Topology preserving maps as aggregations for Graph Convolutional Neural Networks

ABSTRACT

In Graph Convolutional Neural Networks, the capability of learning the representation of graph nodes comes at hand when dealing with one of the many graph analysis tasks, namely the prediction of node properties. Furthermore, node-level representations can be aggregated to obtain a single graph-level representation and predictor. Such aggregator functions are essential to retain the most information about graph topology. This work explores an alternative route for the definition of the aggregation function compared to existing approaches. We propose a graph aggregator that exploits Generative Topographic Mapping (GTM) to transform a set of node-level representations into a single graph-level one. The integration of GTM in a GCNN pipeline allows to estimate node representation probability densities and project them in a low-dimensional space, while retaining the information about their mutual similarity. A novel dedicated training procedure is specifically designed to learn from these reduced representations instead of the complete initial data. Experimental results on several graph classification benchmark datasets show that this approach achieves competitive predictive performances with respect to the commonly adopted aggregation architectures present in the literature, while retaining a well-grounded theoretical framework.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

Graph Neural Network, Generative Topographic Mapping, Node Aggregation

ACM Reference Format:

. 2023. Topology preserving maps as aggregations for Graph Convolutional Neural Networks. In *Proceedings of ACM SAC Conference (SAC'23).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Graphs are an effective tool for the representation of entities and relations thereof for data coming from many application domains (e.g., chemistry, bioinformatics, social sciences). Many deep learning models for graphs have been developed in recent years. Actually, the first definition of neural networks for graphs has been proposed several years ago [20], while more recently Micheli [12] proposed a model exploiting an idea that has been re-branded later as *graph convolution*. Several different definitions of graph convolution have been proposed in the literature [8]. The core property of graph convolutions is that isomorphic graphs (i.e., graphs that represent the same relationship among nodes) should produce the same node representations. To date, there are no polynomial-time algorithms to decide if two graphs are isomorphic. Thus, this property has to be verified by design.

In the setting where the graph representation is exploited to represent samples (abstracted as nodes) that are not i.i.d., i.e. that are in relation one with each other (abstracted as edges), graph convolution is a powerful tool to generate node representations and node-level predictions. However, in the alternative, but not less common, setting in which each training example is represented as a distinct graph and the prediction has to be performed at the graph level (e.g., predicting properties of chemical compounds, each one represented as a different graph), another non-trivial representation issue arises: it is necessary to define an *aggregation* operator associating a single representation for the whole graph.

The definition of the aggregation function is not trivial for three main reasons: first, it has to map a variable number of node representations into a single (preferably fixed-size) graph-level one; second, it should be independent from the node ordering, that is it should be a graph invariant (isomorphic graphs should produce the same representation), and third, we would like the representations of similar graphs (e.g. a graph $G^{(1)}$ that is a subgraph of another graph $G^{(2)}$) to be similar.

The simplest approach, that is commonly adopted in literature, is to consider commutative *global* aggregation functions such as the element-wise sum, mean, or maximum. However, it has been shown [13] in that using such simple aggregations inevitably results in a loss of information, possibly impacting the predictive performance of the Graph Neural Network (GNN) architecture. More complex, non-linear aggregations have thus been proposed in literature [26].

Another approach consists in treating the node representations as elements belonging to an unordered set [21]

and to produce an order-invariant representation from them.

In this setting, Deep Sets [25] is a general framework to define a universal approximator of functions over sets, that has been adopted as graph aggregation [13]. It has been proved that, under some hypothesis, any function $f(X)$ over a set $X = \{x_1, \ldots, x_M\}, x_m \in \mathfrak{X}$ can be decomposed as $f(X) = \rho\left(\sum_{x \in X} \phi(x)\right)$ for suitable transformations $\rho(\cdot)$ and $\phi(\cdot)$. Implementing these functions as neural networks and learning them via backpropagation is a viable approach, but may lead to overfitting.

The SOM-based aggregator [17] implements the $\phi(\cdot)$ function of Deep Sets exploiting a self-organizing map (SOM) [9] to map the node representations in the space defined by the activations of the SOM neurons. The resulting representation embeds the information about the similarity between the various inputs. In fact, similar input structures will be mapped in similar output representations (i.e. node embeddings). The SOM is then followed by a

SAC'23, March 27 –April 2, 2023, Tallinn, Estonia

ACM ISBN 978-1-4503-9517-5/23/03…$15.00
https://doi.org/xx.xxx/xxx_x

Graph Convolution layer to partially incorporate the task supervision in the $\phi(\cdot)$ function. SOMs, however, suffer from some relevant drawbacks, such as the lack of an associated cost function and thus of a general proof of convergence. Because of that, it is also difficult to control the outcome of the learning process, which is driven by many heuristics requiring a careful setting of the hyperparameters, such as the shape of the function governing the width of the neighborhood used during training. While in general this may not be a practical issue, we show that in some cases, the SOM-based aggregation scheme exhibits performances that are below the state of the art.

In this paper, we address these issues by developing an alternative aggregation function $\phi(\cdot)$ that is based on a principled probabilistic model, namely the Generative Topographic Mapping (GTM) [1]. Specifically, by adopting this approach, we are able to have better control of the hyperparameters defining the projection of the node representations on the 2-dimensional GTM probabilistic latent space. This should make more effective the training procedure, leading to better identification of the node representations manifold, and consequently to more expressive graph-level hidden representations. In fact, contrarily to the SOM where only one winning neuron gets activated for the whole map for each input node (yielding to a global smoothing), the GTM grid of normal distributions enables for a coarser transformation that preserves local structures of the representation. These transformed representations are then exploited with a dedicated training procedure, on which various pooling techniques can be applied [10]. An additional feature of the proposed aggregation function is the amenability to a direct inspection of the internal representations of the model *that are used to produce the output*: the GTM latent space is organized in a 2-dimensional grid that can be easily plotted and whose corresponding values have a precise probabilistic meaning. Moreover, the internal representations are directly used by the model to produce the output and are obtained a posteriori by any dimensionality reduction method that inevitably produces artifacts. This constitutes an interesting base on which to develop GNN models that are interpretable by design.

