Graph-time convolutional neural networks

Isufi, Elvin; Mazzola, Gabriele

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

GRAPH-TIME CONVOLUTIONAL NEURAL NETWORKS

Elvin Isufi and Gabriele Mazzola

ABSTRACT

Spatiotemporal data can be represented as a process over a graph, which captures their spatial relationships either explicitly or implicitly. How to leverage such a structure for learning representations is one of the key challenges when working with graphs. In this paper, we represent the spatiotemporal relationships through product graphs and develop a first principle graph-time convolutional neural network (GTCNN). The GTCNN is a compositional architecture with each layer comprising a graph-time convolutional module, a graph-time pooling module, and a nonlinearity. We develop a graph-time convolutional filter by following the shift-and-sum principles of the convolutional operator to learn higher-level features over the product graph. The product graph itself is parametric so that we can learn also the spatiotemporal coupling from data. We develop a zero-pad pooling that preserves the spatial graph (the prior about the data) while reducing the number of active nodes and the parameters. Experimental results with synthetic and real data corroborate the different components and compare with baseline and state-of-the-art solutions.

Index Terms— Graph signal processing; graph neural networks; graph-time neural networks; spatiotemporal learning.

1. INTRODUCTION

Multivariate temporal data provide unique challenges to the learning algorithms because of their intrinsic spatiotemporal dependencies. These dependencies can be captured by a graph either explicitly such as in sensor or social networks or implicitly such as in recommender systems. This graph represents the spatial coupling between data, which translates into a tantamount graph-temporal coupling. The learning algorithm should, therefore, be equipped with effective biases to exploit this structure for learning spatiotemporal representations. Building on recent advances in processing and learning over graphs [1, 2], different solutions have been proposed to learn from spatiotemporal data [3]. The key to learning is the algorithm's ability to embed spatiotemporal relations into its inner-working mechanisms.

Spatiotemporal graph-based models can be divided into hybrid and fused. *Hybrid models* combine learning algorithms developed separately for the graph domain and the temporal domain. They use graph neural networks to extract higher-level spatial features and process the latter with a temporal RNN, CNN, or variants of them. The works in [4–6] use a graph convolutional neural network (GCNN) per timestamp followed by an LSTM. Instead, authors in [7] first use a temporal RNN and then a GCNN. The works in [8–10] prefer temporal CNNs since convolutions are easier to train and have fewer parameters. *Fused models* force the graph structure into conventional spatiotemporal solutions and provide a single strategy to jointly capture the spatiotemporal relationships. They substitute the parameter matrices in these models with graph convolutional filters, which are at the core of GCNNs [11]. The work in [12] proposes a graph-based VARMA model to learn spatiotemporal representations. The works [13, 14] consider the RNN family, whereas [15] discusses also graph-based gating [16]. The work in [17] builds replicas of the spatial graph, connects nodes at time t with their replicas at time $t - 1$, and learns over this larger graph.

Hybrid models have the advantage that their spatial and temporal blocks are modular and can be implemented efficiently. But it re-

mains unclear how to best interleave these blocks for learning from spatiotemporal relationships. Instead, fused models capture naturally these relationships as they have graph-time dependent inner-working mechanisms. One effective way to represent spatiotemporal relationships is to model the time as a graph (e.g., directed line); the evolution as a *time-varying* signal over this graph [18]; and the overall data as a *time-invariant* signal over the product graph between the spatial and the temporal graph [19]. This solution has resulted useful to develop a graph-time Fourier representation [20], autoregressive models [12,21], and signal interpolation algorithms [22]. However, despite the success of product graphs to capture spatiotemporal relations, learning solutions over product graphs remain little explored.

In this paper, we develop a graph-time convolutional neural network (GTCNN) that implements a compositional learning model, where each layer performs convolutions over the product graph. Our contribution is threefold:

C.1. We develop a graph-time convolutional module build from the first principles of the convolution operator [18, 23] over a parametric product graph [24]. Working with first principles and parametric product graphs allows learning the spatiotemporal coupling from data and generalizes [17], which can be seen as an order one convolutional filter over the Cartesian product.

C.2. We propose a recursive implementation of the graph-time convolutional module, which avoids working with large product graphs. This recursive implementation has a linear cost in the product graph dimensions and a constant number of parameters.

C.3. We develop a graph-time pooling module based on a zero-padding strategy [25] to reduce the number of active nodes; hence parameters. The advantage of zero-pad pooling is that it preserves the original spatial graph structure in the deeper layers and does not resort to coarsening or clustering techniques.

Numerical results with synthetic and three real datasets corroborate the effectiveness of the proposed approach.

2. SIGNALS OVER PRODUCT GRAPHS

Consider an $N \times 1$ multivariate signal $\mathbf{x}_t$ collected over T time instances in matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$, such as sensor recordings in a sensor network. Signals in $\mathbf{X}$ have *spatiotemporal* relations, which if fully-exploited serve as a powerful inductive bias to learn representations [26]. When signal $\mathbf{x}_t$ has an (hidden) underlying structure, we can represent its spatial relations through a *spatial* graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of N nodes in set $\mathcal{V} = \{1, \ldots, N\}$ and $|\mathcal{E}|$ edges in set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Signal $\mathbf{x}_t = [x_{t1}, \ldots, x_{tN}]^\top$ are a collection of values x_{ti} residing on node i at time t. Likewise, we can capture the temporal relations by viewing each row $\mathbf{x}^i = [x_{1i}, \ldots, x_{Ti}]^\top$ of $\mathbf{X}$ as a graph signal over the nodes of a *temporal* graph $\mathcal{G}_T = (\mathcal{V}_T, \mathcal{E}_T)$ of T nodes $\mathcal{V}_T = \{1, \ldots, T\}$ and $|\mathcal{E}_T|$ edges $\mathcal{E}_T = (t, t')$. Set $\mathcal{E}_T$ contains an edge if signals at time instances t and t' are related. Examples for $\mathcal{G}_T$ are the directed line graph that assumes signal $\mathbf{x}_t$ depends only on the former instance $\mathbf{x}_{t-1}$, the cyclic graph that accounts for periodicity, or any other graph that encodes the temporal dependencies in $\mathbf{X}$ [1]. We will represent graphs $\mathcal{G}$ and $\mathcal{G}_T$ through their respective graph shift operator matrices $\mathbf{S} \in \mathbb{R}^{N \times N}$ and $\mathbf{S}_T \in \mathbb{R}^{T \times T}$; e.g., adjacency, Laplacians [1, 18].

