Gated Graph Convolutional Recurrent Neural Networks

Luana Ruiz, Fernando Gama and Alejandro Ribeiro

***Abstract*—Graph processes model a number of important problems such as identifying the epicenter of an earthquake or predicting weather. In this paper, we propose a Graph Convolutional Recurrent Neural Network (GCRNN) architecture specifically tailored to deal with these problems. GCRNNs use convolutional filter banks to keep the number of trainable parameters independent of the size of the graph and of the time sequences considered. We also put forward Gated GCRNNs, a time-gated variation of GCRNNs akin to LSTMs. When compared with GNNs and another graph recurrent architecture in experiments using both synthetic and real-word data, GCRNNs significantly improve performance while using considerably less parameters.**

***Index Terms*—graph neural networks, recurrent neural networks, gating, graph processes**

I. Introduction

The availability of ever-growing volumes of data — often referred to as *big data* — has propelled the use of neural network architectures in both engineering and less traditional fields, such as medicine [1] and business consulting [2]. But learning from large datasets comes with a challenge: it requires complex models with many parameters which, on the one hand, are time and memory-intensive and, on the other, increase the risk of overfitting. To get around these issues, a lot of effort has been put into designing architectures that exploit the underlying structure of data using an amenable number of parameters. The first example are Convolutional Neural Networks (CNNs) [3], which use banks of convolutional filters whose number of parameters is independent of the size of the input to extract shared features across grid-like signals (e.g. images). Then, there are Graph Convolutional Neural Networks (GNNs) [4]–[7], which achieve the same purpose on graph data using graph convolutional filters also known as linear shift-invariant graph filters (LSI-GFs) [8]. A third example are Recurrent Neural Networks (RNNs) [9], designed to process sequential data through the addition of a state or memory variable that stores past information.

The sequences processed by RNNs are usually temporal processes, but they are rarely one-dimensional, i.e., they do not vary only in time. In particular, we will be interested in sequences that are best represented by *graph processes* [10]. Graph processes model a variety of important problems; some illustrative examples are weather prediction from data collected at weather station networks [11] and identifying the epicenter of an earthquake from seismic waves [12].

Supported by NSF CCF 1717120, ARO W911NF1710438, ARL DCIST CRA W911NF-17-2-0181, ISTC-WAS and Intel DevCloud. The authors are with the Dept. of Electrical and Systems Eng., Univ. of Pennsylvania. Email: {rubruiz,fgama,aribeiro}@seas.upenn.edu.

To deal with these scenarios, we introduce a Graph Convolutional Recurrent Neural Network (GCRNN) architecture where the hidden state is a graph signal computed from the input and the previous state using banks of graph convolutional filters and, as such, stored individually at each node. In addition to being local, in GCRNNs the number of parameters to learn is independent of time because the graph filters that process the input and the state are the same at every time instant. GCRNNs can take in graph processes of any duration, which gives control over how frequently gradient updates occur. They can also learn many different representations: a signal (whether supported on a graph or not) or a sequence of signals; a class label or a sequence of labels. While other graph-based recurrent architectures have been proposed in [13]–[15], they are limited to representing sequences of graph signals and, in general, problem specific (most commonly to traffic forecasting). A fourth graph recurrent formulation has been introduced in [16], but it uses recurrence as a way of re-introducing the input data at each layer to capture multiple types of diffusion, and as such does not operate on graph processes. In this architecture, the number of learnable parameters also depends on the number of recurrent layers.

Our GCRNN architecture is further extended to include time gating variables analogous to the input and forget gates of Long Short Term Memory units (LSTMs) [9], which are also implemented using GCRNNs. The objective of gating is twofold: on the input side, to control the importance given to new information, and on the state side, how much of the stored past information the model should "forget".

GCRNNs' ability to learn both graph and time dependencies and the importance of the gating mechanism for long input sequences are demonstrated in experiments on synthetic and real-world data. GCRNNs are compared with basic GNNs and with the DCRNN, a gated graph recurrent architecture from the existing literature [13]. Numerical results show that: (i) GCRNNs largely improve upon GNNs when processing sequential graph data, and (ii) our model uses considerable less parameters and achieves better performance than the DCRNN.

