On Backpropagation-Free Graph Convolutions*

Luca Pasa[0000−0002−3023−3046], Nicolò Navarin[0000−0002−4108−1754], Wolfgang Erb[0000−0003−3541−5401], and Alessandro Sperduti[0000−0002−8686−850X]

Department of Mathematics, University of Padova, Italy
{luca.pasa, nicolo.navarin, wolfgang.erb, alessandro.sperduti}@unipd.it

Abstract. In this paper, we present neural models for graphs that do not rely on backpropagation for training. This makes learning more biologically plausible and amenable to parallel implementations. The base component of our architecture is a generalization of Gated Linear Networks which allows the adoption of multiple graph convolutions. Every neuron is a *set* of graph convolution filters (weight vectors) and a gating mechanism that selects the weight vector to use for processing based on the node and its topological context. We focus on a message-passing aggregation scheme where the gating mechanism is embedded directly into the graph convolution. We compare the effectiveness of different definitions of node contexts (depending on input or hidden features) and of gating functions (based on hyper-planes or on prototypes). We evaluate the proposed convolutions on several node classification benchmark datasets. The experimental results show that our backpropagation-free graph convolutions are competitive with backpropagation-based counterparts. Moreover, we present a theoretical result on the expressiveness of the proposed models.

Keywords: Graph Convolutional Networks · Graph Neural Network · Deep Learning · Structured Data · Machine Learning on Graphs

1 Introduction

In the last years, several definitions of neural architectures capable to deal with data in structured form, such as graphs, have been presented [1, 2]. The vast majority of graph neural networks in literature are based on the idea of message passing, in which the representation of a node at layer l (or time t if the network is recurrent) is defined as a transformation of the label of the same node and of its neighbors at layer $l-1$ ($t-1$).

While many works focused on defining alternative architectures, at the best of author's knowledge all of them rely on backpropagation to learn the networks' weights. Backpropagation is a powerful and effective method to train deep neural networks (NNs), that has been successfully applied almost ubiquitously in recent

* This work was partly funded by the SID/BIRD project *Deep Graph Memory Networks*, Department of Mathematics, University of Padua.

years. When the amount of available data is not huge, however, the standard approach of training a non-linear NN with backpropagation may quickly lead to overfit the training data. This is in clear contrast to how humans learn, since we do not require nearly the amount of training data modern NNs do to learn how to generalize. Moreover, the backpropagation mechanism is not biologically plausible [3, 4], suggesting that the brain may use different learning algorithms.

Recently, some alternative definitions of multilayer neural networks that do not rely on backpropagation for their training [4, 5] have been proposed. They define local learning rules where each neuron, given its inputs, is trained independently from the rest of the network exploiting a global error signal. This approach allows these networks to: *(i)* be more biologically plausible (i.e. from the current knowledge about the functioning of animal neurons, it seems implausible for a neuron to have access to the connections in a brain area responsible for a subsequent processing step); *(ii)* be more sample efficient/simplify the overall training procedure, since each neuron solves an independent (possibly convex) problem;

The aim of this paper is to explore a contamination between these two cutting-edge research fields, studying how to define neural networks for graph processing that do not rely on backpropagation for their training.

Our exploration is based on the recently proposed Gated Linear Networks (GLN) [5], a family of backpropagation-free neural networks that have been developed for online learning and that have shown promising results. The main characteristic of such networks is that, contrarily to the mainstream approach, the non-linearity is achieved via a gating mechanism instead of element-wise non-linear functions. More specifically, each neuron receives a context vector as additional input, that is used to select one weight vector in a pre-defined set. The only non-linearity lies in such gating mechanism. In fact, once the weight vector is selected, each neuron behaves linearly. The resulting network is a piece-wise linear model (similarly to ReLU networks). While the gating mechanism is not trained, each neuron learns to predict (by modifying the weights) a binary output, and can be trained independently from the rest of the network. In the case of multi-class classification, a one-vs-all approach is exploited.

In order to define neural networks capable of processing graph data, we explore a generalization of the above mechanism based on the approach adopted by many graph convolutional networks, in which the network architecture reflects the structure of the input graph, and node representations are refined at each layer according to the local graph topology via an aggregation operation over neighboring nodes. We also provide a theoretical result on the incremental expressiveness of our models.

The properties inherited from GLNs ensure that our models are (at least in principle) as expressive as their backpropagation-based counterparts, with a significantly easier training phase. Nonetheless, several choices have to be made, such as which neighbor aggregation mechanism to adopt, how to define the contexts on graphs, and how to define the gating mechanism in an efficient way

on graphs for limiting the number of parameters of the network while obtaining good predictive performances.

We experimentally evaluate our backpropagation-free graph convolutional neural architectures on commonly adopted node classification benchmarks, and verify their competitive performance. This work paves the way to novel neural approaches for graph learning.

2 Background

In this section, we introduce the background notation and knowledge on which our model hinges.