Given graphs $\mathcal{G}$ and $\mathcal{G}_T$, we can capture the spatiotemporal relations in $\mathbf{X}$ through the product graph $\mathcal{G}_\diamond = \mathcal{G}_T \diamond \mathcal{G} = (\mathcal{V}_\diamond, \mathcal{E}_\diamond)$, where

The authors are with the Intelligent Systems Department, Delft University of Technology, Delft, The Netherlands. Corresponding author: E. Isufi. e-mails: e.isufi-1@tudelft.nl; gabriele.m1995@gmail.com
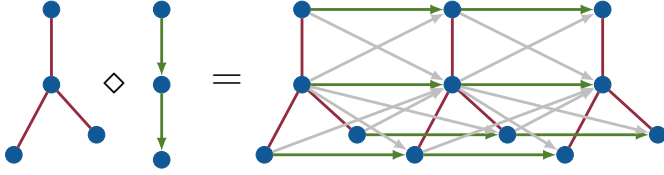
Kronecker: $\mathbf{S}_\otimes = \mathbf{S}_T \otimes \mathbf{S}_\otimes$; $|\mathcal{E}_\otimes| = |\mathcal{E}_T||\mathcal{E}|$
Cartesian: $\mathbf{S}_\times = \mathbf{S}_T \otimes \mathbf{I}_N + \mathbf{I}_T \otimes \mathbf{S}$; $|\mathcal{E}_\times| = T|\mathcal{E}| + N|\mathcal{E}_T|$
Strong: $\mathbf{S}_\boxtimes = \mathbf{S}_\otimes + \mathbf{S}_\times$ $|\mathcal{E}_\boxtimes| = |\mathcal{E}_\otimes| + |\mathcal{E}_\times|$

Fig. 1: Product Graphs. Kronecker product: $\mathcal{G}_\otimes = \mathcal{G}_T \otimes \mathcal{G}$ has the grey edges. Cartesian product: $\mathcal{G}_\times = \mathcal{G}_T \times \mathcal{G}$ has only the red and green edges. Strong product: $\mathcal{G}_\boxtimes = \mathcal{G}_T \boxtimes \mathcal{G}$ has all edges. Parametric product: $\mathcal{G}_\diamond = \mathcal{G}_T \diamond \mathcal{G}$ has all edges of the strong and self loops if all $s_{ij} \neq 0$ in (1).

the vertex set $\mathcal{V}_\diamond = \mathcal{V}_T \times \mathcal{V}$ is the Kronecker product between $\mathcal{V}_T$ and $\mathcal{V}$ while the edge set depends on the product [19–22, 27]. Typical product graphs include the Kronecker, Cartesian, and strong product, which are particular cases of a parametric product graph with the shift operator

$$\mathbf{S}_\diamond = \sum_{i=0}^1 \sum_{j=0}^1 s_{ij}\left(\mathbf{S}_T^i \otimes \mathbf{S}^j\right) \qquad (1)$$

where $\{s_{ij}\}$ are scalars and "$\otimes$" the Kronecker product [24]; see Fig. 1. The parametric product graph captures the spatiotemporal coupling with the four scalars s_{ij}. If all s_{ij}s are non-zero, the parametric product graph has $|\mathcal{E}_\diamond| = |\mathcal{E}_\boxtimes| + NT$ edges, which are NT mode edges than the strong product because of self-loops ($s_{00} \neq 0$).

Column-vectorizing $\mathbf{X}$ yields a product graph signal $\mathbf{x}_\diamond = \mathrm{vec}(\mathbf{X}) \in \mathbb{R}^{NT}$ in which node $i_\diamond \in \mathcal{V}_\diamond$ represents the space-time location (i,t) with value x_{ti}, i.e., the $i_\diamond$th entry of $\mathbf{x}_\diamond$. Our goal is to exploit the coupling *product graph signal–product graph* to learn spatiotemporal representations in a form akin to temporal or graph convolutional neural networks [11, 23].

3. GRAPH-TIME CONVOLUTIONAL NEURAL NETWORKS

A graph-time neural network is a compositional architecture of L layers each having a graph-time convolutional module, a graph-time pooling module, and a nonlinearity. At layer ℓ, we have as input a collection of $F_{\ell-1}$ graph signal features $\mathbf{x}_{\diamond,\ell-1}^g$ for $g = 1, \ldots, F_{\ell-1}$. These features are the output of the previous layer and can be seen as a collection of signals over the vertices of a product graph $\mathcal{G}_{\diamond,\ell-1}$. Each input feature $\mathbf{x}_{\diamond,\ell-1}^g$ is processed in parallel by a bank of F_ℓ graph-time filters $\mathbf{H}_\ell^{fg}(\mathbf{S}_{\diamond,\ell-1})$ to yield the aggregated (filtered) features

$$\mathbf{u}_{\diamond,\ell}^{fg} = \mathbf{H}_\ell^{fg}(\mathbf{S}_{\diamond,\ell-1})\mathbf{x}_{\diamond,\ell-1}^g \quad \text{for} \quad (f;g) = 1, \ldots, (F_\ell; F_{\ell-1}). \quad (2)$$

Aggregated features $\mathbf{u}_{\diamond,\ell}^{fg}$ obtained from a common input $\mathbf{x}_{\diamond,\ell-1}^g$ are summed to form the higher-level linear features of layer ℓ

$$\mathbf{u}_{\diamond,\ell}^f = \sum_{g=1}^{F_{\ell-1}} \mathbf{u}_{\diamond,\ell}^{fg} = \sum_{g=1}^{F_{\ell-1}} \mathbf{H}_\ell^{fg}(\mathbf{S}_{\diamond,\ell-1})\mathbf{x}_{\diamond,\ell-1}^g \quad \text{for} \quad f = 1, \ldots, F_\ell \quad (3)$$

which are again a collection of F_ℓ product graph signals. The linear features $\mathbf{u}_{\diamond,\ell}^f$ are input in parallel to the graph-time pooling module $\rho_\ell(\cdot; \mathcal{G}_{\diamond,\ell-1})$ to obtain the pooled features

$$\mathbf{z}_{\diamond,\ell}^f = \rho_\ell\left(\mathbf{u}_{\diamond,\ell}^f; \mathcal{G}_{\diamond,\ell-1}\right) \quad \text{for} \quad f = 1, \ldots, F_\ell \quad (4)$$

which are vectors of dimensions $N_\ell T_\ell$ with $N_\ell < N_{\ell-1}$ and $T_\ell < T_{\ell-1}$ being the number of spatial and temporal nodes at layer ℓ, respectively. Function $\rho_\ell(\cdot; \mathcal{G}_{\diamond,\ell-1})$ signifies pooling is performed over the product graph $\mathcal{G}_{\diamond,\ell-1}$. The pooled features are passed to a nonlinearity

$\sigma(\cdot; \mathcal{G}_{\diamond,\ell})$ to produce a collection of F_ℓ higher-level nonlinear features $\mathbf{x}_{\diamond,\ell}^f$ which constitute the output of layer ℓ,

$$\mathbf{x}_{\diamond,\ell}^f = \sigma_\ell\left[\mathbf{z}_\ell^f; \mathcal{G}_{\diamond,\ell}\right] \quad \text{for} \quad f = 1, \ldots, F_\ell \quad (5)$$

where $\sigma[\cdot; \mathcal{G}_{\diamond,\ell}]$ signifies the nonlinear function is performed over graph $\mathcal{G}_{\diamond,\ell}$ obtained from pooling.

In the last layer $\ell = L$, we assume there is a single feature signal, which we consider the output of the graph-time neural network. We write this output compactly as

$$\mathbf{\Phi}(\mathbf{x}_\diamond; \mathbf{S}_\diamond; \mathcal{H}) = \sigma_L\left[\rho_L\left(\sum_{g=1}^{F_{L-1}} \mathbf{H}_L^g(\mathbf{S}_{\diamond,L-1})\mathbf{x}_{\diamond,L-1}^g; \mathcal{G}_{\diamond,L-1}\right)\right]$$
(6)

to specify the dependence from the starting product graph $\mathbf{S}_\diamond$, signal $\mathbf{x}_\diamond$, and parameters set $\mathcal{H}$ defining all graph-time filters in (3).