II. Graph Processes

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ be a graph where $\mathcal{V}$ is a set of N nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges and $\mathcal{W} : \mathcal{E} \to \mathbb{R}$ is a function that assigns weights to each edge. The topology of the graph $\mathcal{G}$ can be described by a matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$ that captures the sparsity pattern of its structure, i.e., $[\mathbf{S}]_{ij} = s_{ij}$ can be nonzero only if $(j, i) \in \mathcal{E}$ or $j = i$. Typical choices for this matrix in the literature are the adjacency [17], the Laplacian [18], the random walk [19] and their normalized counterparts [4], [6].

978-9-0827-9703-9/19/$31.00 ©2019 IEEE

We define the neighborhood of node i as $\mathcal{N}_i = \{j \in \mathcal{V} : (j,i) \in \mathcal{E}\}$, and use the expression *local operations* to refer to operations that can be computed by successive interactions of a node i with its neighborhood $\mathcal{N}_i$.

We model data as *graph signals*: a graph signal $\mathbf{x} : \mathcal{V} \to \mathbb{R}$ is such that each element $[\mathbf{x}]_i = x_i$ corresponds to the value of the graph signal at node $i \in \mathcal{V}$. The most basic interaction between the graph signal $\mathbf{x}$ and the graph $\mathcal{G}$ is given by the operation $\mathbf{S}\mathbf{x}$, where, for each $i = 1, \ldots, N$, we have

$$[\mathbf{S}\mathbf{x}]_i = \sum_{j=1}^{N} [\mathbf{S}]_{ij} [\mathbf{x}]_j = \sum_{j \in \mathcal{N}_i} s_{ij} x_j \ . \tag{1}$$

The operation described by (1) is local, because its output can be computed by interacting only with $\mathcal{N}_i$ due to the sparsity pattern of $\mathbf{S}$. The matrix $\mathbf{S}$ has the effect of shifting around the graph the information contained in its nodes and is henceforth called the *graph shift operator* (GSO).

We are interested in graph signals that change with time over the same graph support, typically known as *graph processes* [10]. A graph process $\{\mathbf{x}_t\}_{t \in \mathbb{N}_0}$ is a sequence of graph signals $\mathbf{x}_t \in \mathbb{R}^N$, which are all defined over the same graph support, $\mathbf{x}_t : \mathcal{V} \to \mathbb{R}$. More often than not, there exists some dependency relationship between graph signals at different time instants. This (causal) dependency can be described, generically, by $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots)$ for some function f that in practice is usually unknown.

The graph process is typically accompanied by some target representation $\mathcal{Y}$ relevant to the task at hand, which gives rise to pairs $(\{\mathbf{x}_t\}, \mathcal{Y})$. In regression problems, the target representation is usually another sequence $\{\mathbf{y}_t\}$, while in classification problems it is a single element $\mathbf{y}$ characterizing the sequence over some time interval. The general objective of learning over sequences is to obtain a meaningful estimate $\hat{\mathcal{Y}}$ of the target representation $\mathcal{Y}$ using $\{\mathbf{x}_t\}$. To do this, we propose a novel architecture that we call the *Graph Convolutional Recurrent Neural Network* (GCRNN).

To enhance the descriptive capabilities of our data model, in what follows we consider sequences comprised of F different features $\mathbf{x}_t^f$ for $f = 1, \ldots, F$, where each $\mathbf{x}_t^f \in \mathbb{R}^N$ is a graph signal. A more compact representation is given by the matrix $\mathbf{X}_t \in \mathbb{R}^{N \times F}$, where each column $\mathbf{x}_t^f \in \mathbb{R}^N$ is a graph signal ($f = 1, \ldots, F$) and each row $\tilde{\mathbf{x}}_t^i \in \mathbb{R}^F$ gathers the feature values collected at a single node ($i = 1, \ldots, N$)

$$\mathbf{X}_t = \left[\mathbf{x}_t^1, \ldots, \mathbf{x}_t^F\right] = \begin{bmatrix} (\tilde{\mathbf{x}}_t^1)^\mathsf{T} \\ \vdots \\ (\tilde{\mathbf{x}}_t^N)^\mathsf{T} \end{bmatrix} \ . \tag{2}$$

While we have defined a local operation on $\mathbf{x}_t^f$ as one that respects the sparsity of the graph [cf. (1)], we note that any operation on $\tilde{\mathbf{x}}_t^i$ can be called local as well, since it involves values that are already available at that node.