2.1 Graph Neural Networks

A learning problem on a graph can be formulated as learning a function that maps nodes to labels. The underlying graph structure is given as $G = (V, E, \mathcal{L})$, where $V = \{v_1, \ldots, v_n\}$ is the set of nodes, $E \subseteq V \times V$ is the set of edges connecting the nodes, and $\mathcal{L} : \mathcal{V} \rightarrow \mathbb{R}^s$ is a function associating a vector of attributes to each node. With $\mathcal{N}(v)$ we denote the set of nodes adjacent to v, i.e. $\mathcal{N}(v) = \{u \,|\, (v, u) \in E\}$. To simplify the notation, we define for a fixed a graph G the matrix $\mathbf{X} = [\mathcal{L}(v_1), \ldots, \mathcal{L}(v_n)]^\intercal$

Given a graph G, our training set is composed by the target information associated to some of the graph nodes, i.e., $\{(v, y) \,|\, v \in W, y \in \mathcal{Y}\}$ with $W \subset V$. For the sake of simplicity, in our presentation we will only consider binary values $\mathcal{Y} \in \{0, 1\}$.

A Graph Neural Network (GNN) is a neural model that exploits the structure of the graph and the information embedded in feature vectors of each node in order to learn a representation $\mathbf{h}_v \in \mathbb{R}^m$ for each vertex $v \in V$. In many GNN models, the computation of $\mathbf{h}_v$ can be divided in two main steps: *aggregate* and *combine*. We can define aggregation and combination by using two functions, $\mathcal{A}$ and $\mathcal{C}$, respectively: $\mathbf{h}_v = \mathcal{C}(\mathcal{L}(v), \mathcal{A}(\{\mathcal{L}(u) : u \in \mathcal{N}(v)\}))$.
The choice of aggregation function $\mathcal{A}$ and combination function $\mathcal{C}$ defines the type of *Graph Convolution (GC)* adopted by the GNN. In [6], the first model that uses graph convolutions is introduced. In the last few years, several different GCs have been proposed [7–13].

In this work, we build on top of two widely adopted graph convolutions. The first one is the GCN [7]

$$\mathbf{H}^{(i)} = \mathcal{F}\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I} + \mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}}\,\mathbf{H}^{(i-1)}\mathbf{W}^{(i)}\right), i > 1 \tag{1}$$

where $\mathbf{A}$ denotes the standard adjacency matrix of the graph G and $\tilde{\mathbf{D}}$ a diagonal degree matrix with the diagonal elements defined as $\tilde{d}_{ii} = 1 + \sum_j a_{ij}$. Further, $\mathbf{H}^{(i)} \in \mathbb{R}^{n \times m_i}$ is a matrix containing the representation $\mathbf{h}_v^{(i)}$ of all nodes in the graph (one per row) at layer i, $\mathbf{W}^{(i)} \in \mathbb{R}^{m_{i-1} \times m_i}$ denotes the matrix of the layer's parameters, and $\mathcal{F}$ is the element-wise (usually, nonlinear) activation function.

The second graph convolution we consider is a slight variation of the first model and commonly referred to as GraphConv [2]:

$$\mathbf{h}_v^{(i)} = \mathcal{F}(\mathbf{h}_v^{(i-1)}\mathbf{W}_1^{(i)} + \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(i-1)}\mathbf{W}_2^{(i)}),$$

where $\mathbf{W}_1^{(i)}, \mathbf{W}_2^{(i)} \in \mathbb{R}^{m_{i-1} \times m_i}$ (with $m_0 = s$, the input dimensionality) are the network parameters.

2.2 Backpropagation-free neural networks

In this paper we exploit recently defined neurons that can be trained locally and independently instead of exploiting backpropagation. We consider the recently proposed Gated Linear Networks [14] (GLNs) where the local optimization problem obtained for each neuron, adopting an appropriate loss, is convex. Moreover, it has been shown that GLNs can represent any function that represent a probability arbitrarily well [15].

The main differences of GLNs compared to MLPs are the following: first, each neuron in a GLN is a Gated Geometric Mixer. Geometric mixing [16] is an ensemble technique that assigns a weight to each weak predictor in input. In GLNs, every unit produces in output its prediction for the target. Given an input vector of probabilities $\mathbf{p} = [p_1, \ldots, p_n]^\top$, geometric mixing is defined as: $\sigma\left(\mathbf{w}^\top \sigma^{-1}(\mathbf{p})\right)$, where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, $\sigma^{-1}(x) = logit(x) = \log(x) - \log(1-x)$ is the logit function (that is the inverse of the sigmoid function), and both of them are applied element-wise.

To achieve non-linearity, specifically piecewise-linearity, GLNs employ a gating mechanism in each neuron. Each neuron divides its input space in *regions*. A geometric mixing (that is a linear model) is associated to each region. The association from examples to regions is carried by a *region assignment* function c. GLNs assume that for each example we have a vectorial representation available, $\mathbf{x} \in \mathbb{R}^{d^{(x)}}$, and a vector representing side-information (or context), i.e. $\mathbf{z} \in \mathbb{R}^{d^{(z)}}$. The c function is defined depending on side-information associated to each input (in case no side-information is available, it is possible to set $\mathbf{z} = \mathbf{x}$). Each neuron in a GLN solves a convex problem, and is trained independently to predict the target.