The above steps indicate that building a graph-time neural network reduces to specifying the linear filtering module $\mathbf{H}(\mathbf{S}_{\diamond,\ell})$ [cf. (2)-(3)], the pooling module $\rho(\cdot, \mathcal{G}_{\diamond,\ell})$ [cf. (4)], and the nonlinearity module $\sigma[\cdot; \mathcal{G}_{\diamond,\ell}]$ [cf. (5)]. Each of these modules can be generalized from the corresponding ones developed for graph neural networks [28]. The filtering module can be convolutional [11]; message passing [29], attention [30], or even an edgenet-based [16]. Likewise, the pooling module can be zero-pad [25], self-attention [31], or hierarchical (e.g., Kron reduction) [28]. The nonlinearity module can either be pointwise (e.g., ReLU) or graph-adaptive [32]. We will develop on a graph-time convolutional module with zero-pad pooling and pointwise ReLU nonlinearities. Our rationale is that convolutions allow for effective parameter sharing, inductive learning, and efficient implementation, while zero-pad pooling and pointwise nonlinearities make the architecture independent from graph-reduction techniques or other modules.

3.1. Graph-Time Convolutional Filtering

Following the shift-and-sum principle of the convolutional operator [18, 23], we define the output of a graph convolutional filter or order K over the parametric product graph $\mathbf{S}_\diamond$ as

$$\mathbf{u}_\diamond = \sum_{k=0}^K h_k \mathbf{S}_\diamond^k \mathbf{x}_\diamond = \sum_{k=0}^K h_k\left(\sum_{i=0}^1 \sum_{j=0}^1 s_{ij}(\mathbf{S}_T^i \otimes \mathbf{S}^j)\right)^k \mathbf{x}_\diamond \quad (7)$$

where $h_0, \ldots, h_K$ are the filter parameters. Expression (7) shifts-and-sums signal $\mathbf{x}_\diamond$ via the shift operator $\mathbf{S}_\diamond$ over the parametric product graph $\mathcal{G}_\diamond$ to obtain the output $\mathbf{u}_\diamond$. Since $\mathcal{G}_\diamond$ captures graph-time locations, the shifts are now performed over the spatial graph $\mathcal{G}$ and the temporal graph $\mathcal{G}_T$ justifying the qualifier graph-time convolution for operation (7).[1] Defining the graph-time filtering matrix $\mathbf{H}(\mathbf{S}_\diamond) = \sum_{k=0}^K h_k \mathbf{S}_\diamond^k$ allows writing (7) as $\mathbf{u}_\diamond = \mathbf{H}(\mathbf{S}_\diamond)\mathbf{x}_\diamond$ [cf. (2)].

Contrasting (7) with the graph convolutional filter [11], we can see that the graph-time convolutional filter aggregates at the space-time location (i,t) information from space-time neighbors that are up to K hops away over the product graph $\mathcal{G}_\diamond$. This information is obtained from the shifts $\mathbf{S}_\diamond \mathbf{x}_\diamond, \ldots, \mathbf{S}_\diamond^K \mathbf{x}_\diamond$. The space-time location (i,t) in $\mathcal{G}_\diamond$ receives in this way information from other space-time locations (j, τ) that are up to K hops away in $\mathcal{G}_\diamond$ for $j \in \mathcal{V}$ and $\tau \in [T]$. In other words, node i at time t receives *present* signal information $\{x_{jt}\}$ from its spatial $K-$hop neighbors, and *past* information $\tau < t$ from itself $\{x_{i\tau}\}$ and its spatial neighbors $\{x_{j\tau}\}$ that can be reached through a path of length K in $\mathcal{G}_\diamond$. Thus, the filter order K controls the spatiotemporal locality of the graph-time convolutional filter (7).

---

[1]We can see the convolutional nature of (7) by particularizing the parametric product graph to the Cartesian product. Then, setting $N = 1$ node for the spatial graph with a self-loop and the temporal graph to the directed line, expression (7) implements the temporal convolution. Setting $T = 1$ node for the temporal graph with a self-loop, expression (7) reduces to a graph convolution.

Computation & recursive implementation. We now discuss the recursive implementation of the proposed GTCNN to provide insights on its computational complexity and scalability. While working with the product graphs, we can exploit the sparsities in $\mathbf{S}_T$ and $\mathbf{S}$ to reduce the computational cost for the output (7). If parameters $\{s_{ij}\}$ are fixed (i.e., the product graph), we can work directly with $\mathbf{S}_\diamond$, which has a sparsity of order $|\mathcal{E}_\diamond| = NT + N|\mathcal{E}_T| + T|\mathcal{E}| + |\mathcal{E}_T||\mathcal{E}|$. Computing output $\mathbf{u}_\diamond$ requires computing the shifts $\mathbf{x}_\diamond^{(k)} = \mathbf{S}_\diamond^k \mathbf{x}_\diamond$. For this, we can use the well-know recursive implementation of shifting signals over a graph [21, 25, 32] and write $\mathbf{x}_\diamond^{(k)} = \mathbf{S}_\diamond^k \mathbf{x}_\diamond = \mathbf{S}_\diamond \mathbf{x}_\diamond^{(k-1)}$ with $\mathbf{x}_\diamond^{(0)} = \mathbf{x}_\diamond$; hence, we can obtain the output $\mathbf{u}_\diamond$ with the linear cost $\mathcal{O}(K|\mathcal{E}_\diamond|)$.

If parameters $\{s_{ij}\}$ are to be learned (i.e., the product graph), computing $\mathbf{S}_\diamond$ beforehand or using (7) can be unaffordable in large-scale settings because of the powers of $\mathbf{S}_\diamond$ (cubic cost in NT). To allow scalability, we first expand all polynomials of order k and rearrange the terms to write (7) as

$$\mathbf{u}_\diamond = \sum_{k=0}^{K} h_k \mathbf{S}_\diamond^k \mathbf{x}_\diamond = \sum_{k=0}^{\overline{K}} \sum_{l=0}^{\widetilde{K}} h_{kl} \big(\mathbf{S}_T^l \otimes \mathbf{S}^k\big) \mathbf{x}_\diamond \qquad (8)$$

for some orders $\overline{K}$ and $\widetilde{K}$ and parameters $\{h_{kl}\}$. To compute output $\mathbf{u}_\diamond$, we need to compute all terms of the form $\mathbf{x}_\diamond^{(kl)} = (\mathbf{S}_T^l \otimes \mathbf{S}^k) \mathbf{x}_\diamond$. Exploiting the Kronecker product property $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}$, we can write the latter as

$$\mathbf{x}_\diamond^{(kl)} = (\mathbf{S}_T \otimes \mathbf{I}_N)(\mathbf{I}_T \otimes \mathbf{S})(\mathbf{S}_T^{l-1} \otimes \mathbf{S}^{k-1}) \mathbf{x}_\diamond. \qquad (9)$$

Thus, we can compute $\mathbf{x}_\diamond^{(kl)}$ again recursively as

$$\begin{aligned} \mathbf{x}_\diamond^{(kl)} &= (\mathbf{S}_T \otimes \mathbf{I}_N)(\mathbf{I}_T \otimes \mathbf{S}) \mathbf{x}_\diamond^{(k-1,l-1)} \\ &= (\mathbf{S}_T \otimes \mathbf{I}_N) \mathbf{x}_\diamond^{(k,l-1)} \end{aligned} \qquad (10)$$

with initialization $\mathbf{x}_\diamond^{(00)} = \mathbf{x}_\diamond$. Recursion (10) implies we can compute $\mathbf{x}_\diamond^{(kl)}$ from $\mathbf{x}_\diamond^{(k-1,l-1)}$ with a cost of order $\mathcal{O}(T|\mathcal{E}| + N|\mathcal{E}_T|)$ and since we need to perform the latter for all $k \in [\overline{K}]$ and $l \in [\widetilde{K}]$, we have a computational cost of order $\mathcal{O}(\overline{K} T|\mathcal{E}| + \widetilde{K} N|\mathcal{E}_T|)$, which is linear in the product graph dimensions.