We experimentally evaluate the GTM-based aggregation on seven graph classification tasks comparing it with other well established models and aggregation functions in the literature.

2 BACKGROUND

Throughout this work, we use italic letters to refer to variables, bold lowercase to refer to vectors, bold uppercase letters to refer to matrices, and uppercase letters to refer to sets or tuples. Let $G = (V, E, \mathbf{X})$ be a graph, where $V = \{v_0, \ldots, v_{n-1}\}$ denotes the set of n nodes (or vertices), $E \subseteq V \times V$ denotes the set of edges, and $\mathbf{X} \in \mathbb{R}^{n \times s}$ encodes the node attributes, namely its i^{th} row represents the features of node v_i. The set of nodes linked to node v_i, also known as neighborhood, is denoted as $\mathcal{N}(v_i)$.

2.1 Generative Topographic Mapping

The GTM algorithm [1] is a form of non-linear latent variable model which is based on a constrained mixture of Gaussians, whose parameters can be optimized using the EM (expectation-maximization) procedure [5]. Let us now provide a brief description of the GTM: given a dataset $\boldsymbol{\mathcal{X}}$ of N data points $\mathbf{x}_i \in \mathbb{R}^D$, the goal of a latent variable model is to find a representation for the distribution $p(\mathbf{x})$ of data in a D-dimensional space with respect to latent variables $\mathbf{u}$ embedded in a L-dimensional latent space, where $L \ll D$. A schematic illustration of a GTM's workings is provided in Fig 1The GTM is built by first introducing a regular grid of K nodes $\mathbf{u}_i$ in the latent space, labelled by the index $i = 1, 2, \ldots, K$, and a set of M fixed non-linear radial basis functions (RBF) $\boldsymbol{\phi}(\mathbf{u}) = \{\phi_j(\mathbf{u})\}$, with $j = 1, 2, \ldots, M$. Using the RBFs (the combination of RBFs is a good universal function approximator [16]), it is possibile to define a generalized linear regression model from the latent space to the data space, such that each point $\mathbf{u}$ in latent space is mapped to a corresponding point $\mathbf{y}$ in the D-dimensional data space $\mathbf{y}(\mathbf{u}, \mathbf{W}) = \mathbf{W}\boldsymbol{\phi}(\mathbf{u})$, where $\mathbf{W}$ is a $D \times M$ matrix of learnable weight parameters. In this fashion, each node $\mathbf{u}_i$ is projected to a D-dimensional reference vector $\mathbf{m}_i = \mathbf{W}\boldsymbol{\phi}(\mathbf{u})$, and if we set a prior distribution on the latent space nodes $p(\mathbf{u})$ this mapping will also induce a corresponding distribution in the data space $p(\mathbf{y}|\mathbf{W})$ confined in a L-dimensional manifold. Since in reality the dataset $\boldsymbol{\mathcal{X}}$ will only approximately lay on a lower-dimensional manifold, it is appropriate to include a noise model for the $\mathbf{x}$ vectors. Therefore, we assume that $\mathbf{x}$, for a given $\mathbf{u}$ and $\mathbf{W}$, is distributed as a radially-symmetric Gaussian centred on $\mathbf{y}(\mathbf{u}, \mathbf{W})$ and having variance β^{-1}:

$$p(\mathbf{x}|\mathbf{u}, \mathbf{W}, \beta) = \left(\frac{\beta}{2\pi}\right)^{D/2} \exp\left\{-\frac{\beta}{2}\|\mathbf{y}(\mathbf{u}, \mathbf{W}) - \mathbf{x}\|^2\right\}.$$

By marginalizing over $p(\mathbf{u})$

$$p(\mathbf{x}|\mathbf{W}, \beta) = \int p(\mathbf{x}|\mathbf{u}, \mathbf{W}, \beta) p(\mathbf{u}) d\mathbf{u}. \tag{1}$$

and by choosing the prior distribution $p(\mathbf{u})$ to be a superposition of delta functions located at the K nodes of the regular grid in latent space, (which is equivalent to say that the prior probabilities of each of the components is assumed to be constant and equal to $1/K$), the distribution in the data space can be expressed as $p(\mathbf{x}|\mathbf{W}, \beta) = \frac{1}{K}\sum_{i=1}^{K} p(\mathbf{x}|\mathbf{u}_i, \mathbf{W}, \beta)$.The posterior probabilities of the latent variables (or *responsibilities* R_i) given an input $\mathbf{x}$ can be computed by applying Bayes' theorem,and the final response as $R(\mathbf{x}; \mathbf{W}, \beta) = \sum_i p(\mathbf{u}_i|\mathbf{x}, \mathbf{W}, \beta)$.

Since the GTM represents a parametric probability density model, it can be fitted to the dataset $\boldsymbol{\mathcal{X}}$ by computing the optimal parameters $\mathbf{W}$ and β^{-1} via likelihood maximization. The log likelihood function is given by $\mathcal{L}(\mathbf{W}, \beta) = \sum_{n=1}^{N} \ln(p(\mathbf{x}_n|\mathbf{W}, \beta)) = \sum_{n=1}^{N} \ln\left\{\frac{1}{K}\sum_{i=1}^{K} p(\mathbf{x}_n|\mathbf{u}_i, \mathbf{W}, \beta)\right\}$,to which a regularization term can be added to reduce overfitting and improve convergence, e.g. by choosing a Gaussian prior over the weights governed by a hyperparameter $\lambda \in \mathbb{R}$ The maximization of the resulting loss function can be carried out by standard optimization techniques, but since we are dealing with a latent variable model a viable approach is to employ the well-established Expectation-Maximization algorithm [5]. Significant performance improvements in training can be achieved by updating the parameters incrementally using data in smaller batches [2], which is particularly suited for deep learning applications, and thus adopted in this paper.