For the sake of simplicity, we omit bias terms in the following formulations. Given a neuron j at the i-th layer, its output is defined as:

$$\mathbf{h}_{j,(\mathbf{x},\mathbf{z})}^{(i)} = \sigma\left(\sigma^{-1}\left(\mathbf{h}_{(\mathbf{x},\mathbf{z})}^{(i-1)}\right)^\top \mathbf{w}_{j,(\mathbf{z})}^{(i)}\right), i > 1 \tag{2}$$

with $\mathbf{h}_{(\mathbf{x},\mathbf{z})}^{(0)} = \sigma(\mathbf{x})$. The vector $\mathbf{w}_{j,(\mathbf{z})}^{(i)} \in \mathbb{R}^{m_{i-1}}$ stores the weights associated to the region activated by the context $\mathbf{z}$ for the corresponding neuron. Let us discuss this weight vector in more detail and the gating mechanism that defines how a specific set of weights is selected.

Given an example $(\mathbf{x}, \mathbf{z})$, we can select the weights of a single neuron j at the i-th layer as:

$$\mathbf{w}^{(i)}_{j,(\mathbf{z})} = \left(\boldsymbol{\Theta}^{(i)}_j \mathbf{c}^{(i)}_{j,(\mathbf{z})}\right) \tag{3}$$

where $\boldsymbol{\Theta}^{(i)}_j \in \mathbb{R}^{m_{i-1} \times k}$, k is the number of regions (we assume for simplicity that each neuron in the network considers the same number of regions), and $\mathbf{c}^{(i)}_{j,(\mathbf{z})} \in \mathbb{R}^k$. Notice that the main characteristic of a Gated Linear Neuron is that, instead of having a single weight vector, each GL neuron depends on a *matrix* of parameters $\boldsymbol{\Theta}^{(i)}_j$.

The original paper [14] proposes to implement the gating in the c functions with an halfspace-gating mechanism. Given a vector $\mathbf{z} \in \mathbb{R}^{d^{(z)}}$, and a hyperplane with parameters $\mathbf{a}_i \in \mathbb{R}^{d^{(z)}}$ and $b_i \in \mathbb{R}$, let us define a context function $\tilde{c}_i : \mathbb{R}^{d^{(z)}} \to \{0, 1\}$ as:

$$\tilde{c}_i(\mathbf{z}) = \begin{cases} 1 & \text{if } \mathbf{a}_i^\top \mathbf{z} > b_i \\ 0 & \text{otherwise} \end{cases}$$

that divides $\mathbb{R}^{d^{(z)}}$ in two half-spaces, according to the hyperplane $\mathbf{a}_i^\top \mathbf{z} = b_i$. We can compose $log_2(k)$ (assuming k to be a power of 2) context functions of the same kind, obtaining an higher-order context function $\tilde{c} : \mathbb{R}^{d^{(z)}} \to \{0, 1\}^{log_2(k)}$, $\tilde{c} = [\tilde{c}_1, \ldots, \tilde{c}_k]^\top$. We can then easily define a function f mapping from $\{0, 1\}^{log_2(k)}$ to $\{0, \ldots, k-1\} \subset \mathbb{N}$, obtaining the function $\hat{c} : \mathbb{R}^{d^{(z)}} \to \{0, \ldots, k-1\}$, $\hat{c} = f \circ \tilde{c} = f(\tilde{c}(\mathbf{z}))$. We can exploit the one-hot encoding of the output of such function and re-define it as $c : \mathbb{R}^{d^{(z)}} \to \{0, 1\}^k$, $c = one\_hot(\hat{c})$.

Given a layer i, each neuron j computes a different function $c^{(i)}_j : \mathbb{R}^{d^{(z)}} \to \{0, 1\}^k$. For the j-th neuron at the i-th layer, the output of the context function applied to $\mathbf{z}$ is thus the (one-hot) vector $\mathbf{c}^{(i)}_{j,(\mathbf{z})}$.

2.3 Layer-wise formulation

Exploiting the definition of a single Gated Linear Neuron in the previous section, we can define a whole GLN layer. This formulation will be exploited in the remainder of the paper. The output for the i-th layer in a GLN (with m_i neurons) for a sample $(\mathbf{x}, \mathbf{z})$ is defined as:

$$\mathbf{h}^{(i)}_{(\mathbf{x},\mathbf{z})} = \sigma\left(\sigma^{-1}\left(\mathbf{h}^{(i-1)}_{(\mathbf{x},\mathbf{z})}\right)^\top \mathbf{W}^{(i)}_{(\mathbf{z})}\right), i > 1 \tag{4}$$

where

$$\mathbf{W}^{(i)}_{(\mathbf{z})} = [\mathbf{w}^{(i)}_{1,(\mathbf{z})}, \ldots, \mathbf{w}^{(i)}_{m_i,(\mathbf{z})}], \tag{5}$$

and $\mathbf{W}^{(i)}_{(\mathbf{z})} \in \mathbb{R}^{m_{i-1} \times m_i}$.