III. Graph Convolutional Recurrent Neural Networks

A recurrent neural network (RNN) approximates the temporal dependencies of a sequence $\{\mathbf{x}_t\}$ using a hidden Markov model, i.e. $\mathbf{x}_{t+1} \approx g(\mathbf{x}_t, \mathbf{h}_t)$ for some function g and some *hidden state sequence* $\{\mathbf{h}_t\}$. The hidden state sequence is

$$\mathbf{h}_t = \sigma\left(\mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{h}_{t-1}\right) \ , \tag{3}$$

where $\mathbf{A}$ and $\mathbf{B}$ are linear transforms and σ is a nonlinear function, so as to endow the RNN (3) with higher descriptive power. The representation estimate $\hat{\mathcal{Y}}$ can then be obtained from these hidden states. For instance, in a regression problem, we would have $\hat{\mathbf{y}}_t = \rho(\mathbf{C}\mathbf{h}_t)$ with ρ a nonlinear function and $\mathbf{C}$ a linear transform, and in a classification problem, $\hat{\mathbf{y}} = \rho(\mathbf{C}\mathbf{h}_T)$ for the state $\mathbf{h}_T$ computed after some interval T.

The parameters of the linear transforms $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ can be learned by minimizing some loss function $\mathcal{L}(\mathcal{Y}, \hat{\mathcal{Y}})$ over a training set $\mathcal{T} = \{(\{\mathbf{x}_t\}, \mathcal{Y})\}$. We note that the learned linear transforms are the same for all t, giving the RNN (3) enough flexibility to adapt to sequences of different length. Likewise, the number of parameters is independent of the length of the sequence. The hidden state $\mathbf{h}_t$ is expected to store all the past information that is relevant for estimating the target representation.

The knowledge that the sequence $\{\mathbf{x}_t\}$ is comprised of graph signals defined over the same graph $\mathcal{G}$ with GSO $\mathbf{S}$ can be exploited by forcing the linear transforms $\mathbf{A}$ and $\mathbf{B}$ to account for this structure,

$$\mathbf{h}_t = \sigma\big(\mathbf{A}(\mathbf{S})\mathbf{x}_t + \mathbf{B}(\mathbf{S})\mathbf{h}_{t-1}\big) \ . \tag{4}$$

We name model (4) a *Graph Recurrent Neural Network* (GRNN). A particularly compelling parametrization of the linear transforms $\mathbf{A}(\mathbf{S})$ and $\mathbf{B}(\mathbf{S})$ is given by banks of linear shift-invariant graph filters (LSI-GFs). LSI-GFs endow the GRNN model (4) with convolutional characteristics, since they are permutation-invariant local operations and make the number of learnable parameters independent of the size of the graph. More precisely, let the hidden state be described by D features, where each feature $\mathbf{h}_t^d \in \mathbb{R}^N$ is a graph signal. We define the *Graph Convolutional Recurrent Neural Network* (GCRNN) as

$$\mathbf{h}_t^d = \sigma\left(\sum_{f=1}^{F} \mathbf{A}^{df}(\mathbf{S})\mathbf{x}_t^f + \sum_{d'=1}^{D} \mathbf{B}^{dd'}(\mathbf{S})\mathbf{h}_{t-1}^{d'}\right) \ , \tag{5}$$

where $\mathbf{A}^{df}(\mathbf{S})$ and $\mathbf{B}^{dd'}(\mathbf{S})$ are the LSI-GFs

$$\mathbf{A}^{df}(\mathbf{S}) = \sum_{k=0}^{K-1} a_k^{df} \mathbf{S}^k \quad , \quad \mathbf{B}^{dd'}(\mathbf{S}) = \sum_{k=0}^{K-1} b_k^{dd'} \mathbf{S}^k \ , \tag{6}$$