Note also that form (8) improves our control on the spatiotemporal locality through orders $\overline{K}$ and $\widetilde{K}$. A larger $\overline{K}$ implies more reach over the spatial graph (i.e., $\mathbf{S}^{\overline{K}}$), while a larger $\widetilde{K}$ implies more reach over the temporal graph (i.e., $\mathbf{S}_T^{\widetilde{K}}$). Both orders are design choices.

3.2. Graph-Time Pooling

Building upon [25], we propose a zero-pad graph-time pooling module to reduce the dimensionality of the graph-time features without resorting to any coarsening approach. This simple, yet non-trivial, generalization needs to account now for the spatiotemporal peculiarities induced by the product graph. The pooling approach has three steps: $i)$ summarization; $ii)$ slicing; $iii)$ downsampling.

Summarization changes the signal value of a node with a summary (e.g., max, mean) of the values in the local neighborhood. Given the $N_{\ell-1} T_{\ell-1}$ convolutional features $\mathbf{u}_{\diamond,\ell-1}$, graph $\mathbf{S}_{\diamond,\ell-1}$, and defined the reachability integer α_ℓ, we denote the summarized features as

$$\mathbf{v}_{\diamond,\ell} = \Gamma(\mathbf{u}_{\diamond,\ell-1}; \alpha_\ell; \mathbf{S}_{\diamond,\ell-1}) \qquad (11)$$

which signifies the ith entry $[\mathbf{v}_{\diamond,\ell}]_i$ is the summarization of signal values $[\mathbf{u}_{\diamond,\ell-1}]_j$ from nodes j that are up to α_ℓ hops away, i.e., $\{j : [\mathbf{S}_{\diamond,\ell-1}^k]_{ij} \neq 0 \text{ for some } k \leq \alpha_\ell\}$. Function $\Gamma(\cdot)$ can be, for instance, $\max(\cdot)$ or $\text{mean}(\cdot)$. Since the local neighborhood of a node in $\mathcal{G}_{\diamond,\ell-1}$ includes also spatial nodes from different time instances, the features in $\mathbf{v}_{\diamond,\ell}$ are summarized over both the graph and the temporal domain. The $N_{\ell-1} T_{\ell-1}$ summarized features in $\mathbf{v}_{\diamond,\ell}$ can now be seen as another product graph signal over graph $\mathcal{G}_{\diamond,\ell-1}$.

Remark 1. Summarization is an implicit low-pass operation and the type of product graph has an impact on its severity. If the product is parametric or strong, summarization is performed over larger spatiotemporal neighborhoods due to the inter-connections in different time instances [cf. Fig. 1; grey edges]. This wide summarization leads to a stronger low pass and reduces the signal variability. We limit the local spatiotemporal neighborhood by using the Cartesian product [cf. Fig. 1; green edges] in the summarization step. ∎

Slicing reduces the dimensionality across the temporal dimension. Given the $N_{\ell-1} T_{\ell-1}$ summarized features $\mathbf{v}_{\diamond,\ell}$, we de-vectorize it into the union of $T_{\ell-1}$ spatial graph signals

$$[\mathbf{v}_{\ell,1}, \ldots, \mathbf{v}_{\ell,T_{\ell-1}}] = \text{vec}^{-1}(\mathbf{v}_{\diamond,\ell})$$

where each $\mathbf{v}_{\ell,t}$ is a spatial graph signal of dimension $N_{\ell-1}$. Denoting the slicing ratio at layer ℓ by R_ℓ, we keep from $[\mathbf{v}_{\ell,1}, \ldots, \mathbf{v}_{\ell,T_{\ell-1}}]$ one column (or slice) every R_ℓ, resulting in the $T_\ell = \lceil T_{\ell-1}/R_\ell \rceil$ output slices $\{\mathbf{w}_{\ell,\tau}\}$. These output slices are vectorized back into the product graph signal $\mathbf{w}_{\diamond,\ell} = \text{vec}([\mathbf{w}_{\ell,1}, \ldots, \mathbf{w}_{\ell,T_\ell}])$. Denoting the slicing operation at layer ℓ as $\Delta_\ell(\cdot) : \mathbb{R}^{N_{\ell-1}T_{\ell-1}} \to \mathbb{R}^{N_\ell T_\ell}$, we can write the sliced features as

$$\mathbf{w}_{\diamond,\ell} = \Delta_\ell(\mathbf{v}_{\diamond,\ell}; R_\ell). \qquad (12)$$

I.e., slicing a product graph signal $\mathbf{v}_{\diamond,\ell}$ of dimensions $N_{\ell-1}T_{\ell-1}$ yields another product graph signal of dimensions $N_{\ell-1}T_\ell$, in which only the temporal dimension is reduced. The product graph over which the sliced signal $\mathbf{w}_{\diamond,\ell}$ resides can be built using the same rule we built the initial product graph $\mathcal{G}_\diamond$ but with a smaller temporal graph of $T_\ell \leq T_{\ell-1}$ nodes (e.g., a directed line containing fewer nodes).

Downsampling reduces the number of active nodes across the spatial dimension from $N_{\ell-1}$ to N_ℓ without modifying the underlying spatial graph. This is done via zero-padding, i.e., we set to zero the value of inactive nodes while preserving the value on the active ones. Given a binary sampling matrix $\mathbf{C}_\ell$ from the combinatorial set

$$\mathcal{C}_\ell = \{\mathbf{C}_\ell \in \{0,1\}^{N_\ell T_\ell \times N_{\ell-1} T_\ell} : \mathbf{C}_\ell \mathbf{1} = \mathbf{1}, \mathbf{C}_\ell^\top \mathbf{1} \preceq \mathbf{1}\}$$

we compute the downsampled features as

$$\mathbf{z}_{\diamond,\ell} = \mathbf{C}_\ell \mathbf{w}_{\diamond,\ell}. \qquad (13)$$

When $[\mathbf{C}_\ell]_{ij} = 1$, it means the jth component of $\mathbf{w}_{\diamond,\ell}$ is selected and stored in the ith entry of $\mathbf{z}_{\diamond,\ell}$. Therefore, vector $\mathbf{z}_{\diamond,\ell}$ is a product graph signal of dimensions $N_\ell T_\ell$ residing over the active nodes $N_{\ell-1} T_\ell$ of the product graph $\mathcal{G}_{\diamond,\ell}$ obtained from the slicing step. The remaining nodes in $\mathcal{G}_{\diamond,\ell}$ have a zero value.

Downsampling through the sampling matrix $\mathbf{C}_\ell$ requires discussing two main aspects. First, designing $\mathbf{C}_\ell$ is to a large extent an art and it needs to capture the physicality of the problem but also the coupling between the higher-level features at layer ℓ and the spatial graph. We consider $\mathbf{C}_\ell$ to select the nodes with the highest degree as [25]. Second, the set of active nodes in deeper layers is by definition a subset of the active nodes in the earlier layers. To track the location of these active nodes we can consider the nested sampling matrix $\mathbf{D}_\ell = \mathbf{C}_\ell \ldots \mathbf{C}_1$ and use it for zero-padding. The sparsity of $\mathbf{C}_\ell$ and $\mathbf{D}_\ell$ can be in turn used to compute the graph-time convolutional filter output [cf. (7)] with a reduced shift operator like for the conventional zero-pad pooling in GNNs; refer to [25, Sec. III-A] for the technicalities of this implementation.