Several layers can then be stacked. For a binary classification problem, the last layer will comprise a single neuron, i.e. for a network with l layers we have $\mathbf{W}^{(l)}_{(\mathbf{z})} = \mathbf{w}^{(l)}_{1,(\mathbf{z})}$, $\mathbf{W}^{(l)}_{(\mathbf{z})} \in \mathbb{R}^{m_{l-1} \times 1}$. The resulting model is, by construction, piecewise-linear.

Specifically, given a context $\mathbf{z}$, the model is (up to a final activation function) linear and can be written as $y_{(\mathbf{x},\mathbf{z})} = \sigma\left(\mathbf{x}^\top \mathbf{W}^{(1)}_{(\mathbf{z})} \dots \mathbf{W}^{(l-1)}_{(\mathbf{z})} \mathbf{W}^{(l)}_{(\mathbf{z})}\right) = \sigma\left(\mathbf{x}^\top \mathbf{w}_{(\mathbf{z})}\right)$, with a weight vector $\mathbf{w}_{(\mathbf{z})} \in \mathbb{R}^{d^{(x)}}$.

3 Back-Propagation Free Graph Convolutions

In this section, we define our proposed model, which generalize GLNs to graph-structured data. In particular, we show how to embed the GLN idea into graph convolutions to build models based on the message-passing paradigm.

The core concept behind several definitions of Graph Neural Networks is the aggregation function used to obtain information about the local graph structure surrounding a graph node. The simplest aggregation mechanism involves just the summation over the representations of neighbouring nodes. For this simple mechanism, we obtain the following definition for a single layer in a Gated Linear Graph Neural Network:

$$\mathbf{h}^{(i)}_{(v,\mathbf{z})} = \sigma\left(\sigma^{-1}\left(\mathbf{h}^{(i-1)}_{(v,\mathbf{z})}\right)^\top \mathbf{W}^{(i,1)}_{(\mathbf{z})} + \left(\sum_{(u,\mathbf{z}')\in\mathcal{N}_v} \sigma^{-1}\left(\mathbf{h}^{(i-1)}_{(u,\mathbf{z}')}\right)^\top \mathbf{W}^{(i,2)}_{(\mathbf{z})}\right)\right),$$

for $i \geq 1$, and $\mathbf{h}^{(0)}_{v,\mathbf{z}} = \sigma(\mathcal{L}(v))$. The weights $\mathbf{W}^{(i,2)}_{(\mathbf{z})}, \mathbf{W}^{(i,1)}_{(\mathbf{z})}, and \mathbf{W}^{(i)}_{(\mathbf{z})}$ are defined as per eq. (5) and can be obtained by a backpropagation-free training. This model can be considered as a modification of GraphConv proposed in [2] in which gated geometric mixing has been applied. For this reason we refer to this model also as Backpropagation Free - GraphConv (BF-GraphConv). Similarly to common formulations of graph neural networks, we can express the hidden representation for all the nodes in the graph as a single matrix. We obtain the following form of BF-GraphConv: $\mathbf{H}^{(i)}_{(\mathbf{z})} = \sigma^{-1}\left(\mathbf{H}^{(i-1)}_{(\mathbf{z})}\right)\mathbf{W}^{(i,1)}_{(\mathbf{z})} + \mathbf{A}\sigma^{-1}\left(\mathbf{H}^{(i-1)}_{(\mathbf{z})}\right)\mathbf{W}^{(i,2)}_{(\mathbf{z})}$, where $\mathbf{H}^{(i)}_{(\mathbf{z})} \in \mathbb{R}^{n\times m_i}$ and $\mathbf{H}^{(0)}_{(\mathbf{z})} = \sigma(\mathbf{X})$. BF-GraphConv can therefore be regarded as a piecewise linear GNN depending on the context information $\mathbf{z}$ of the neurons.

Following common definitions of Graph Neural Networks, we can resort to any message passing mechanism and define the Gated Linear counterpart. For instance, we can also consider the GCN presented in eq. (1) which leads to the following BF-GCN: $\mathbf{H}^{(i)}_{(\mathbf{z})} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I}+\mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}}\,\sigma^{-1}\left(\mathbf{H}^{(i-1)}_{(\mathbf{z})}\right)\mathbf{W}^{(i)}_{(\mathbf{z})}\right)$. The resulting model can be regarded as a piecewise linear GCN. In particular, after l layers, the output $\mathbf{H}^{(l)}_{(\mathbf{z})}$ for the context $\mathbf{z}$ can be written as $\mathbf{H}^{(l)}_{(\mathbf{z})} = \sigma\left((\tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I}+\mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}})^l \mathbf{X}\mathbf{w}_{(\mathbf{z})}\right)$, i.e., the BF-GCN model with l layers is a generalization of the simple graph convolutional network (SGC) introduced in [17] and further investigated in [18], where the vetor of weights $\mathbf{w}_{(\mathbf{z})}$ changes based on the input context. Notice that the main differences between the Gated Linear Graph Neural Networks and commonly adopted GNN formulations are the local training and the gating mechanism.