for $d, d' = 1, \ldots, D$ and $f = 1, \ldots, F$. The *filter taps* $\{a_k^{df}\}_{k=0}^{K-1}$ and $\{b_k^{dd'}\}_{k=0}^{K-1}$ are the learnable parameters of the linear transform. Note that there are $DFK + D^2K$ such parameters and that their number is independent of the sequence length and of the size of the graph N. Another primary feature of GCRNNs is that the LSI-GFs (6) are local operations, since they can be computed by $K-1$ successive interactions with the neighbors of each node. The capacity of GCRNNs can be further increased (while maintaining the convolutional characteristics) by using graph convolutional neural networks [5] with several layers in place of $\mathbf{A}(\mathbf{S})$ and $\mathbf{B}(\mathbf{S})$ in (4).

Describing the hidden states $\mathbf{h}_t^d$ as a collection of D graph signals lets us again exploit the graph structure in the computation of the target representation $\hat{\mathcal{Y}}$. Let $\mathbf{H}_t \in \mathbb{R}^{N \times D}$ be a matrix where the hidden states $\mathbf{h}_t^d \in \mathbb{R}^N$ are its columns,

$$\mathbf{H}_t = \left[\mathbf{h}_t^1, \ldots, \mathbf{h}_t^D\right] = \begin{bmatrix} (\tilde{\mathbf{h}}_t^1)^\mathsf{T} \\ \vdots \\ (\tilde{\mathbf{h}}_t^N)^\mathsf{T} \end{bmatrix} , \tag{7}$$

and where the rows of $\mathbf{H}_t$ collect the D features at node i, $\tilde{\mathbf{h}}_t^i \in \mathbb{R}^D$ for $i = 1, \ldots, N$. The estimated representation is computed as $\hat{\mathbf{y}}_t = \rho(\mathbf{C}(\mathbf{S})\mathbf{H}_t)$ for the regression problem and $\hat{\mathbf{y}} = \rho(\mathbf{C}(\mathbf{S})\mathbf{H}_T)$ for the classification problem. Operation $\mathbf{C}(\mathbf{S})$ can be replaced by a graph convolutional neural network to exploit locality, followed by a fully connected layer to adapt dimensions when mapping $\mathbf{H}_t$ to $\hat{\mathbf{y}}_t$ or $\hat{\mathbf{y}}$.

The regression problem where the target representation sequence $\{\mathbf{y}_t\}$ is a sequence of graph signals $\mathbf{y}_t^g \in \mathbb{R}^N$, with $g = 1, \ldots, G$ denoting different features, is of particular interest, as it allows for two possible local models to compute $\hat{\mathbf{y}}_t^g$. The first possibility is to apply a LSI-GF to $\mathbf{h}_t^d$,

$$\hat{\mathbf{y}}_t^g = \rho\left(\sum_{d=1}^{D} \mathbf{C}^{gd}(\mathbf{S})\mathbf{h}_t^d\right) \quad , \quad \mathbf{C}^{gd}(\mathbf{S}) = \sum_{k=0}^{K-1} c_k^{gd}\mathbf{S}^k \ , \tag{8}$$

which demands $K-1$ successive interactions with the neighbors of each node. This requires DGK learnable parameters. Alternatively, the second possibility is to estimate the target features at each node by applying a linear transformation to its own features,

$$\hat{\tilde{\mathbf{y}}}_t^i = \mathbf{C}\tilde{\mathbf{h}}_t^i \tag{9}$$

where $\mathbf{C} \in \mathbb{R}^{G \times D}$, for each $i = 1, \ldots, N$. This alternative entails no neighbor interactions since $\tilde{\mathbf{h}}_t^i$ is stored at node i, and requires only DG parameters if the same linear transform $\mathbf{C}$ is learned for all nodes.

IV. Gating

Deeper RNNs allow taking long term dependencies into account, but this often comes with the challenge of vanishing gradients as long term interactions get exponentially smaller weights at each training step [9]. This is addressed by time gating mechanisms such as the input and forget gates of Long Short-Term Memory units (LSTMs) and the reset and update gates of Gated Recurrent Units (GRUs). These gates are essentially variables taking values between 0 and 1 — each one estimated by their own neural network model — that multiply either or both the input and the state to control the amount of information passed through with time.