In summary, a GTCNN is an architecture in which each layer is composed of a graph-time convolutional module defined by (7)-(8), a graph-pooling module defined by (11)-(12)-(13), and a nonlinearity module (5). The higher-level features of layer ℓ are input to the successive layer as per (2) and propagated down the cascade until the final GTCNN output [cf. (6)] is obtained, which is the joint graph-time embedding. The embedded features are fed optionally to fully connected layers and then to a loss function to learn the parameters in $\mathcal{H} = \{\{h_{k\ell}^{fg}\}; \{s_{ij,\ell}\}\}$ comprising the coefficients of all convolutional
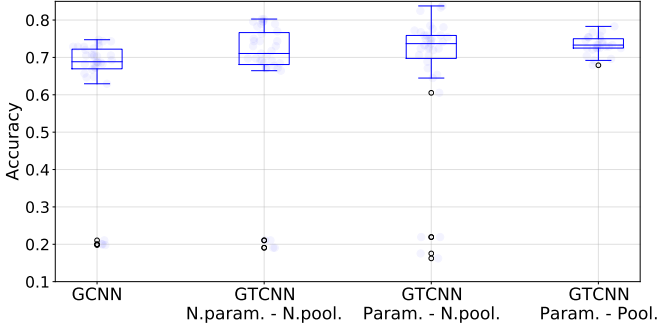
Fig. 2: Comparison of the GCNN baseline with the non-parametric and parametric GTCNN without pooling and with the parametric GTCNN with pooling.

filters $\{h_{k\ell}^{fg}\}$, all parametric product graphs $\{s_{ij,\ell}\}$, and optionally of the fully connected layers. Remark the parameters in $\mathcal{H}$ are independent of the graph dimensions, while the cost of the GTCNN is linear and governed by the graph-time convolutional filters.

4. NUMERICAL RESULTS

This section corroborates the performance of the GTCNN to provide insights on its inner-working mechanisms and compare it with baseline and state-of-the-art alternatives. We used ADAM to train all models with decaying factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$ [33]. The temporal graph is the directed line.[2]

4.1. Source Localization

First, we consider a controlled synthetic experiment to highlight the role of the GTCNN key components. The task consists of finding the source of a diffusion process by observing a sequence of T graph signals $\mathbf{x}_t, \ldots, \mathbf{x}_{t+T}$ for a random time instance t. The graph is an undirected stochastic block model of $N = 100$ nodes and $C = 5$ communities. The basic experimental setup is the same as in [25] but we considered $1,200$ data points to avoid duplicates with an $80\% - 10\% - 10\%$ split. For a fair comparison with the graph-only GCNN baseline, we considered T successive signal realizations as features in the input layer. All architectures have two layers and two filters per layer of order two and are trained over $8,000$ epoch with a batch size of 100 samples. We evaluated features in $F_1, F_2 \in \{2, 4, 16, 32\}$, downsampling sizes $N_1 \in \{30, 75, 100\}$ and $N_2 \in \{10, 30, 50\}$, and temporal windows $T \in \{1, 2, 3\}$; see [Supplement; Sec. I].

Fig. 2 compares the GTCNNs with parametric and non-parametric product graphs [cf. Fig. 1], with and without pooling, and with the baseline GCNN. We can see that accounting for the temporal domain via the sparse connectivity of the product graph improves upon GCNN solutions. Better results are achieved via the parametric product graph and by the use of pooling as evidenced by the larger median value and the smaller deviation of the right-most boxplot. Differently from the others, the latter architecture has also no negative outliers, which indicates it learned in all graphs and data splits. We attribute the latter to the fact that the spatiotemporal coupling is learned in a sparse way and to the zero-pad pooling that preserves the original spatial graph.

4.2. Forecasting

We now consider the task of forecasting future values of a multivariate time-varying signal given a sequence of T past realizations. We used the setting in [12] and considered the Molene dataset comprising 744 hourly temperature measurements across $N = 32$ stations in a region of France; and the NOAA dataset comprising $8,579$ hourly temperature measurements across 109 in the U.S.. The loss function is the MSE
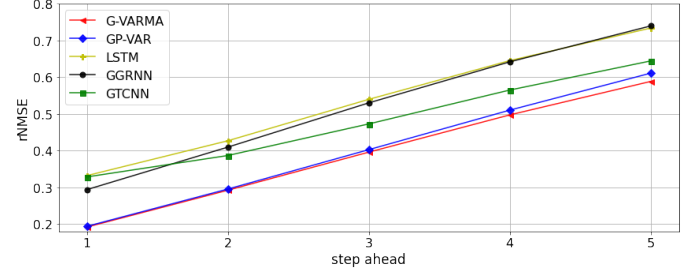
Fig. 3: Root normalized MSE versus future prediction steps for the different methods in the Molene dataset.
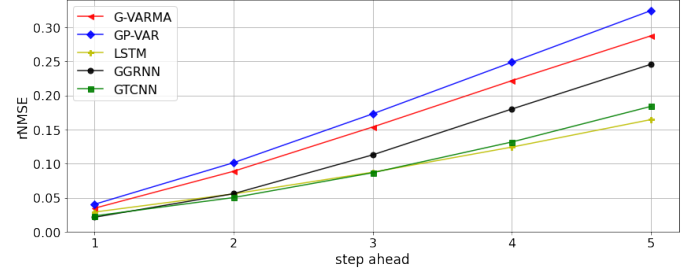

Fig. 4: Root normalized MSE versus future prediction steps for the different methods in the NOAA dataset.

between the one-step ahead prediction $\hat{\mathbf{x}}_{t+1}$ and the true value $\mathbf{x}_{t+1}$ regularized by the norm-one of all parametric product graph coefficients $\mathbf{s} = \text{vec}(\{s_{ij,\ell}^{fg}\})$ [cf. (1)], i.e., $\mathcal{L} = \text{MSE}(\hat{\mathbf{x}}_{t+1}; \mathbf{x}_{t+1}) + \beta\|\mathbf{s}\|_1$, where $\beta \geq 0$ is a scalar. We compared the GTCNN with: i) the linear models G-VARMA and GP-VAP [12]; ii) the gated graph-based RNN (GGRNN) [15]; iii) the time-only LSTM. For the G-VARMA and GP-VAP we used the parameters from [12], while for the GTCNN and GGRNN we evaluated features $F \in \{2, \ldots, 20\}$, orders $K \in \{2, \ldots, 5\}$, observation windows $T \in \{3, 4, 5\}$, and norm-one sparsity weights $\beta \in [0, 0.05]$. For the LSTM we varied the number of hidden units in $\{8, 16, 32, 64\}$. We also considered different learning rates in $[5 \times 10^{-4}, 10^{-3}]$.

Figs. 3 and 4 show the root normalized MSE (rNMSE) for up to five steps ahead prediction for the Molene and the NOAA dataset, respectively. For the Molene dataset, we can see that all graph-based approaches achieve a lower rNMSE than the LSTM. This is because the dataset contains fewer training samples; thus, imposing an inductive bias [26] through the product graph during learning is helpful. In fact, the best performance is achieved by the linear graph models, while the GTCNN performs the best among the neural network alternatives. For the NOAA dataset, instead, we see the opposite trend: the neural network solutions achieve a lower rNMSE compared with the linear graph models. Since the NOAA dataset contains more training samples it allows neural networks to learn more complicated representations. The GTCNN achieves the best performance together with the LSTM, while the GGRNN suffers when predicting more than three steps. Overall, these results put the GTCNN as a valid alternative to learn representations with inductive spatiotemporal biases when both the number of training samples is limited and large.

4.3. Earthquake classification

Lastly, we propose an experiment to find the epicenter of precursor-based earthquakes [34]. Precursor-based detection relies only on wave recordings up to 20 seconds before the strike but not on historical trends. This is a challenging task and our main goal is to show how the GTCNN can be used to approach the latter. We built a dataset from the New Zeeland earthquake service (Supplement Sec. II).