3.1 Incremental expressivity of GLNs

The usage of a region assignment function c for the side information $\mathbf{z} \in \mathbb{R}^{d_\mathbf{z}}$ is equivalent to a partitioning $\mathcal{P}(c)$ of the space $\mathbb{R}^{d_\mathbf{z}}$ into k disjoint regions. We can therefore compare different GLNs based on the corresponding partitioning of $\mathbb{R}^{d_\mathbf{z}}$. If a partitioning $\mathcal{P}(c_1)$ is a refinement of a second partitioning $\mathcal{P}(c_2)$, we intuitively expect that the GLN corresponding to the refined partitioning $\mathcal{P}(c_1)$ is more expressive. In the following, we will show this intuition for the two gated linear networks BF-GCN and BF-GraphConv.

Theorem 1. *Consider two Gated Linear GNNs (either BF-GCN or BF-GraphConv) with gated geometric mixing based on two region assignment functions c_1 and c_2. We assume that the partitioning $\mathcal{P}(c_1)$ is a refinement of the partitioning $\mathcal{P}(c_2)$. Then, the Gated Linear GNN based on c_1 is more expressive than the GNN based on c_2.*

Proof. We will only consider the BF-GCN model. For simplicity, we will also assume that $\mathbf{X} = \mathbf{x} \in \mathbf{R}^{n \times 1}$, i.e., that the dimension of the input variable is $s = 1$ and, thus, that we have real-valued weights $w_{(\mathbf{z})} \in \mathbb{R}$. Now, let $\mathcal{H}(c)$ denote the set of all possible functions $\mathcal{H}(c) = \left\{ \mathbf{y}_{(\mathbf{z})} = \sigma((\tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I} + \mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}})^l \, \mathbf{x} \, w_{(\mathbf{z})}) \mid w_{(\mathbf{z})} \in \mathbb{R} \right\}$ generated by a BF-GCN network with a region assignment function c. Let $R \in \mathcal{P}(c_2)$ be one of the regions in the partitioning given by the function c_2. As $\mathcal{P}(c_1)$ is a refinement of $\mathcal{P}(c_2)$ we can decompose R as $R = R_1 \cup \cdots \cup R_r$ with $R_i \in \mathcal{P}(c_1)$ for $i \in \{1, \ldots, r\}$. Now, if in $\mathcal{H}(c_1)$ we impose that $w_{\mathbf{z}_i} = w_{\mathbf{z}_j}$ if $\mathbf{z}_i \in R_i$ and $\mathbf{z}_j \in R_j$ for $i, j \in \{1, \ldots, r\}$, then every $\mathbf{z} \in R$ obtains the same weight $w_\mathbf{z}$. Thus, we see that every function in $\mathcal{H}(c_2)$ can be formulated as a function in $\mathcal{H}(c_1)$. On the other hand, if we can choose $w_{\mathbf{z}_i} \neq w_{\mathbf{z}_j}$ freely for $\mathbf{z}_i \in R_i$ and $\mathbf{z}_j \in R_j$ we can enforce $\mathbf{y}_{(\mathbf{z_1})} \neq \mathbf{y}_{(\mathbf{z_2})}$ for $\mathbf{z}_i, \mathbf{z}_j \in R$ (excluding exceptional cases in which $(\tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I} + \mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}})^l \mathbf{x}$ vanishes on domains linked to R_i and R_j). Therefore, the set $\mathcal{H}(c_1)$ is, in general, strictly larger than $\mathcal{H}(c_2)$.

3.2 Context functions

The context function presented in Section 3 and exploited in (3) is based on random half-space gating. That definition is suited for online learning, where the training data distribution is not known beforehand. However, it is not data-driven and may result in the necessity of defining a high number of context regions to obtain a sufficiently non-linear model. Notice that the halfspace gating mechanism depends on some hyperparameters: in addition to the number of regions k, one has to choose the parameters of the distribution from which to sample the weights corresponding to each hyperplane (e.g. mean and variance assuming they are sampled from a normal distribution). Setting these hyperparameters may be challenging, since results can be strongly affected by their choice.