Notice that for the particular noise model given by Eq. 1, the distribution $p(\mathbf{x}|\mathbf{W}, \beta)$ indeed corresponds to a *constrained* Gaussian

mixture model since the centres of the Gaussians, i.e. $\mathbf{y}(\mathbf{u_i}, \mathbf{W})$, cannot move independently but instead are adjusted indirectly through changes to the weight matrix $\mathbf{W}$. Besides, the projected points $\mathbf{m}_i$ will necessarily have a topographic ordering in the sense that any two points $\mathbf{u}_A$ and $\mathbf{u}_B$ which are close in the latent space are mapped to points $\mathbf{m_A}$ and $\mathbf{m_B}$ which are also close in the data space.

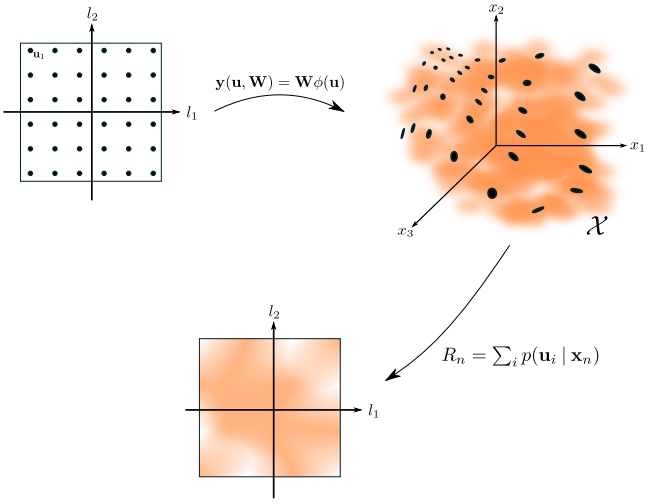

Figure 1: The GTM first considers a distribution of superposition of delta functions centered at K nodes of a regular array (left). Each node $\mathbf{u}_i$ is projected into the data space, where it becomes the center of a Gaussian distribution. Then, these projections are fitted to the data manifold $\mathcal{X}$ (center), and thanks to Bayes' Theorem, the posterior distribution in the latent space is retrieved (right).

3 GTM-BASED AGGREGATION FUNCTION

The Generative Topographic Mapping can be employed to effectively transform data from a high-dimensional space into a low-dimensional latent space while retaining the intrinsic properties of the dataset probability distribution $p(\mathbf{x})$. Additionally, the fact that GTM preserves the topological ordering guarantees that similar node representations are mapped into similar distributions in the lower-dimensional space. This method of feature extraction can be integrated into a GCNN pipeline since the GTM can be well exerted as unsupervised dimensionality reduction of the graph's node representations $\mathbf{h}_v$ before being aggregated. In this section, we describe an implementation of a GTM-based aggregation function for a GCNN, or briefly GTM-GCNN. Firstly we will focus on its architecture and components, then we will describe the dedicated training procedure to learn from labeled data.

3.1 Architecture

The proposed architecture of the GTM-GNN is made of three main components: a *Graph Convolutional* part, the GTM-based aggregator, and a *Readout* module for the final graph classification task, based on the work of [17].A graphical rendering of the architecture is reported in Figure 2.

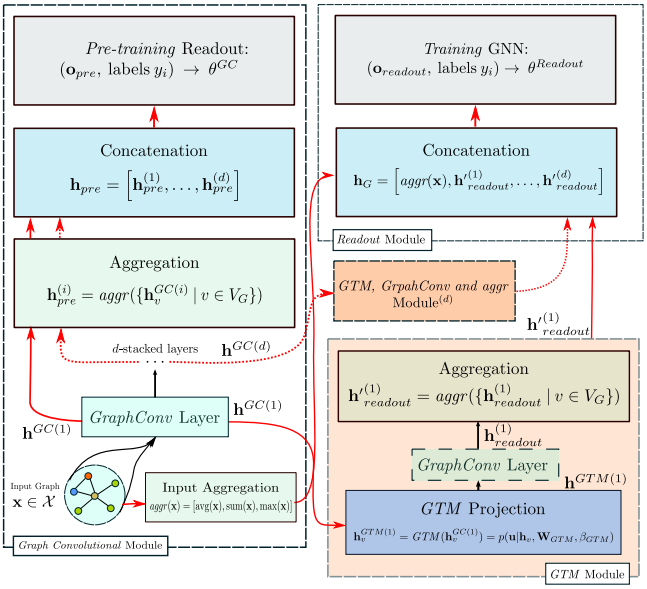

Figure 2: Graphical representation of the GTM-based GNN architecture.

First of all, an amount d of stacked graphs convolutional layers learn stable node representations from the input dataset $\mathcal{X}$. For this implementation we opted for *GraphConv* [4], due to its wide adoption and convincing performances, and we choose the *LeakyReLU* as activation function σ. All the d convolutional layers are followed by a batch normalization layer. We dubbed to output of the i^{th} graph convolutional layer as $h_v^{GC(i)}$. We refer to the learnable parameters of this initial *Graph Convolutional* part as $\boldsymbol{\theta}^{GC}$.