Experimental setup. We considered $4,633$ seismic wave recordings between 2016 and 2020 across 58 stations. These recordings are of
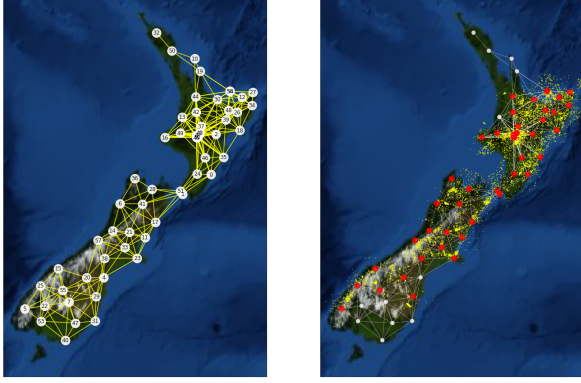
Fig. 5: (Left) Graph structure among the seismic stations. (Right) Yellow dots are the earthquake epicentres; red nodes are stations with an assigned label; white nodes are stations without a label.
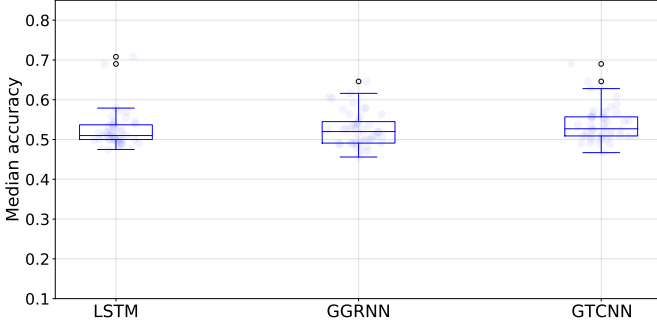

Fig. 6: Median accuracy distribution of the different stations in the one-vs.-all experiment. We can see the GGRNN and the GTCNN have more stations achieving median accuracies of more than 60%, where the GTCNN has a few stations exceeding 65% of accuracy.

earthquakes with a magnitude between one and three and not further than 200km from the closest station. The epicenter is assigned to one of the seismic stations and in total there are 45 stations with assigned labels each with approximately 175 earthquakes. The remaining 13 stations record the waves but do not have an assigned earthquake. We built a geometric graph of $N = 58$ nodes and an edge exists if two stations are within $170, 3$km; like the Molene/NOAA graphs in [12]. Fig. 5 illustrates the graph and the earthquake distribution. The graph signal consists of 20 timesteps of recording in the ten seconds before the strike over all 58 stations. We compared again the GTCNN with the LSTM and the GGRNN. All models are trained w.r.t. the cross-entropy loss for 100 epochs with a batch size of 128, learning rate 10^{-3}, and early stopping at 20 epochs. The results are averaged over 20 realizations. The GTCNN has three layers with grid-searched features $F_1 = 4, F_2 = 8, F_3 = 12$, filter orders $K = 2$, slicing ratios $R = 2$, and active pooling nodes 100%, 90%, and 70% of the total nodes in layers one, two, and three, respectively. The LSTM has 20 grid-searched hidden units, while the GGRNN has the same parameters as in [15]. We considered two experiments: *one-vs-all* binary classification, which assigns the wave to a specific station or any of the other 44 stations; *all-vs-all* 45 class classification scenario, which assigns the wave to one of the 45 stations.

One-vs-all: In this setting, we balanced the dataset by considering half of the points from the class of interest and the other half from all remaining 44 stations. From Fig. 6 and Table 1, we can see that, while all methods have a very similar statistical performance, the GTCNN has a higher average value compared with the other alternatives.

All-vs-all: Since for the approximately $4, 5$k data points a 45 class classification problem is challenging, we measure the performance with a radius-based accuracy metric. That is, if an earthquake has as correct label station i, we consider a correct classification also a

Table 1: Performance of the different methods for the one-vs-all earthquake classification task. In brackets, it is shown the standard deviation of the respective metric. See supplement for each class.

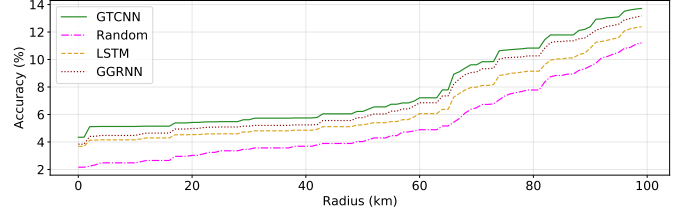| Model	Accuracy	Precision	Recall	F1
LSTM	0.52 (0.05)	0.53 (0.05)	0.53 (0.06)	0.50 (0.06)
GGRNN [15]	0.53 (0.04)	0.53 (0.05)	0.54 (0.06)	0.51 (0.05)
GTCNN	0.54 (0.04)	0.54 (0.05)	0.55 (0.06)	0.53 (0.05)

Fig. 7: Radius-based accuracy results of the different models as a function of the distance from the correct station. I.e., a classification for an earthquake with label station i is considered correct even if assigned to a station that is within a given radius.

station within the radius. Another reason for such a choice is that several earthquakes have their epicenters far from any station (e.g., 70km) or between two or more stations. We considered a split of $60\% - 20\% - 20\%$. From Fig. 7, we can see that graph-based methods perform better than the LSTM highlighting again the impact of this prior when the problem is challenging. As we increase the radius, the performance of all approaches increases with the GTCNN achieving a slightly better result. Note that even by increasing the radius just to 3km the GTCNN shows the biggest jump, which indicates it has assigned several epicenters to stations close to the true label.

We may still correctly argue the reported performance is still far from satisfactory. Reasons for these are multiple (station distribution across the country; the match between the spatial graph and wave propagation; use of only the vertical velocity of the wave), but they, however, show promise for the GTCNN and the other graph-based solutions; and in the one-vs-all accuracy (Fig. 6) we have also seen accuracies for particular stations up to 80%; see Supplement Sec. II.

5. CONCLUSIONS

We proposed a graph-time convolutional neural network to learn spatiotemporal dependencies with a convolutional prior over both the graph and temporal domain. The spatiotemporal data are first transformed into a static signal over a larger product graph between the spatial relationship graph and the temporal relationship graph. The product graph is parametric such that we can learn the spatiotemporal coupling directly from the data. The convolutional module follows the first principles of the convolution operator and builds the output as a shift-and-sum of the input signal over the product graph. A graph-time pooling module is proposed based on spatial zero-padding to preserve the spatial graph-prior in the deeper layers and with a temporal slicing to reduce the dimension across time. We corroborated the GTCNN on classification and regression tasks showcasing its ability to learn spatiotemporal representations. By providing a new alternative to learn from temporal data, the GTCNN opens the doors to a novel research stream including applications from different scientific disciplines. Future work will consider parallelization of the GTCNN to handle large-scale graphs and theoretical advances to shed light on the capability of the GTCNN to discriminate graphs in a spatiotemporal manner.

6. REFERENCES

[1] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Van-dergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[2] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[3] S. Wang, J. Cao, and P. Yu, "Deep learning for spatio-temporal data mining: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[4] D. Chai, L. Wang, and Q. Yang, "Bike flow prediction with multi-graph convolutional networks," in *Proceedings of the 26th ACM SIGSPATIAL international conference on advances in geographic information systems*, 2018, pp. 397–400.

[5] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *Pattern Recognition*, vol. 97, p. 107000, 2020.