In this section, we propose an alternative approach that can be exploited in the batch learning scenario and that does not depend on any parameter but the number of regions to consider. In particular, we propose to define a

partition of the space based on a set of prototypes [19]. Each point in the space is assigned to its closest prototype, obtaining a Voronoi tessellation. Note that half-space gating generates a division of the context space that can be represented as a planar straight-line graph (PSLG) instead. It is possible to show that any PSLG coincides with the Voronoi diagram of some set S of points (i.e. prototypes) [20]. Similarly to the half-space gating mechanism, the prototypes are not learned. However, instead of randomly generating them, we propose to sample at random among the training examples. This ensures that each prototype will lie on the input data manifold. Moreover, as mentioned before, this approach relieves us from many hyper-parameter choices.Let $\mathbf{P}_j^{(i)} \in \mathbb{R}^{2^m \times m_z}$ be the matrix of prototypes. We can formally define the context vector $\mathbf{c}_{(z)}^{(i)} \in \{0,1\}^{c_i}$ as $\mathbf{c}_{j,(z)}^{(i)} = one\_hot(argmin(2\_dist(\mathbf{P}_j^{(i)}, \mathbf{z}))$, where 2_dist computes the 2-norm distance between each row of $\mathbf{P}_j^{(i)}$ and $\mathbf{z}$ is the context vector.

4 Experimental Results

We empirically validated the proposed backpropagation-free graph neural networks on four widely adopted datasets of node classification: Citeseer, Cora, Pubmed and WikiCS [21].

We developed all the models involved in the comparison using PyTorch Geometric [22]. As baseline models, we considered the GCN and the GraphConv convolutions. For these models we exploit the implementation provided by PyTorch Geometric.

Furthermore, we do not limit our exploration of backpropagation-free models, and in particular the multilayer ones (BF-GCN and BF-GraphConv), to the standard methodology that uses the input $\mathbf{X}$ as context, but we also assess the effects of using each node's inputs, i.e. the hidden representation computed at the previous layer label ($\mathbf{H}$) as a context for all gated neurons. For all the datasets, we solve the resulting optimization problems with the Adam algorithm (a variant of stochastic gradient descent with momentum and adaptive learning rate).We used early stopping (with the patience set to 15 epochs) and model checkpoint, monitoring the accuracy on the validation set. We set the maximum number of epochs to 250. All the baseline experiments involved softmax activation function applied to the last layer. The results were obtained by performing 5 runs for each model. For our experiments, we adopted a machine equipped with: 2 x Intel(R) Xeon(R) CPU E5-2630L v3, 192GB of RAM and a Nvidia Tesla V100.

4.1 Model selection

A key aspect to consider is the procedure adopted to select the hyper-parameters (such as learning rate, regularization, network architecture, etc.). Many papers report, for each dataset, the best performance (on the test set) obtained after testing many hyper-parameter configurations. Thus, the results reported in

literature for different graph convolutions are not always comparable one to each other.

We recall that the contribution of this work to show that it is possible to match the performance of different graph convolutional neural networks (and thus to perform an effective representation learning) even not relying on backpropagation.

For these reasons, we decided to focus on two widely adopted graph convolutions, and run all the experiments using the same fair model selection procedure, where we select all the hyper-parameters of each method on the validation set. The hyper-parameters of the model (number of hidden units, number of layers, learning rate, weight decay, and dropout only for backpropagation-based models) were selected by using a limited grid search, where the explored sets of values do change based on the considered dataset. Further details are reported in Appendix A.

4.2 Discussion

The results obtained by the backpropagation-free graph neural networks, namely BF-GCN, BF-GraphConv, are in general comparable and sometimes higher than the ones of the baselines trained with backpropagation. We recall that the backpropagation-free alternatives are significantly easier to train since they optimize convex optimization problems (one for each neuron, that can be parallelized), compared to the single but significantly more complex nonlinear problem optimized by the baselines. In Table 1 we report the results obtained validating all the hyper-parameters on the validation set. For each method and dataset we report the average accuracy and the standard deviation over 5 runs. Let us start discussing the GCN convolution mechanism, where the results of backpropagation-based (GCN) and the proposed backpropagation-free (BF-GCN) models with the two alternative context definitions, i.e. $\mathbf{H}$ and $\mathbf{X}$ are reported.

Moreover, for BF-GCN (as for the all backpropagation-free models) we experimented with both the context function based on random half-space gating and our proposal of defining the partition of the space based on a set of prototypes (*context function* column in Table 1). In this case, the backpropagation-free models perform slightly better than the backpropagation counterpart in 2 datasets out of four. In all the cases, the methods are less than one standard deviation apart. We can conclude that, in this case, all the backpropagation-free models based on GCN are capable of learning a representation that is comparably expressive to the one of the backpropagation-based GCN. With this convolution mechanism there is not a clear advantage of using either $\mathbf{H}$ or $\mathbf{X}$ as contexts. These results show that the proposed backpropagation-free methods are pretty resilient and show consistent performance even with significantly different choices of the context space. Moreover, the results suggest that the prototype-based context function allows to reach slightly better performance in terms of accuracy compared to half-space gating.