Time gating can be readily extended to the context of graph processes. Based on the GCRNN model (5), we define a time-gated architecture as follows,

$$\mathbf{h}_t^d = \sigma\left(\alpha_t \sum_{f=1}^{F} \mathbf{A}_d^f(\mathbf{S})\mathbf{x}_t^f + \beta_t \sum_{d'=1}^{D} \mathbf{B}_d^{d'}(\mathbf{S})\mathbf{h}_{t-1}^{d'}\right) , \tag{10}$$

where the parameter $\alpha_t \in [0,1]$ is the external input gate, and $\beta_t \in [0,1]$ is the forget gate. We call model (10) a *Gated Graph Convolutional Recurrent Neural Network* (GGCRNN). Note that α_t adjusts the importance given to the external inputs $\{\mathbf{x}_t^f\}_{f=1}^F$ at time t and β_t controls how much the GGCRNN (10) will forget (or, equivalently, remember) from the hidden states $\{\mathbf{h}_t^d\}_{d=1}^D$.

Each gate is calculated as the output of a new GCRNN (5) followed by a fully connected layer. To ensure that α_t and β_t are well within the unit interval, a sigmoid activation function follows the fully connected layer computations. More precisely, let $\boldsymbol{\mu}_t^u \in \mathbb{R}^N$ be the U graph signals that describe the state of the input gate (alternatively, let $\tilde{\boldsymbol{\mu}}_t^i \in \mathbb{R}^U$ be the U internal values stored at node i that determine the state of the input gate). These states are updated as

$$\boldsymbol{\mu}_t^u = \xi\left(\sum_{f=1}^{F} \boldsymbol{\Gamma}^{uf}(\mathbf{S})\mathbf{x}_t^f + \sum_{u=1}^{U} \boldsymbol{\Delta}^{uu'}(\mathbf{S})\boldsymbol{\mu}_{t-1}^{u'}\right) , \tag{11}$$

where ξ is a nonlinear function, and $\boldsymbol{\Gamma}^{uf}(\mathbf{S})$ and $\boldsymbol{\Delta}^{uu'}(\mathbf{S})$ are LSI-GFs, cf. (6). The value $\alpha_t \in [0,1]$ of the input gate is then calculated by projecting the states $\{\boldsymbol{\mu}_t^u\}_{u=1}^U$ of the input gate onto a learned vector $\boldsymbol{\omega} \in \mathbb{R}^{NU}$ and applying a sigmoid

$$\alpha_t = \text{sigmoid}(\boldsymbol{\omega}^\mathsf{T}[(\boldsymbol{\mu}_t^1)^\mathsf{T}, \ldots, (\boldsymbol{\mu}_t^U)^\mathsf{T}]^\mathsf{T}) \ . \tag{12}$$

Analogously, let $\boldsymbol{\nu}_t^v \in \mathbb{R}^N$ be V graph signals that make up the state of the forget gate. These states are updated as

$$\boldsymbol{\nu}_t^v = \eta\left(\sum_{f=1}^{F} \boldsymbol{\Phi}^{vf}(\mathbf{S})\mathbf{x}_t^f + \sum_{v'=1}^{V} \boldsymbol{\Psi}^{vv'}(\mathbf{S})\boldsymbol{\nu}_{t-1}^{v'}\right) . \tag{13}$$

with η a nonlinear function, and $\boldsymbol{\Phi}^{vf}(\mathbf{S})$ and $\boldsymbol{\Psi}^{vv'}(\mathbf{S})$ collections of LSI-GFs, cf. (6). Then, the forget gate $\beta_t \in [0,1]$ is computed as