[6] Y. Sun, Y. Wang, K. Fu, Z. Wang, C. Zhang, and J. Ye, "Constructing geographic and long-term temporal graph for traffic forecasting," *arXiv preprint arXiv:2004.10958*, 2020.

[7] M. Khodayar and J. Wang, "Spatio-temporal graph deep neural network for short-term wind speed forecasting," *IEEE Transactions on Sustainable Energy*, vol. 10, no. 2, pp. 670–681, 2018.

[8] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," *arXiv preprint arXiv:1709.04875*, 2017.

[9] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 922–929.

[10] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling," *arXiv preprint arXiv:1906.00121*, 2019.

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

[12] E. Isufi, A. Loukas, N. Perraudin, and G. Leus, "Forecasting time series with varma recursions on graphs," *IEEE Transactions on Sig. Proc.*, vol. 67, no. 18, pp. 4870–4885, 2019.

[13] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *International Conference on Neural Information Processing*. Springer, 2018, pp. 362–373.

[14] C. Si, W. Chen, W. Wang, L. Wang, and T. Tan, "An attention enhanced graph convolutional lstm network for skeleton-based action recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 1227–1236.

[15] L. Ruiz, F. Gama, and A. Ribeiro, "Gated graph recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 68, pp. 6303–6318, 2020.

[16] E. Isufi, F. Gama, and A. Ribeiro, "EdgeNets: Edge varying graph neural networks," *arXiv:2001.07620v1 [cs.LG]*, 21 Jan. 2020. [Online]. Available: http://arxiv.org/abs/2001.07620

[17] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," *arXiv preprint arXiv:1801.07455*, 2018.

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

[19] A. Sandryhaila and J. M. Moura, "Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure," *IEEE Signal Processing Magazine*, vol. 31, no. 5, pp. 80–90, 2014.

[20] F. Grassi, A. Loukas, N. Perraudin, and B. Ricaud, "A time-vertex signal processing framework: Scalable processing and meaningful representations for time-series on graphs," *IEEE Transactions on Signal Processing*, vol. 66, no. 3, pp. 817–829, 2017.

[21] 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.

[22] D. Romero, V. N. Ioannidis, and G. B. Giannakis, "Kernel-based reconstruction of space-time functions on dynamic graphs," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 856–869, 2017.

[23] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[24] A. Natali, E. Isufi, and G. Leus, "Forecasting multi-dimensional processes over graphs," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 5575–5579.

[25] 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.

[26] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[27] G. Ortiz-Jiménez, M. Coutino, S. P. Chepuri, and G. Leus, "Sampling and reconstruction of signals on product graphs," in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2018, pp. 713–717.

[28] 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.

[29] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

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

[31] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," *arXiv preprint arXiv:1904.08082*, 2019.

[32] B. Iancu, L. Ruiz, A. Ribeiro, and E. Isufi, "Graph-adaptive activation functions for graph neural networks," in *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2020, pp. 1–6.

[33] D. P. Kingma and J. L. Ba, "ADAM: A method for stochastic optimization," in *3rd Int. Conf. Learning Representations*. San Diego, CA: Assoc. Comput. Linguistics, 7-9 May 2015, pp. 1–15.

[34] R. J. Geller, "Earthquake prediction: a critical review," *Geophysical Journal International*, vol. 131, no. 3, pp. 425–450, 1997.

Supplementary material

This document contains the supplementary material of the paper *Graph-Time Convolutional Neural Networks*. Section 1 analyzes of the different graph convolutional neural network (GTCNN) components in the source localization dataset. Section 2 provides details about the earthquake labeling experiment and the dataset analysis.

1. SOURCE LOCALIZATION

In this section, we analyze the impact of the type of product graph and pooling on the GTCNN.

Product graph. We start with a GTCNN of two layers of $F_1 = F_2 = 2$ and two filters per layer of orders $K = 2$. Our rationale is that this minimalistic architecture allows understanding better how much the product graph aids learning: i) considering more features per layer will lead to a more complex network that may overfit; and ii) the results are independent of pooling, which we analyze next.

Table 1: Mean accuracy (std. dev.) of the GTCNN with different product graphs compared with the baseline GCNN.

Model	T = 1	T = 2	T = 3
GCNN (baseline)	0.64 (0.16)	0.42 (0.21)	0.44 (0.28)
GTCNN Cartesian	n/a	0.65 (0.19)	0.66 (0.19)
GTCNN Strong	n/a	0.63 (0.20)	0.67 (0.17)
GTCNN Parametric	n/a	0.68 (0.18)	0.69 (0.20)

We ignored the GTCNN with the Kronecker product since this architecture does not have a connected product graph. As a baseline, we considered a GCNN working with the time-varying signal of window T as multiple features over the nodes. This comparison is shown in Table 1. The parametric GTCNN achieves the highest mean performance but there is no significant difference with the strong product GTCNN. These two results suggest the temporal relations between neighboring nodes aid learning. Contrarily, if the product graph is not used and the baseline graph convolutional neural network (GCNN) is employed, we see the performance degrades substantially. This is because such a network fails more often to learn from particular graph realizations and data splits, especially, when $T \geq 2$. We attribute the latter to the fact that this procedure is not exploiting the physicality of the problem to capture spatiotemporal relations in a sparse manner but rather treats them as a union of features.

Pooling. We now investigate the effects of pooling in the GTCNN. We considered the two-layered parametric GTCNN with a temporal window $T = 2$. The temporal slicing ratios are $R_1 = 1$ and $R_2 = 2$, i.e., all instances are kept in the first layer and only half in the second layer. Initially, we analyze the pooling effects in the second layer for a different number of features $F_2 \in \{2, 4, 6, 16, 32\}$, and active nodes $N_2 \in \{10, 30, 50\}$. From Fig. 1, we can see the highest performance is achieved when $F_2 \geq 16$ and $N_2 \geq 30$. This indicates that, when the GTCNN is equipped with a higher expressive power (more filters), it can allow for a more drastic pooling in the second layer without affecting the performance. Remark in the latter setting, the GTCNN has also fewer outliers (i.e., cases where it cannot learn), indicating more robustness to graph realizations and data splits.

Next, we analyze the effects of pooling in the first layer. From the earlier results, we fix $F_2 = 16$ features and $N_2 = 30$ active nodes and test for $F_1 \in \{2, 4, 16, 32\}$ and $N_1 \in \{30, 75, 100\}$ to have $N_1 \geq N_2$. From Fig. 2, we can see the performance degrades when more filters are considered in the first layer. This is because the network with $F_2 = 16$ filters has already sufficient discriminatory power for the dataset at hand and increasing it further leads to overfitting. From the pooling perspective, these results indicate that all nodes ($N_1 = 100$) should be kept in the first layer (higher median and lower spread). This is not entirely surprising since the first layer learns lower-level representations and exploits all data. The latter observation is particularly
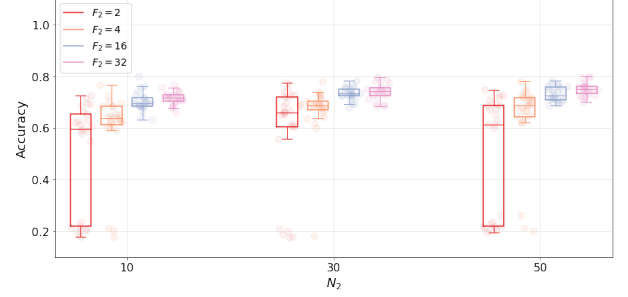

Fig. 1: Accuracy versus the number of active nodes in the second layer of the GTCNN for different features. The first layer has $F_1 = 2$ features and $N_1 = 100$ active nodes. The GTCNN performs better when its expressive power (higher F_2) increases and requires fewer active nodes in the second layer (lower N_2).
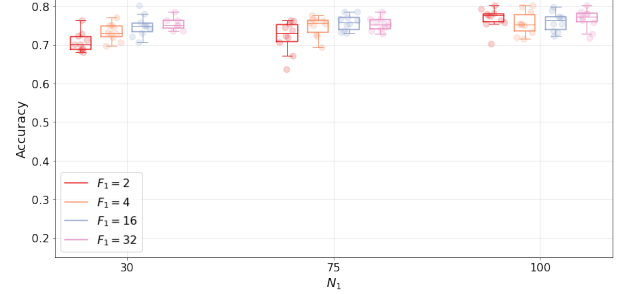

Fig. 2: GTCNN performance for different downsampling nodes and features in the second layer.

visible for $F_1 = 2$ when the network is less prone to overfitting. In conclusion, these results indicate the GTCNN may require all lower level features in the input layer to learn the intermediate representation but can sacrifice a large portion of active nodes in the next layer without affecting the performance.