Let us now consider the GraphConv convolution mechanism. In this case, backpropagation-free models show slight but consistent accuracy improvements on the Citeseer (up to 1.1%) and the Cora (up to 1.4%) datasets. On Pubmed,

their results are at most 2.8% lower than the baseline GraphConv, while on WikiCS all the backpropagation-free variants improve over GraphConv, showing an accuracy improvement up to 5.5%. Notice that the differences in these last two cases are greater than one standard deviation. We can notice that with GraphConv, using $\mathbf{H}$ as context tends to provide slightly higher performances compared to using $\mathbf{X}$. Analyzing the accuracy, no clear advantages can be noticed in using the prototypes-based context function instead of a half-space gating mechanism.

We can conclude that the proposed backpropagation-free graph convolutions are competitive with their backpropagation-based counterparts, while inheriting all the advantages of backpropagation-free methods.

For what concerns the comparison between the two considered gating mechanisms (halfspace and prototype), we obtained no strong evidence in term of accuracy in favour of using one approach over the other. However, the prototype approach does have an advantage in reducing the number of hyperparameters. In fact, it is not straightforward to define the half-space gating hyperparameters, as random initialization of hyperplanes introduces a strong assumption on the data distribution. In our experiments, we decided to keep the same parameters used in [5] for the distribution from which the weights corresponding to each hyperplane are sampled, since modifying them even slightly seemed to impact the predictive performance. On the other hand, the proposed prototype-based context function allows to initialize the gating mechanism in a data-driven way, which turns out to be very simple since we can just uniformly sample them from the training set. Finally, in terms of time complexity the BackPropagation-free models present a huge advantage with respect to the standard GNN. Indeed, the computation (both forward and backward step) of each neuron is independent from all the others, thus it is possible to perform the computation of each unit in parallel. Considering the Message-Passing GLNs, the layerwise construction of the model allows all neurons in the same layer to be computed in parallel.

5 Conclusions and Future Directions

In this paper, we explored a novel locally trainable graph convolutional operator dubbed backpropagation-free graph convolutional network. The proposed GC is inspired by Gated Linear Networks and extends them to be applied to graph-structured data. It relies on a representation space of graph nodes that is shattered into different subspaces according to the node context. Indeed, each neuron that composes the GC operator is defined as a *set* of weight vectors. A gating mechanism within each neuron selects the weight vector to use for processing the input, based on its context. This mechanism allows training each neuron independently, without using back-propagation, resulting in a set of convex problems to solve. We analyzed the strengths and the weaknesses of two variants of our approach exploiting the common message-passing based convolutions (GCN and GraphConv). It is worth noticing that the proposed approach is not limited to the considered graph operators but it can be applied to any graph

Model	gating	context	Citeseer	Cora	Pubmed	WikiCS
GCN	-	-	76.6±1.0	87.5±0.9	88.5±0.3	81.6±0.7
GraphConv	-	-	75.1±1.6	87.1±0.5	**88.9±0.4**	76.8±1.3
BF-GCN	Halfspace	**H**	76.0±1.4	87.7±0.5	88.1±0.3	79.8±0.8
			(2, 16, 2)	(1, 16, 2)	(1, 8, 2)	(2, 6, 2)
		X	76.3±1.8	88.0±0.5	88.1±0.4	80.7±0.7
			(2, 24, 2)	(2, 16, 2)	(2, 4, 2)	(2, 6, 8)
	Proto	**H**	**77.0±0.4**	87.8±0.3	88.1±0.7	81.0±0.9
			(2, 16, 4)	(2, 8, 2)	(2, 32, 4)	(2, 4, 16)
		X	76.9±2.0	88.0±0.4	88.0±0.6	80.4±0.8
			(2, 8, 1)	(2, 32, 2)	(2, 8, 2)	(2, 6, 16)
BF-GraphConv	Halfspace	**H**	76.5±0.9	88.0±1.0	86.8±0.9	80.6±0.4
			(1, 32, 2)	(2, 24, 2)	(2, 2, 2)	(4, 8, 2)
		X	76.3±1.8	88.0±1.1	86.5±0.4	80.5±0.5
			(2, 24, 2)	(1, 16, 2)	(2, 8, 4)	(2, 6, 2)
	Proto	**H**	76.2±1.2	**88.5±1.3**	86.1±0.2	81.4±0.8
			(2, 8, 2)	(2, 8, 1)	(2, 32, 4)	(2, 32, 2)
		X	74.9±0.6	87.8±1.1	86.5±0.3	**82.3±0.7**
			(2, 16, 2)	(2, 8, 4)	(2, 16, 4)	(2, 8, 16)

Table 1: Accuracy comparison between the Back-propagation-free models and standard models. The model selection is preformed considering the results obtained on the validation set. Under each result we report the hyper-parameters selected via validation process: (l, m, k) for the BF-GCN and BF-GraphConv and (l,k)

convolution operator. We empirically assessed the performances of BF-GCN and BG-GraphConv on four commonly adopted node classification benchmarks, and verified their competitive performances. Moreover, we analyzed the behavior of such models considering different options for shattering the context space associated to graph nodes. In our implementation, we use layer-wise training via Stochastic Gradient Descent (SDG), but many other methods can be exploited to solve the resulting convex problem, making it suitable for online and continual learning scenarios. Indeed, we plan to explore the application of the proposed backpropagation-free graph neural networks to continuous learning tasks in the near future.

