Duvenaud *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 2224–2232.

[7] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *Proc. 33rd Int. Conf. Int. Conf. Mach. Learn.*, 2016, pp. 40–48.

[8] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[9] D. Bacciu, F. Errica, and A. Micheli, "Contextual graph Markov model: A deep and generative approach to graph processing," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 294–303.

[10] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2014, *arXiv:1312.6203*.

[11] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," 2015, *arXiv:1506.05163*.

[12] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.

[13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2016.

[14] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *Proc. NIPS Workshop Bayesian Deep Learn.*, 2016.

[15] N. Tremblay, P. Goncalves, and P. Borgnat, "Design of graph filters and filterbanks," in *Cooperative and Graph Signal Processing*. Amsterdam, The Netherlands: Elsevier, 2018, pp. 299–324.

[16] A. Susnjara, N. Perraudin, D. Kressner, and P. Vandergheynst, "MATHICSE Technical Report: Accelerated filtering on graphs using Lanczos method," 2015, *arXiv:1509.04537*.

[17] R. Liao, Z. Zhao, R. Urtasun, and R. Zemel, "LanczosNet: Multi-scale deep graph convolutional networks," in *Proc. Int. Conf. Learn. Representations*, 2019.

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

[19] S. K. Narang, A. Gadde, and A. Ortega, "Signal processing techniques for interpolation in graph structured data," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2013, pp. 5445–5449.

[20] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11782

[21] D. I. Shuman, P. Vandergheynst, and P. Frossard, "Chebyshev polynomial approximation for distributed signal processing," in *Proc. Int. Conf. Distrib. Comput. Sensor Syst. Workshops*, 2011, pp. 1–8.

[22] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 3538–3545.

[23] A. Loukas, A. Simonetto, and G. Leus, "Distributed autoregressive moving average graph filters," *IEEE Signal Process. Lett.*, vol. 22, no. 11, pp. 1931–1935, Nov. 2015.

[24] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2004, pp. 321–328.

[25] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the web," *Stanford InfoLab*, 1999. [Online]. Available: http://ilpubs.stanford.edu:8090/422/

[26] K. Goebel and W. Kirk, "A fixed point theorem for asymptotically nonexpansive mappings," *Proc. Amer. Math. Soc.*, vol. 35, no. 1, pp. 171–174, 1972.

[27] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, *Recurrent Neural Networks for Short-Term Load Forecasting: An Overview and Comparative Analysis*. Berlin, Germany: Springer, 2017.

[28] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Rev.*, vol. 3, no. 3, pp. 127–149, 2009.

[29] C. Gallicchio and A. Micheli, "Fast and deep graph neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 3898–3905.

[30] P. Tiňo, "Dynamical systems as temporal feature spaces," *J. Mach. Learn. Res.*, vol. 21, no. 44, pp. 1–42, 2020.

[31] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2847–2854.

[32] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2016.

[33] R. Levie, I. Elvin, and K. Gitta, "On the transferability of spectral graph filters," in *Proc. 13th Int. conf. Sampling Theory Appl.*, pp. 1–5, 2019, doi: 10.1109/SampTA45681.2019.9030932.

[34] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*.

[35] C. Gallicchio and A. Micheli, "Graph echo state networks," in *Proc. Int. Joint Conf. Neural Netw.*, 2010, pp. 1–8.

[36] T. Pham, T. Tran, D. Phung, and S. Venkatesh, "Column networks for collective classification," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2485–2491.

[37] F. Wu, T. Zhang, A. H. D. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger, "Representation learning on graphs with jumping knowledge networks," in *Proc. 35th Int. Conf. Int. Conf. Mach. Learn.*, 2018, pp. 5453–5462.

[38] P. Holme, "Modern temporal network theory: A colloquium," *Eur. Phys. J. B*, vol. 88, no. 9, 2015, Art. no. 234.

[39] D. Grattarola, D. Zambon, C. Alippi, and L. Livi, "Change detection in graph streams by learning graph embeddings on constant-curvature manifolds," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 6, pp. 1856–1869, Jun. 2020.

[40] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "CayleyNets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, Jan. 2019.

[41] F. Wu, T. Zhang, A. H. D. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Proc. 36th Int. Conf. Int. Conf. Mach. Learn.*, 2019, pp. 6861–6871.

[42] H. Nt and T. Maehara, "Revisiting graph neural networks: All we have is low-pass filters," 2019, *arXiv: 1905.09550*.