2. EARTHQUAKE CLASSIFICATION

Dataset. We extracted $90,000$ initial recordings between 2016 and 2020 over 58 seismic stations; Fig. 3. The signal is the weak motion measured along the vertical axis at 100Hz. We kept only those recordings for which the magnitude was between one and three [cf. Fig. 4 (left)] and the depth smaller than 200 km [cf. Fig. 4 (right)]. This resulted into a more uniform distribution with $87,000$ recordings. We further discarded those datapoints for which one of the stations was inactive and retained those earthquakes for which the epicentre was within 75km from the closest station, leaving to approximately $70,000$ datapoints; Fig. 5 (top). The dropped recordings are mostly in the ocean.

We further analyzed the number of assigned earthquakes per station Fig. 5 (bottom). There was one station with no earthquakes assigned; several stations with less than 100; and 27 stations with more than $1,000$. This distribution leads to an unbalanced dataset, which was prioritising the majority classes even by using conventional learning approaches for unbalanced datasets. To achieve a more balanced dataset, we discarded those stations with less than 150 earthquakes and undersampled randomly the recorded seismic waves in those stations containing more than $1,000$ recordings. This led to the final dataset comprising $4,633$ recordings assigned to 45 stations (out of the 58 available), while the graph remains defined over the $N = 58$ stations.

Graph construction. We built a geometric graph following the great-circle distance strategy with an edge if the distance between two stations is smaller than $170,3$ km. We further weighted the graph by setting the edge weights as $A_{ij} = e^{-d(i,j)/\bar{d}}$, where $d(i,j)$ is the great-circle distance between stations i and j and $\bar{d}$ is the average distance. Refer to Fig. 5 in the main document.

Fig. 3: Earthquake Epicentre Distribution. Each yellow point represents an earthquake.
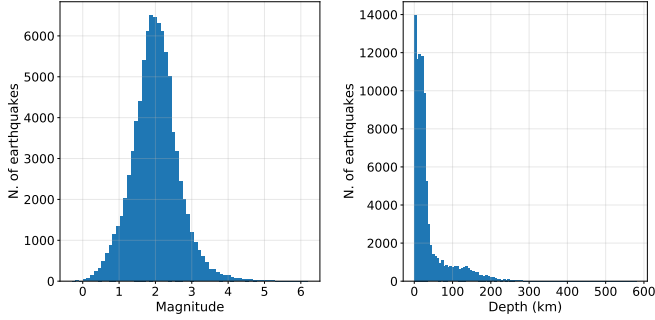

Fig. 4: (Left) Magnitude Distribution. (Right) Depth Distribution.

One-vs-all. Fig. 6 shows the boxplot distributions for the different classes. We can see the GTCNN can reach a median accuracy of about 60% in 11 cases and also hitting up to 70% in a few of them. Nevertheless, the scarcity of the data, the uneven spatial distribution of the earthquakes, and difficulty of working with recordings before the actual earthquake make it difficult for all methods to achieve far superior accuracies. Further research will be based on this aspect and assessing the role of the graph for this setting.
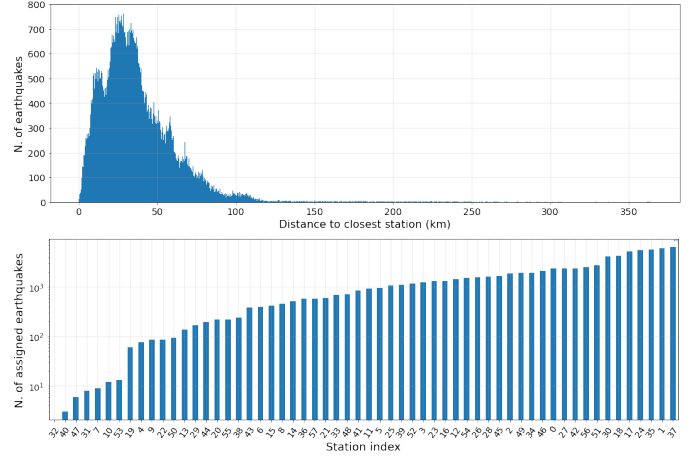

Fig. 5: (Top) Distribution of the epicenter distance from the closest station. (Bottom) Number of assigned earthquakes per station.
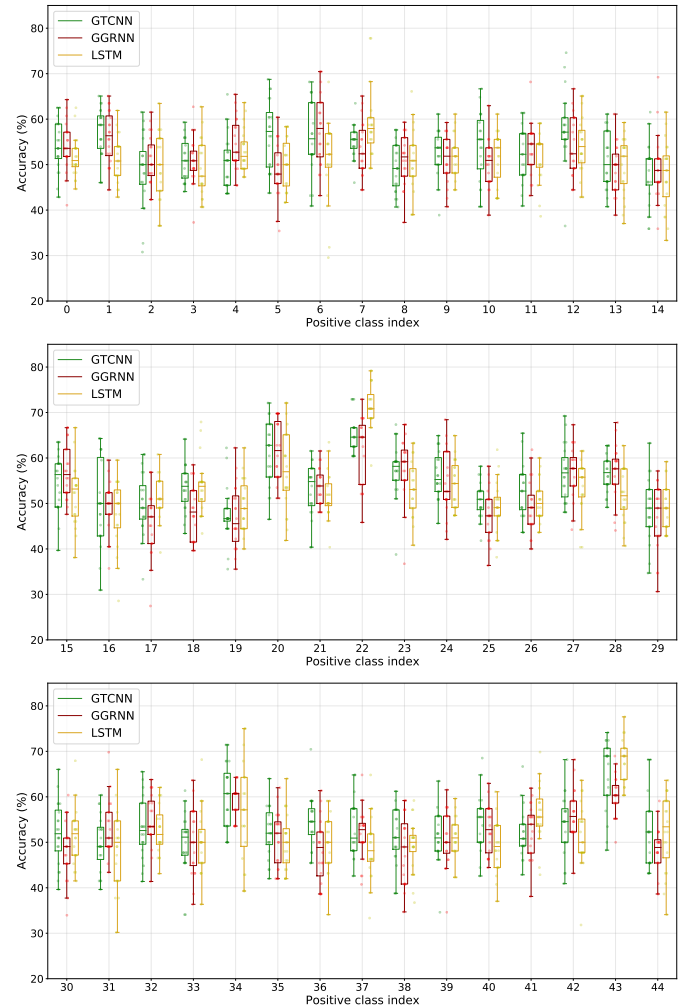

Fig. 6: Accuracies of the proposed GTCNN and alternatives GGRNN and LSTM for the different positive classes (stations) in the one-vs-all classification.