The enriched node embeddings $\mathbf{h}_v^{GC(i)}$ for each layer are the one-to-one input of d independent GTMs, that constitute the aggregator module. Recall that the representations $\mathbf{h}_v^{GC(i)}$ are vectors in a high-dimensional space, whose size is governed by the number of neurons of each *GraphConv* layer. Additionally, to improve numerical stability, these representations are further bounded in $[-1, 1]$ by applying the *hyperbolic tangent* function, i.e. $\hat{\mathbf{h}}_v^{GC(i)} = tanh(\mathbf{h}_v^{GC(i)})$. The GTM parameters $\boldsymbol{\theta}^{GTM} = \{\mathbf{W}_i, \beta_i\}$ are optimized via the EM algorithm and, once convergence has been reached, the GTMs are exploited to project the input vectors $\hat{\mathbf{h}}_v^{GC(i)}$ into the L-dimensional latent lattice, returning the posterior distribution $\forall v \in V_G$: $\mathbf{h}_v^{GTM(i)} = p(\mathbf{u}|\hat{\mathbf{h}}_v^{GC(i)}, \mathbf{W}_i, \beta_i)$. The components of $\mathbf{h}_v^{GTM(i)}$ are then further normalized to reduce the variability of node representations in the same graph, i.e., $\hat{\mathbf{h}}_v^{GTM(i)} = \mathbf{h}_v^{GTM(i)} / \xi_v^{GTM(i)}$, where $\xi_v^{GTM(i)}$ is the maximum value among the components of $\mathbf{h}_v^{GTM(i)}$. In the third module, called *Readout* and defined by the parameters $\boldsymbol{\theta}^{Readout}$, each $\hat{\mathbf{h}}_v^{GTM(i)}$ is fed through another *GraphConv* layer (so that the graph topology is brought back). Then, these transformed representations are aggregated by taking the concatenation of their average, sum, and component-wise maximum (this idea was introduced by the [24]). At the end, all d feature maps are concatenated to obtain one single graph-level representation $\mathbf{h}_G$ ready for the output layer

and the supervised learning task, achieved by means of a MLP with *LogSoftmax* output function.

3.2 Training Procedure

To conciliate the unsupervised framework of the GTMs with the supervised task of graph classification, the training of the GTM-based GNN takes four steps that are carried out one after the other, optimizing in each turn different sets of learnable parameters.

The first training step consists in optimizing the parameters $\boldsymbol{\theta}^{GC}$ by adding an ad-hoc readout layer, which we indicate as *pre-training* readout, to perform supervised learning with standard backpropagation. In other words, this allows training of this part of the model separately from the rest of the network. This pre-training readout layer further aggregates the node representations by taking the concatenation of their average, sum, and component-wise maximum.Then, it stacks these vectors for all the d layers and applies a linear transformation and the log softmax activation function in the end. The pre-training readout layer is discarded and thus will not make part of the final model.

Next, the parameters $\boldsymbol{\theta}^{GTM}$ of the GTMs are initialized using the first two principal components of the node representations PCA [1], and later optimized via the EM algorithm. Then, the parameters $\boldsymbol{\theta}^{Readout}$ are optimized via backpropagation with reference to the negative log-likelihood loss on $\mathbf{o}_{readout}$ for the c-class graph classification.

Finally, the last training step consists of a *fine-tuning* phase. The purpose of this step is to tune the model parameters $\boldsymbol{\theta}^{GC}$ and $\boldsymbol{\theta}^{Readout}$ while maintaining the $\boldsymbol{\theta}^{GTM}$ fixed. The pseudo-code that summarizes the training procedure is reported in Algorithm 1.

Algorithm 1 GTM-based GNN Training Procedure

Input: Graphs Dataset $\mathcal{X}$ with associated labels y

$\boldsymbol{\theta}^{GC}, \mathbf{o}_{pre} \leftarrow Pretraining(\mathcal{X}, y)$ ▷ Pre-training of *GraphConv* Module

$\forall$ stacked layer $i : \mathbf{h}^{GC(i)} \leftarrow f\Big(GraphConv\Big(\mathbf{h}_v^{GC(i-1)}\Big)\Big)$ ▷ Get stable representations from pre-training module

$\boldsymbol{\theta}^{GTM(i)} \leftarrow GTM\_training\Big(tanh\Big(\mathbf{h}^{GC(i)}\Big)\Big)$ ▷ Expectation Maximization

$\mathbf{h}^{GTM(i)} \leftarrow GTM\Big(tanh\Big(\mathbf{h}^{GC(i)}\Big), \boldsymbol{\theta}^{GTM}\Big)$ ▷ Projected representations

$\forall v : \hat{\mathbf{h}}_v^{GTM(i)} \leftarrow \frac{\mathbf{h}_v^{GTM(i)}}{\xi_v^{GTM(i)}}$, where $\xi_v^{GTM(i)}$ is the maximum of $\mathbf{h}_v^{GTM(i)}$ components

$\forall i : \mathbf{h}^{(i)}_{readout} \leftarrow f\Big(GraphConv\Big(\hat{\mathbf{h}}_v^{GTM(i)}\Big)\Big)$ ▷ *Readout* Module

$\forall i : \mathbf{h'}^{(i)}_{readout} \leftarrow aggr\Big(\Big\{\mathbf{h}^{(i)}_{readout} \mid v \in V_G\Big\}\Big)$ ▷ Aggregation

$\mathbf{h}_G \leftarrow \Big[\text{avg}(\mathbf{X}), \text{sum}(\mathbf{X}), \max(\mathbf{X}), \mathbf{h'}^{(1)}_{readout}, \ldots, \mathbf{h'}^{(d)}_{readout}\Big]$ ▷ Concatenation

$\mathbf{o}_{readout} \leftarrow f(MLP(\mathbf{h}_G))$

$\boldsymbol{\theta}^{readout} \leftarrow LogSoftMax(\mathcal{X}, y; \mathbf{o}_{readout})$ ▷ Readout training

$\boldsymbol{\theta}^{GC}, \boldsymbol{\theta}^{readout} \leftarrow FineTuning(\mathcal{X}, y; \boldsymbol{\theta}^{GC}, \boldsymbol{\theta}^{readout})$ ▷ Fine-Tuning training

Output: GTM-based GNN$(\boldsymbol{\theta}^{GC}, \boldsymbol{\theta}^{GTM}, \boldsymbol{\theta}^{readout})$

4 EXPERIMENTAL RESULTS

In this section we present our model setup, then we report and discuss the results obtained by the GNN that exploits the proposed GTM-based aggregation.