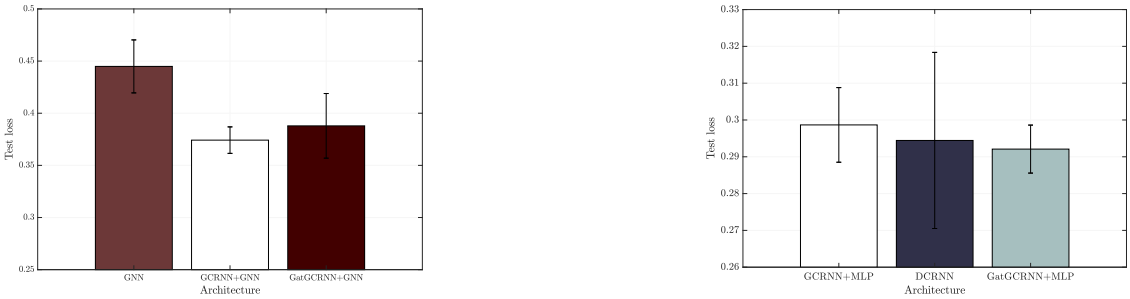
$$\beta_t = \text{sigmoid}(\boldsymbol{\tau}^\mathsf{T}[(\boldsymbol{\nu}_t^1)^\mathsf{T}, \ldots, (\boldsymbol{\nu}_t^V)^\mathsf{T}]^\mathsf{T}) \tag{14}$$

for some learned $\boldsymbol{\tau} \in \mathbb{R}^{NV}$.

Notice that α_t and β_t are the same for every node and every feature, but vary with time. This allows exploiting local operations through the graph filters $\boldsymbol{\Gamma}$, $\boldsymbol{\Delta}$, $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$, and keeps the number of learnable parameters under control.

V. Numerical Experiments

In this section, we present numerical results obtained using multiple variations of our GCRNN architecture in a synthetic experiment — ten-step prediction — and a classification problem involving real seismic data. All simulated architectures consist of a 1-layer GCRNN (5) or GGCRNN (10) and an output neural network mapping the state $\mathbf{H}$ to the target representation $\mathbf{Y}$, which is either a GNN or a *localized* multilayer perceptron that mixes each node's local features individually, cf. (9). In all graph filters, the GSO is the adjacency matrix. The activation function in the GCRNNs is always the hyperbolic tangent, and in the intermediate layers of the output neural network it is the ReLU. In all experiments, the LSI-GFs (6) have $K = 4$ filter taps. If a GCRNN is gated, the GCRNNs used to compute its input and forget gates have state variables with $U = V = D$ features and K filter taps as well, and



(a) (b)

Figure 1: Test loss (mean absolute error) in 10-step prediction. Average loss and standard deviation on 10 different graphs and diffusion datasets. (a) Test loss for a GNN with 2 convolutional layers (right), a GCRNN followed by a GNN (center) and a Gated GCRNN followed by a GNN (left). (b) Test loss for a GCRNN followed by nodewise MLPs with shared parameters (right), the DCRNN from [13] (center) and the gated GCRNN with localized MLPs (left).

are always followed by output neural networks that are full MLPs with a total of ND parameters. All architectures were optimized using ADAM [20] with decaying factors $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

Ten-step prediction. In this experiment, we consider a stochastic block model (SBM) graph $\mathcal{G}$ with $N = 20$ nodes, 4 communities and intra and inter-community edge probabilities of 0.8 and 0.2, respectively. Let $\mathbf{S}$ be the GSO of $\mathcal{G}$. Given an initial graph signal $\mathbf{x}_0 \in \mathbb{R}^N$, $0 \leq [\mathbf{x}_0]_i \leq 1$, the diffused signals $\mathbf{x}_t, t = 1, 2, \ldots$, are generated as

$$\mathbf{x}_t = \mathbf{S}\mathbf{x}_{t-1} + \mathbf{w}_t \ , \tag{15}$$

where $\mathbf{w}_t$ is a zero mean gaussian noise that can be correlated both in time and in between nodes. In particular, we chose $\sigma^2_{\text{time}} = 0.01$ for the variance across time and $\sigma^2_{\text{nodes}} = 0.01$ for the variance across nodes. Likewise, we set the crosscorrelation factors across time and nodes to $\rho_{\text{time}} = \rho_{\text{nodes}} = 0.1$. Fixing the input sequence length to $T = 10$, the 10-step prediction problem consists of estimating $\mathbf{x}_{10}, \mathbf{x}_{11}, \ldots, \mathbf{x}_{19}$ from $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_9$.