[43] A. V. Oppenheim, J. R. Buck, and R. W. Schafer, *Discrete-Time Signal Processing*. vol. 2. Upper Saddle River, NJ, USA: Prentice Hall, 2001.

[44] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018, *arXiv: 1710.10903*.

[45] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km

[46] V. N. Ioannidis, S. Chen, and G. B. Giannakis, "Pruned graph scattering transforms," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rJeg7TEYwB

[47] F. Gama, A. Ribeiro, and J. Bruna, "Stability of graph scattering transforms," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8038–8048.

[48] D. Zou and G. Lerman, "Graph convolutional neural networks via scattering," *Appl. Comput. Harmon. Anal.*, vol. 49, no. 3, pp. 1046–1074, 2020.

[49] F. Gao, G. Wolf, and M. Hirn, "Geometric scattering for graph data analysis," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2122–2131.

[50] D. Grattarola and C. Alippi, "Graph neural networks in TensorFlow and Keras with Spektral," *IEEE Computational Intelligence Mag.*, 2021, vol. 16, no. 1, pp. 99–106, doi: 10.1109/MCI.2020.3039072.

[51] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," 2019, *arXiv: 1903.02428*.

[52] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[53] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 11, pp. 1944–1957, Nov. 2007.

[54] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. Int. Conf. Learn. Representations*, 2013.

[55] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Sci. Data*, vol. 1, 2014, Art. no. 140022.

Filippo Maria Bianchi received the PhD degree from the Department of Information Engineering, Electronics, and Telecommunications, Sapienza University of Rome, Rome, Italy, in 2016. Currently, he is an associate professor at the Department of Mathematics and Statistics, UiT the Arctic University of Norway and a research scientist with NORCE. His research interests include machine learning, complex networks, and dynamical systems.

Daniele Grattarola (Student Member, IEEE) rececived the MSc (*magna cum laude*) degree in computer science and engineering from Politecnico di Milano, Milan, Italy, in 2017. He is currently working toward the PhD degree from the Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland. His research interests include graph neural networks and their application to systems that change over time.

Lorenzo Livi (Member, IEEE) received the PhD degree from the Department of Information Engineering, Electronics, and Telecommunications, Sapienza University of Rome, Rome, Italy, in 2014. He has been with the ICT industry during his studies. From January 2014 until April 2016, he was a post doctoral fellow with Ryerson University, Toronto, Canada. From May 2016 until September 2016, he was a postdoctoral fellow with the Politecnico di Milano, Italy and Universita' della Svizzera Italiana, Lugano, Switzerland. Currently, he is an assistant professor jointly appointed with the Departments of Computer Science and Mathematics, University of Manitoba, Canada. He is also a lecturer (assistant professor) in data science with the Department of Computer Science, University of Exeter, U.K. In November 2018, he was awarded the prestigious Tier 2 Canada Research Chair in Complex Data. He is an associate editor of the *IEEE Transactions on Neural Networks and Learning Systems* and the *Applied Soft Computing* (Elsevier). His research interests include machine learning, time series analysis and complex dynamical systems, with focused applications in systems biology and computational neuroscience.

Cesare Alippi (Fellow, IEEE) is currently a professor at the Politecnico di Milano, Milano, Italy and Universita' della Svizzera italiana, Lugano, Switzerland. He is also a visiting professor at the University of Guangzhou, China and consultant professor with the Northwestern Polytechnic in Xi'an, China. He has been a visiting researcher/professor with UCL (UK), MIT (USA), ESPCI (F), CASIA (RC), A*STAR (SIN), UKobe (JP). He is a member of the Administrative Committee of the IEEE Computational Intelligence Society, Board of Governors member of the International Neural Network Society, Board of Directors member of the European Neural Network Society, past vice-president education of the IEEE Computational Intelligence Society, past associate editor of the *IEEE Transactions on Emerging Topics in Computational Intelligence*, the *IEEE Computational Intelligence Magazine*, the *IEEE Transactions on Instrumentation and Measurements,* the *IEEE Transactions on Neural Networks (and Learning Systems)*. He received the 2018 IEEE CIS Outstanding Computational Intelligence Magazine Award, the 2016 Gabor award from the International Neural Networks Society and the IEEE Computational Intelligence Society Outstanding Transactions on Neural Networks and Learning Systems Paper Award, in 2013 the IBM Faculty award, in 2004 the IEEE Instrumentation and Measurement Society Young Engineer Award. His research interests include adaptation and learning in non-stationary environments, graph learning and Intelligence for embedded, IoT, and cyber-physical systems.