4.1 Setup and Hyperparameters

As already mentioned, the GTM-GNN is made of three main parts, i.e the *pre-training* section, the GTM-based aggregator, and the *Readout* module. For what concerns the first *pre-training* module, we set at $d = 3$ the number of hidden layers and select *GraphConv* as convolutional operator. In relation to the relative size of these layers, we opted for a "funnel" architecture [14] in the sense that the *GraphConv* layers have an increasing number of neurons, namely $\mathbf{h}^{GC(1)} \in \mathbb{R}^l$, $\mathbf{h}^{GC(2)} \in \mathbb{R}^{2l}$ and $\mathbf{h}^{GC(3)} \in \mathbb{R}^{3l}$, where the size l is an hyperparameter. This architecture has been proved to improve performances and therefore it is adopted in the GTM-GCNN. Both the pre-training and the Readout module are trained via backpropagation using the *AdamW* optimizer [11].

The goal of this work is to evaluate the benefit of using the GTM-based aggregation, therefore we focused our attention on the behavior of the GTM parameters. For all GTMs, we set the latent variable dimension to $L = 2$, so that the K latent variables $\mathbf{u}_i$ lay in a bi-dimensional plane. Both their amount in the *width* and the *height* dimensions of the regular grid are hyperparameters (in this way $K = height \times width$), and the grid itself is build accordingly within a bounded $[-1, 1] \times [-1, 1]$ plane when the GTMs are initialized. Other two relevant hyperparameters concern the Radial Basis Functions $\boldsymbol{\phi}(\mathbf{u})$, namely their amount M and variance σ. The former is used to form a $M \times M$ regular grid of RBF center points, that is overlayed to the latent variables grid in the $[-1, 1]^2$ plane. On the other hand, the variance σ can be tested for any value or can be computed as the average minimum distance among the aforementioned RBF centers. Finally, as soon as the latent nodes grid and RBF function are set, the matrix $\boldsymbol{\Phi}_{ij} = \phi_j(\mathbf{u}_i)$ is computed and we pad a bias column of $\mathbf{1}$ to it. Notice that this step is done only once at initialization. The parameters $\mathbf{W}$ and β can be either initially set at random from the standard normal distribution $\mathcal{N}(\mu = 0, \sigma = 1)$, or as explained beforehand they are computed as to mimic the PCA applied to the whole training set. To do this, before the first epoch of the EM algorithm, the whole dataset is loaded into memory and the PCA is performed. We avoided the random initialization since it can be numerically unstable and it takes longer for convergence. This step is also needed to determine the right size of the responsibility matrix R_{in} that is updated from the first epoch with the incremental learning. The last hyperparameter is the regularization constant λ, which can take any fixed value or be equal to β^{-1}.

After the EM optimization, the posterior distribution of the input data is estimated and it is scaled by its maximum value $\xi_v^{GTM(i)}$ before being fed to the next *GraphConv* layer, so that the values are bounded in $[0, 1]$, restricting the learning of their relative scale on the lattice grid rather than absolute magnitude (being unnormalized probabilities). Eventually, the *Readout* module concatenates the three *GraphConv* outputs of the same fixed size l and supplies them to a MLP, whose depth q is also a hyperparameter.

The GTM-based GNN has been implemented with `Python 3.8.8` and `PyTorch 1.8.1` [18], an open source and multi-purpose machine learning framework. We adopted 2 types of machines, respectively equipped with: 2 x Intel(R) Xeon(R) CPU E5-2630L v3, 192GB of RAM, a Nvidia Tesla V100 and 2 x Intel(R) Xeon(R) CPU

E5-2650 v3, 160 GB of RAM, Nvidia T4. For all the other hyperparameters and implementation details please check the publicly available code[1].

4.2 Model Selection

To select the best hyperparameter combination we run 10-fold cross-validation for each dataset. Due to the long time requirements of performing an extensive grid search, we decided to limit the number of values taken into account for each hyperparameter and we performed a random search over the grid of their combination.

Table 1 gives an overview of the arbitrarily chosen values of the GTM hyperparameters grid. Each one of the four training phases runs for 500 epochs and moreover, to reduce overfitting on the training set, we adopted a validation-based *early stopping* regularization that chooses the epoch of the best performing model on the validation set, stopping the training if after 25 epochs no better result is achieved. For what concerns the GTMs, we use the validation loss, i.e. the complete-data log-likelihood, to monitor the convergence and early stopping.

Hyperparameter	List of values
Latent var. grid	(10×15), (11×16), (13×18), (15×20), (20×25)
Amount of RBF M	8, 12, 18
Variance of RBF σ	s, $2s$, 1
GTM Reg. λ	10, 1, 0, 0.1, 0.01, β^{-1}
MLP depth q	1, 3, 5
Hidden neurons l	20, 30, 50

Table 1: Hyperparameter grid for the random search cross validations. Recall that s is the average spacing among RBF centers, e.g. $\sigma = 0.167$ for the (15×10) grid.

4.3 GNN Models Employed as Baselines

We compare the GTM-GCNN with several GNN architectures which achieved state-of-the-art results on the used datasets. In the following, we describe the models considered for the experimental comparison. The first model that we consider in our experimental comparison is the PSCN proposed by Niepert *et al.* [15]. PSCN follows a straightforward approach to define convolutions on graphs, that is conceptually closer to convolutions defined over images. First, it selects a fixed number of vertices from each graph, exploiting a canonical ordering on graph vertices. Then, for each vertex, it defines a fixed-size neighborhood (of vertices possibly at distance greater than one), exploiting the same ordering. This approach requires to compute a canonical ordering over the vertices of each input graph, that is a problem as complex as the graph isomorphism (no polynomial-time algorithm is known).

GraphSage [7] does modify the standard definition of graph convolution empowering the aggregation over the neighborhoods by using sum, mean or max-pooling operators, and then performs a linear projection in order to update the node representations. in addition to that, it exploits a particular neighbors sampling scheme. The convolution proposed in [7] has been extended by GIN [22], which introduces a more expressive aggregation function on multi-sets with the aim to overtake the limitation introduced by GraphSAGE using sum, mean or max-pooling operators.