We consider 4 GCRNN architectures. The first two are a GCRNN and a Gated GCRNN with $D = 10$ state features whose output neural network is a 1-layer GNN with $K = 4$ filter taps and without fully connected layers. The total number of parameters in these architectures are 480 and $1,760$ (480 for the main GCRNN architecture, and 640 for those of each gate) respectively. The baseline for comparison is a GNN with 2 convolutional layers and no fully connected layer containing 480 parameters, the same as the non-gated GCRNN.

The other two architectures are a GCRNN and a Gated GCRNN where the output neural network is a localized MLP, consisting of the same D-parameter MLP per node. These were compared with the Diffusion Convolutional Recurrent Neural Network (DCRNN) from [13] with 1 recurrent layer, 10 recurrent units and diffusion length 4. The number of parameters of the simple GCRNN, the Gated GCRNN and the DCRNN were 450, $1,730$ and $3,370$ respectively.

All of these architectures were used to simulate the 10-step prediction problem in 10 rounds, with 10 different graphs and dataset realizations. In all rounds, the training, validation and test sets comprised $10,000$, $2,400$ and 200 samples, respectively. All models were trained to optimize the L1 loss (mean absolute error) in 5 training epochs, with batch size 100 and learning rate 0.001 for the GCRNNs followed by GNNs and 0.005 for those followed by localized perceptrons. The average test losses and corresponding standard deviations for each architecture are shown in Figure 1. The main takeaway from Figure 1a is that the GCRNN, even when not gated and containing the same number of parameters as the GNN, achieves a considerably smaller loss than the non-recurrent GNN, which attests to the importance of the recurrence mechanism in processing time sequences. In Figure 1b, although the difference among the average test losses achieved by each architecture is minimal, both GCRNN architectures have about a half or less of the $3,370$ parameters of the DCRNN (with the simple GCRNN counting as little as 450 parameters), and present a much smaller variance as well.

Earthquake epicenter placement on a seismograph network. The second numerical experiment is a classification problem based on data from GeoNet [21], a geological hazard database from New Zealand, and the Incorporated Research Institutions for Seismology (IRIS) database [22], which contains data for 8 active seismographs in the country in the timeframe of analysis. By gathering the origin times of all earthquakes registered by GeoNet between 12/25/2018 and 02/25/2019, we obtain seismic wave readings at these seismographs 30s and 60s before each earthquake, sampled at 2Hz. We then construct a 3 nearest-neighbor seismograph network from the seismographs' coordinates, and from the earthquakes' epicenter coordinates we generate labels corresponding to the nearest sensor to the earthquake's epicenter. In this setting, the objective is to accurately predict the closest node to the epicenter of an earthquake from seismic waves collected at the network's nodes immediately before the shock.

The experiment is conducted twice, once for the 30s and once for 60s duration wave. This results in input sequences with length $T = 60$ and $T = 120$ respectively. We consider 2 GCRNNs. They are: (i) a GCRNN followed by a GNN with

Architecture	30-second wave		60-second wave	
	Accuracy (%)	Param.	Accuracy (%)	Param.
GCRNN	33.33	14,880	32.93	58,560
Gated GCRNN	38.32	29,520	39.12	116,640
GNN	28.34	14,880	30.74	58,560

Table I: Test accuracy and # of parameters for each model in the epicenter placement problem for 30s and 60s waves.

1 convolutional layer mapping the state features to a single feature, and (ii) its gated version. The number of state features are 60 and 120 for the 30s and 60s experiments respectively. In the GCRNNs, the input sequences are used to process a state variable of same length, but only the last state is fed into the output GNN for label prediction. The baseline is a GNN with 1 convolutional layer. Because GNNs cannot process sequences, each element of the sequence is interpreted as an input feature. The convolutional layer of this GNN maps T input features to $T+2$ features, which comes down to $4T(T+2)$ parameters because the number of filter taps is always 4. This GNN was chosen to make for a reasonable comparison with the non-gated GCRNN, which has $4T^2+8T$ parameters.

Out of the $2,503$ earthquakes that happened in the two months to which we restrict our analysis, around 80% were used for training and 20% for testing each model. We optimize a cross-entropy loss with learning rate 0.001 over 10 training epochs and in batches of 100. Test accuracy for each model and each experiment are reported in Table I, as well as the number of parameters in the convolutional layers of each architecture.