References

1. T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017, citation Key: Kipf2016a. [Online]. Available: http://arxiv.org/abs/1609.02907
2. C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, Oct 2019, p. 4602–4609. [Online]. Available: http://arxiv.org/abs/1810.02244

3. J. C. Whittington and R. Bogacz, "Theories of error back-propagation in the brain," 2019.
4. D. G. Clark, L. F. Abbott, and S. Chung, "Credit assignment through broadcasting a global error vector," *arXiv:2106.04089 [cs, q-bio]*, Jun 2021, arXiv: 2106.04089. [Online]. Available: http://arxiv.org/abs/2106.04089
5. J. Veness, T. Lattimore, D. Budden, A. Bhoopchand, C. Mattern, A. Grabska-Barwinska, E. Sezener, J. Wang, P. Toth, S. Schmitt, and et al., "Gated linear networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 11, p. 10015–10023, May 2021.
6. A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.
7. T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *ICLR*, 2017, pp. 1–14.
8. M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016, pp. 3844–3852.
9. W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017, pp. 1024–1034.
10. Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated Graph Sequence Neural Networks," in *ICLR*, 2016. [Online]. Available: http://arxiv.org/abs/1511.05493
11. K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?" in *International Conference on Learning Representations*, 2019.
12. D. V. Tran, N. Navarin, and A. Sperduti, "On Filter Size in Graph Convolutional Networks," in *IEEE SSCI*. Bengaluru, India: IEEE, nov 2018, pp. 1534–1541. [Online]. Available: https://ieeexplore.ieee.org/document/8628758/
13. Z. Xinyi and L. Chen, "Capsule graph neural network," in *ICLR*, 2019.
14. J. Veness, T. Lattimore, D. Budden, A. Bhoopchand, C. Mattern, A. Grabska-Barwinska, E. Sezener, J. Wang, P. Toth, S. Schmitt, and M. Hutter, "Gated linear networks," *arXiv*, 9 2019. [Online]. Available: http://arxiv.org/abs/1910.01526
15. J. Veness, T. Lattimore, A. Bhoopchand, A. Grabska-Barwinska, C. Mattern, and P. Toth, "Online learning with gated linear networks," *arXiv*, pp. 1–40, 2017.
16. C. Mattern, "Linear and geometric mixtures - Analysis," *Data Compression Conference Proceedings*, pp. 301–310, 2013.
17. F. Wu, T. Zhang, A. H. de Souza, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying Graph Convolutional Networks," *ICML*, feb 2019.
18. N. Navarin, W. Erb, L. Pasa, and A. Sperduti, "Linear Graph Convolutional Networks," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2020.
19. M. Munari, L. Pasa, D. Zambon, C. Alippi, and N. Navarin, "Understanding catastrophic forgetting of gated linear networks in continual learning," 2022.
20. G. Aloupis, H. Pérez-Rosés, G. Pineda-Villavicencio, P. Taslakian, and D. Trinchet, "Fitting voronoi diagrams to planar tesselations," *arXiv:1308.5550 [cs]*, Aug 2013, arXiv: 1308.5550. [Online]. Available: http://arxiv.org/abs/1308.5550
21. P. Mernyei and C. Cangea, "Wiki-cs: A wikipedia-based benchmark for graph neural networks," *arXiv preprint arXiv:2007.02901*, 2020.
22. M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," in *ICLR 2019 (RLGM Workshop)*, Apr 2019.

A Model Selection

In all experiments, we select all the hyper-parameters on the validation set using the same fair model selection procedure. As for the model hyper-parameters (number of hidden units, number of layers, learning rate, weight decay, and dropout only for backpropagation-based models), their selection was based on a limited grid search, in which the explored sets of values were altered based on the dataset considered. We performed some preliminary tests in order to select the set of values taken into account for each hyper-parameter. In Table 2, we report the sets of hyper-parameter values used for the grid search. In order to perform a fair comparison among the proposed models and the baselines, we use the same hyper parameter grid for all the models. To ensure a fair comparison between baseline and the proposed model that adopts a one-vs-rest approach, for the baselines we also consider the model where the number of hidden is the values reported in Table 2 multiplied by the number of classes of the considered task. As evaluation measure to perform model selection, we used the average accuracy computed on the validation set, while we report in Table 1 the average accuracy on the test set.

hyper-parameter	**values**
m	$2, 4, 8, 16, 24, 32, 64$
l	$1, 2, 4, 6, 8$
k	$1, 2, 4, 8, 16$
learning rate	$0.1, 0.2, 0.01, 0.001$
weight decay	$0, 5^{-4}, 5^{-3}$
drop out	$0, 0.2, 0.5$

Table 2: Sets of hyper-parameters values used for model selection via grid search.