DGCNN [26] extends the GCN proposed by Kipf *et al.* [8] introducing a slightly different propagation scheme for vertices' representations based on random-walks on the graph, and exploiting SortPooling as aggregation function. An extension of this model that exploits the DeepSet (DGCNN-DeepSet) was proposed by Tran *et al.* in 2019 [13].

DiffPool [24] is pooling operator that leverages on hierarchical properties of the graph structure by learning a clustering module that makes the graph more and more coarse at every layer. In particular, it learns a new adjacency matrix for each layer where single nodes can be substituted by clusters (thus the size of the matrix becomes smaller at deeper layers).

The Funnel GCNN (FGCNN) model [14] relies on the similarity of the adopted graph convolutional operator to the way the features of the Weisfeiler-Lehman (WL) Subtree Kernel [19] are computed. Based on this observation, a novel WL-based loss term for the output of each convolutional layer is introduced to guide the network to reconstruct the corresponding explicit WL features. FGCNN also adopts a number of filters at each convolutional layer determined by a measure of the WL-kernel complexity.

4.4 Discussion

In Table 2, we report the results achieved by the GNNs when the comparison among them is fair, i.e the same validation strategy and the common settings for the input datasets are employed. The issue of experimental reproducibility and replicability in the field of GNN is crucial and therefore we hold as baseline only the fair results that are reported in the literature [6]. The results reported in Table 2 were obtained by performing 5 runs of 10-fold cross-validation. The results reported in [3, 22, 23] are not considered in our comparison since the model selection strategy is different from the one we adopted and this makes the results not comparable.

The results reported in Table 2 show that the GTM-GCNN achieved highly competitive performance in all considered datasets. In particular, on PTC, D&D and IMDB-B the proposed method obtained higher results compared to the state-of-the-art, while in NCI1, PROTEINS, ENZYMES and IMDB-M the accuracy results are higher than the ones achieved by most of the models considered in the comparison. On NCI1 and IMDB-M the GTM-GCNN shows the second-best performance, and only the SOM-GCNN performs better than our proposed model. On PROTEINS, the accuracy reached by the GTM-GCNN is lower than the ones obtained by many of the other considered models. The hyperparameter values selected in this case are very different than in the ones selected on the other datasets. Indeed, the selected model is the simpler considered in our experimental assessment ($l = 20$, $q = 1$). Specifically, 20 is the smallest value for l that we considered during the validation process. It is likely that by using smaller values for l the GTM-GCNN could reach better performances and avoid overfitting. Additionally, this dataset has an higher average degree (3.73, see Table 3) compared with NCI1 (2.16) and PTC (2.06); we argue that, compared with the other two datasets that have graphs sizes of the same magnitude, it

[1] omitted to double blind policy

Model/Dataset	PTC	NCI1	PROTEINS	D&D	ENZYMES	IMDB-B	IMDB-M
PSCN [15]	60.00±4.82	76.34±1.68	75.00±2.51	76.27±2.64	-	71±2.29	45±2.84
FGCNN [13]	58.82±1.80	81.50±0.39	74.57±0.80	77.47±0.86	-	-	-
DGCNN [13]	57.14±2.19	72.97±0.87	73.96±0.41	78.09±0.72	-	-	-
DGCNN [6]	-	76.4±1.7	72.9±3.5	76.6±4.3	38.9±5.7	53.3±5.0	38.6±2.2
GIN [6]	-	80.0±1.4	73.3±4.0	75.3±2.9	**59.6±4.5**	66.8±3.9	42.2±4.6
DIFFPOOL [6]	-	76.9±1.9	73.7±3.5	75.0±3.5	59.5±5.6	69.3±6.1	45.1±3.2
GraphSAGE [6]	-	76.0±1.8	73.0±4.5	72.9±2.0	58.2±6.0	69.9±4.6	47.2±3.6
DGCNN-DeepSets [13]	58.16±1.05	74.19±0.59	75.11±0.28	77.86±0.27	-	-	-
SOM-GCNN [17]	62.24±1.7	**83.30±0.45**	**75.22±0.61**	78.10±0.60	50.01±2.92	67.65±1.99	**48.68±3.46**
GTM-GCNN	**62.49±9.60**	82.48±1.33	72.88±4.82	**78.27±3.63**	59.03±5.92	**72.33±3.89**	47.69±4.44
GTM-GCNN w/ Ablation	61.95±8.27	82.28±2.12	73.86±4.74	76.70±3.47	58.72±7.02	71.67±3.56	47.78±3.9
GTM-GCNN Hyperparameters	(15×20) $q = 1$ $\lambda = 0.01$ $l = 30$ $M = 12$	(15×20) $q = 5$ $\lambda = 0.1$ $l = 50$ $M = 12$	(11×16) $q = 1$ $\lambda = 0.01$ $l = 20$ $M = 12$	(12×17) $q = 3$ $\lambda = 0.1$ $l = 50$ $M = 12$	(15×20) $q = 3$ $\lambda = 0.1$ $l = 50$ $M = 12$	(11×10) $q = 1$ $\lambda = 0.1$ $l = 20$ $M = 12$	(15×20) $q = 1$ $\lambda = 0.1$ $l = 30$ $M = 12$

Table 2: Accuracies of GTM-GCNN and *state-of-the-art* models on the seven used datasets. Values for the selected latent variable grid size, depth of the readout MLP q, regularization parameter λ, amount of hidden neurons l and number of RBF M are reported.

could be more difficult to grasp the local features that differentiate between the two classes. Overall, the GTM-CGNN exhibits higher accuracy variances due to varying performances on each CV split. We also argue that, being a probabilistic model, randomness plays a major role in the GTM component. Nevertheless, we recall that this probabilistic framework is theoretically well-founded and more research can be done to exploit its characteristics.