Even though all models achieve higher accuracy than random placement, the GCRNN architectures outperform a GNN with same number of parameters as the non-gated GCRNN. Table I also illustrates the importance of the gating mechanism as input sequences grow longer: the percentage difference in the test accuracy achieved by the gated GCRNN and the non-gated GCRNN is 23.8% bigger in the case of the 60-second wave.

VI. Conclusions

We introduced Graph Convolutional Recurrent Neural Networks (GCRNNs) as NN architectures specifically tailored to deal with problems involving graph processes. Their primary feature is the use of banks of graph convolutional filters to implement the recurrence relationship. Thus, the number of parameters is independent of time and of the size of the graph. We have further extended this architecture to Gated GCRNNs (GGCRNNs) with input and forget gates that are akin to those of LSTMs. Numerical results obtained in a synthetic regression problem show that GCRNNs largely improve performance with respect to GNNs when the graph signals are part of a graph process. As for Gated GCRNNs, their ability to take long term dependencies into account was demonstrated in a real world experiment with different input sequence lengths.

References

[1] Z. C. Lipton, D. C. Kale, Elkan C., and R. Wetzel, "Learning to diagnose with LSTM recurrent neural networks," *arXiv:1511.03677v7 [cs.LG]*, 21 March 2017.

[2] G. Phillips-Wren, L. S. Iyer, U. Kulkarni, and T. Ariyachandra, "Business analytics in the context of big data: A roadmap for research," *Commun. Assoc. Inform. Syst.*, vol. 37, 2015.

[3] C.-C. J. Kuo, "The CNN as a guided multilayer RECOS transform," *IEEE Signal Process. Mag.*, vol. 34, no. 3, pp. 81–89, May 2017.

[4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th Int. Conf. Learning Representations*, Toulon, France, 24-26 Apr. 2017, Assoc. Comput. Linguistics.

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

[6] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Annu. Conf. Neural Inform. Process. Syst. 2016*, Barcelona, Spain, 5-10 Dec. 2016, NIPS Foundation.

[7] L. Ruiz, F. Gama, A. G. Marques, and A. Ribeiro, "Median activation functions for graph neural networks," in *44th IEEE Int. Conf. Acoust., Speech and Signal Process.*, Brighton, UK, 12-17 May 2019, IEEE.

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

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, The Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, MA, 2016.

[10] F. Gama and A. Ribeiro, "Ergodicity in stationary graph processes: A weak law of large numbers," *arXiv:1803.04550v1 [eess.SP]*, 12 March 2018.

[11] N. Perraudin and P. Vandergheynst, "Stationary signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3462–3477, July 2017.

[12] 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 Trans. Signal Process.*, vol. 66, no. 3, pp. 817–829, Feb. 2018.

[13] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Int. Conf. Learning Representations 2018*, Vancouver, BC, 30 Apr.-3 May 2018, Assoc. Comput. Linguistics.

[14] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "GaAN: Gated attention networks for learning on large and spatiotemporal graphs," in *Conf. Uncertainty Artificial Intell. 2018*, Monterey, CA, 6-10 Aug. 2018, number 139, Assoc. Uncertainty Artificial Intell.

[15] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *27th Int. Joint Conf. Artificial Intell.*, Stockholm, Sweden, 13-19 July 2018, pp. 3634–3640, Eur. Assoc. Artificial Intell.

[16] V. N. Ioannidis, A. G. Marques, and G. B. Giannakis, "A recurrent graph neural network for multi-relational data," in *44th IEEE Int. Conf. Acoust., Speech and Signal Process.*, Brighton, UK, 12-17 May 2019, IEEE.

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

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

[19] A. Heimowitz and Y. C. Eldar, "A unified view of diffusion maps and signal processing on graphs," in *2017 Int. Conf. Sampling Theory and Appl.*, Tallin, Estonia, 3-7 July 2017, IEEE.

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

[21] Earthquake Commission, GNS Science, and Land Information New Zealand, "GeoNet," https://www.geonet.org.nz/, 20 Feb. 2019.

[22] Incorporated Research Institutions for Seismology, "IRIS earthquake browser," https://www.iris.edu/hq/, 20 Feb. 2019.