Dataset	#Graphs	#Node	#Edge	Avg. #Nodes	Avg. #Edges	Avg. Degree	#Classes
PTC	344	4915	10108	14.29	14.69	2.06	2
NCI1	4110	122747	265506	29.87	32.30	2.16	2
PROTEINS	1113	43471	162088	39.06	72.82	3.73	2
D&D	1178	334925	1686092	284.32	715.66	5.03	2
ENZYMES	600	19580	74564	32.63	124.27	3.81	6
IMDB-B	1000	19773	193062	19.773	193.06	9.76	2
IMDB-M	600	19502	197806	13.00	131.87	10.14	3

Table 3: Datasets statistics.

4.5 Ablation Study

To investigate further the benefits of the GTM aggregation, we analyzed the case of the removal of the GraphConv layer after the aggregation that was originally inserted to restore the graph topology. The results of this ablation study are reported in Table 2. We can see that albeit the dismissal of a layer, the predictive performances do not show any significant difference compared to the full GTM-GCNN architecture. Therefore, removing the last GC layer will be helpful in reducing the complexity of the GTM module. Moreover, it reduces the number of parameters and hyper-parameters that have to be optimized.

5 SOM VS. GTM

The similarity between the GTM-GCNN and SOM-GCNN makes the comparison between these two models interesting in evaluating the impact of the proposed GTM-based graph aggregator. From this perspective, it is worth noticing that the drop of accuracy on NCI1, PROTEINS, and IMDB-M is limited, while in ENZYMES the difference between SOM-GCNN and GTM-GCNN models is considerable. Indeed, GTM-GCNN improves the SOM-GCNN performance by almost 9 percentage points.

We argue that the higher GTM results may be explained by: *(i)* its training being more theoretically grounded then the one exploited by the SOM—GTM optimization is based on the maximization of a likelihood function that can be carried out by standard optimization techniques such as the well-established Expectation-Maximization algorithm [5]; *(ii)* being able to represent more complex manifolds; it results are more suitable in managing multi-class classification tasks because it may be easier for the GTM to encode the differences in the data distributions of the various classes compared to SOM (see Section 5.1); *(iii)* the model is easier to setup since it does not require to define a neighborhood function with its respective hyper-parameters [17].

5.1 Lattice representations

In order to investigate the reason for the GTM performance improvement compared to SOM on the ENZYEMS dataset, in Figure 3 we plot the heatmaps of the respective lattice representations. The heatmaps were computed following the same procedure proposed in [17]. Each heatmap shows the average value of each neuron in the lattice (either SOM or GTM), computed over the set of graphs belonging to the same class. Thus, each heatmap represents, for each class, the average level of utilization of the different parts of the lattice, meaning that parts that are used by a single class represent discriminative areas for that class. The comparison shows that the GTM tends to create a more distributed pattern of specific areas.

Given the higher accuracy obtained by the GTM, it is clear that the learned node representations benefit from the greater expressiveness and local discriminative power of the GTM in comparison with the SOM. The better representations obtained by the GTM are also due to lower sensibility of the GTM to the values of the hyperparameters, in comparison with SOM. Indeed, as reported in Table 2, the selected latent space dimensions are similar regardless of the complexity of the considered dataset/task. These interesting features, related to the probabilistic definition of the GTMs, help also in having an effective training phase.

5.2 End-to-end Fine Tuning

Arguably, one of the biggest downsides of having a layer trained in an unsupervised way for a supervised task, such as the SOM-GCNN, is that it is not possible to train the overall network using end-to-end backpropagation.

In this section, we show that the proposed GTM-based aggregation can be also trained in an end-to-end fashion. While the aforementioned training procedure is reasonable—given the unsupervised nature of the SOM and GTM maps—thanks to the specific formulation of the GTM it is also possible to train the map using the gradient provided by the subsequent layers.

We thus propose to add a further step in the training procedure, where we fine-tune all the parameters of the network using standard backpropagation. The only modification we need to introduce for GTM is well-known when training probabilistic models using stochastic gradient descent, that is we have to be careful in re-normalizing the output of the GTM to ensure it remains a probability distribution. Notice that it is not possible to pursue this approach with the SOM-based formulation since it relies on an *arg max* operation that is not differentiable. Therefore, modifying the SOM formulation to exploit only differentiable operations would require to define a novel optimization algorithm since the original one is not based on the optimization of a likelihood function.

We applied this end-to-end fine-tuning on the ENZYMES dataset to see if it could improve the performances of our proposed model even more. With no additional hyper-parameter exploration, we started the fine-tuning process from the "optimal" hyper-parameters, obtaining an accuracy on the ENZYMES dataset of 59.9±7.1, where the results before end-to-end fine-tuning as reported in Table 2 are 60.89±6.66. Even though such improvement is not statistically significant, we can see a slight increase in accuracy in all the 5 runs thanks to the class-based supervision that is provided in this last step to the GTM representations, that are not fully unsupervised anymore.

This preliminary result suggests that a viable alternative strategy for training the GTM-based architecture can be to include the GTM likelihood directly in the loss function and to perform a single end-to-end training phase. We will explore this strategy in a future work.

6 CONCLUSIONS AND FUTURE WORKS

In this paper, we addressed the problem of defining a more effective node aggregation function for Graph Neural Networks. Specifically, inspired by the work proposed in [17], where the authors introduced a SOM-based graph aggregator, we developed a novel node aggregation function based on a principled probabilistic model, i.e. Generative Topographic Mapping [1], that owns several nice advantages over SOM: *i)* training optimizes a well-defined cost function; *ii)* a smaller hyperparameter space to explore for model selection; *iii)* experimentally showed to return richer dimensionality reduction mappings, thus increasing the expressiveness of the node aggregation function that can be obtained in practice. In addition to the above advantages, the proposed approach opens the door to more interpretable GNNs since the internal 2-dimensional representations used to generate the output can be directly visualized for inspection in 2-D heatmaps. This comes without compromising the performance of the model, as clearly shown by the reported state-of-the-art empirical results on seven graph-level classification tasks by a GNN exploiting the GTM-based aggregation function.

Interestingly, the GTM-based aggregation operator shows a significant improvement in performance on multi-class problems, in comparison to the closest competitor, the SOM-GCNN. Finally, we show how the adoption of the GTM aggregator enables the possibility to train the whole model in an end-to-end fashion. We exploited this option as a finale fine-tuning step, but this approach will be further investigated obtaining an aggregation layer that does not require any specific training procedure.

In the future, we plan to study how to further exploit the probabilistic representation computed by the GTM, with the aim to improve the interpretability of the model, and we plan to test its performances on other multi-class datasets present in the literature.

REFERENCES

[1] Christopher M Bishop, Markus Svensén, and Christopher KI Williams. 1998. GTM: The generative topographic mapping. *Neural computation* 10, 1 (1998), 215–234.

[2] Christopher M. Bishop, Markus Svensén, and Christopher K.I. Williams. 1998. Developments of the generative topographic mapping. *Neurocomputing* 21, 1 (1998), 203–224.

[3] Ting Chen, Song Bian, and Yizhou Sun. 2019. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv preprint arXiv:1905.04579* (2019).

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems* 29 (2016).

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39, 1 (1977), 1–38.

[6] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. 2020. A Fair Comparison of Graph Neural Networks for Graph Classification. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HygDF6NFPB

[7] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 1025–1035.

[8] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*. 1–14. https://doi.org/10.1051/0004-6361/201527329 arXiv:1609.02907

[9] Teuvo Kohonen. 1982. Self-organized formation of topologically correct feature maps. *Biological cybernetics* 43, 1 (1982), 59–69.

[10] Chuang Liu, Yibing Zhan, Chang Li, Bo Du, Jia Wu, Wenbin Hu, Tongliang Liu, and Dacheng Tao. 2022. Graph Pooling for Graph Neural Networks: Progress, Challenges, and Opportunities. *arXiv preprint arXiv:2204.07321* (2022).

[11] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. arXiv:1711.05101 [cs.LG]

[12] Alessio Micheli. 2009. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks* 20, 3 (2009), 498–511.

[13] Nicolò Navarin, Dinh Van Tran, and Alessandro Sperduti. 2019. Universal Readout for Graph Convolutional Neural Networks. In *International Joint Conference on Neural Networks*. Budapest, Hungary.

[14] Nicolò Navarin, Dinh Van Tran, and Alessandro Sperduti. 2020. Learning kernel-based embeddings in graph neural networks. In *ECAI 2020*. IOS Press, 1387–1394.

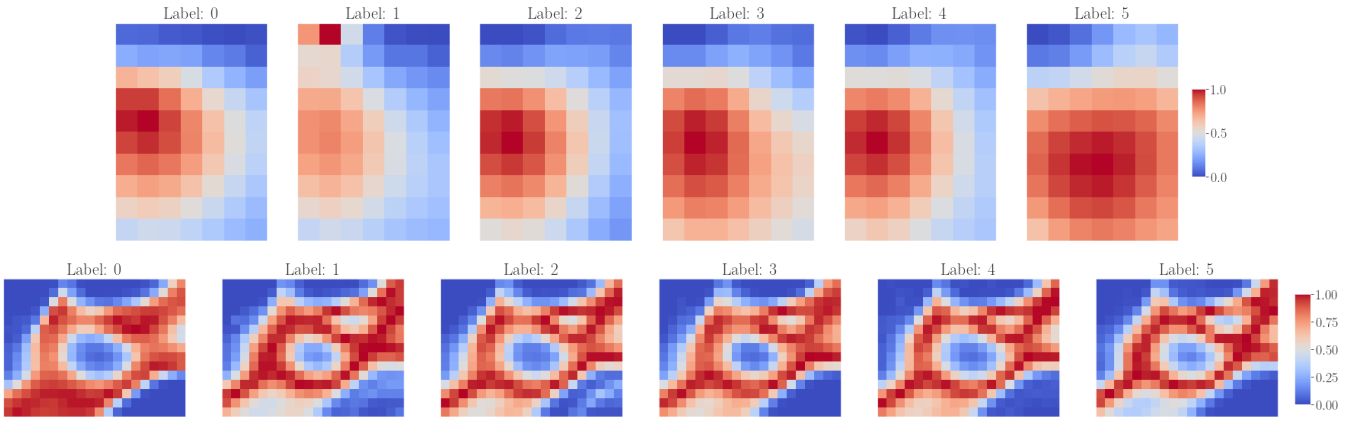

Figure 3: Heatmaps of first-level SOM (top row) and GTM (bottom row) projected representations for the multi-class dataset ENZYMES. Heatmaps at higher levels are similar. Each heatmap is obtained by averaging the contribution of several graphs belonging to the corresponding class.

[15] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*. 2014–2023.

[16] J. Park and I. W. Sandberg. 1991. Universal Approximation Using Radial-Basis-Function Networks. *Neural Computation* 3, 2 (1991), 246–257.

[17] Luca Pasa, Nicolò Navarin, and Alessandro Sperduti. 2020. SOM-based aggregation for graph convolutional neural networks. *Neural Computing and Applications* (2020), 1–20.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035.

[19] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, 77 (2011), 2539–2561.

[20] Alessandro Sperduti and Antonina Starita. 1997. Supervised neural networks for the classification of structures. *IEEE Trans. Neural Networks* 8, 3 (1997), 714–735. https://doi.org/10.1109/72.572108

[21] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2015. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391* (2015).

[22] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?

[23] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804* (2018).

[24] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2019. Hierarchical Graph Representation Learning with Differentiable Pooling.

[25] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. 2017. Deep sets. *arXiv preprint arXiv:1703.06114* (2017).

[26] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.