Generative Topographic Mapping Based Aggregation Function for GCNN

Paolo Frazzetto[1], Luca Pasa[1], Nicolò Navarin[1], and Alessandro Sperduti[1]

Department of Mathematics, University of Padua, Padua, Italy
paolo.frazzetto@phd.unipd.it, {luca.pasa, nicolo.navarin, alessandro.sperduti}@unipd.it

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.

Keywords: Graph Neural Network, Generative Topographic Mapping, Node Aggregation

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). Deep learning has shown astounding results on tasks for non-structured data, such as image classification, so it is not surprising that 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 [24], while more recently Micheli [16] proposed a model exploiting an idea that has been re-branded later as *graph convolution*. In a nutshell, a *graph convolution* layer receives in input a representation for each node in a graph and, similarly to convolutions defined for regular topologies (e.g., images) it computes a new representation for each node that also considers its local neighborhood, i.e., the neighboring nodes. Following this idea, several different definitions of graph convolution have been proposed in the literature [12]. 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 [17] 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 [31].

Another approach consists in treating the node representations as elements belonging to an unordered set [25] and to produce an order-invariant representation from them. In this setting, Deep Sets [30] is a general framework to define a universal approximator of functions over sets, that has been adopted as graph aggregation [17]. It has been proved that, under some hypothesis, any function $f(X)$ over a set $X = \{x_1, \dots, x_M\}, x_m \in \mathfrak{X}$ 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 [21] implements the $\phi(\cdot)$ function of Deep Sets exploiting a self-organizing map (SOM) [13] 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 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.

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 [14]. An additional feature of the proposed aggregation function is the amenability to directly inspect 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 five graph classification tasks comparing it with other well-established aggregation functions in the literature. Moreover, we show examples of plots of the GTM-based internal representations in a multi-class graph classification task.

2 Notation

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 $\mathcal{N}(v_i)$.

3 Background

Broadly speaking, developing a GNN for graphs classification requires to implement three functions: a *operator* on graph-structured data, an *aggregator* and a *readout* function. The graph *operator* exploits the graph topology and signal $G = (V, E, \mathbf{X})$ to create the nodes representations $\mathbf{h}_v$. Then, since graphs do not usually have all the same size, an *aggregation* procedure is needed to gather all the node representations $\mathbf{h}_v$ and obtain a fixed-size graph-level one—$\mathbf{h}^G$. Eventually, a *readout* function translates $\mathbf{h}^G$ into the output label $\mathbf{o}$, whose error with reference to its true value can be estimated through any loss function. The aforementioned functions are parameterized by a set of learnable parameters $\mathbf{\Theta}$ that can be optimized, for instance, by minimizing the loss function via backpropagation [9]. In the scope of this work, we can formally illustrate a general GNN by means of the following equations. First, d stacked layers perform a non-linear transformation of node representations, considering the local graph topology:

$$\mathbf{h}_v^{(i)} = f\Big(\mathit{graph\_operator}\Big(\mathbf{h}_v^{(i-1)}, \Big\{\mathbf{h}_u^{(i-1)} \mid u \in \mathcal{N}(v)\Big\}\Big)\Big) \qquad 1 \leq i \leq d,$$

where $f(\cdot)$ is an element-wise non-linear activation function, $\mathit{graph\_operator}(\cdot,\cdot)$ is a operator on graphs, $\mathbf{h}_v^{(i)}$ is the representation of node v at the i^{th} layer and $\mathbf{h}_v^{(0)} = \mathbf{x}_v$ is the input signal for node v. Stacking these operator layers allows for increasing the discriminatory power of the network since the representation of a node v is influenced by all the nodes up to distance d from v. Then, the aggregator that takes as input all the node representations can be written down as $\mathbf{h}^G = \mathit{aggr}\Big(\{\mathbf{h}_v^{(i)}\ \forall\, v \in V,\ 1 \leq i \leq d\}\Big)$. The $\mathit{aggr}(\cdot)$ further applies a non-linear function to $\mathbf{h}_v^{(i)}$ that aggregates all the node representations. Finally, another function reads out the graph representation and then an output layer is applied to obtain the output, e.g. class probabilities for a c-class classification problem can be obtained by $\mathbf{o} = \mathrm{LogSoftMax}(\mathbf{h}^S)$, where $\mathbf{h}^S = \mathit{readout}(\mathbf{h}^G)$.

3.1 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 [6]. 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 1, whereas for more detailed information we refer the reader to Appendix A.1 and the original papers [1, 2].

4 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

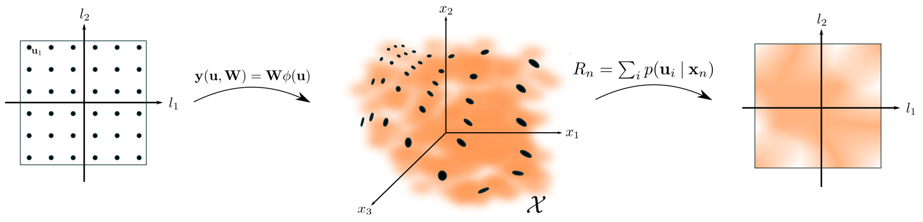
l_2
$\mathbf{u}_1$
l_1
$\mathbf{y}(\mathbf{u}, \mathbf{W}) = \mathbf{W}\phi(\mathbf{u})$
x_2
x_1
x_3
$\mathcal{X}$
$R_n = \sum_i p(\mathbf{u}_i \mid \mathbf{x}_n)$
l_2
l_1

Fig. 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).

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.

4.1 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, loosely based on the work of [21]. Let us now illustrate in detail each component of the model. A graphical rendering of the architecture is reported in Appendix A.2.

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* [5], 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. Thus this step can be formally written as

$$\mathbf{h}_v^{GC(i)} = \sigma\Big(GraphConv\Big(\mathbf{h}_v^{GC(i-1)}, \Big\{\mathbf{h}_u^{GC(i-1)} \mid u \in \mathcal{N}(v)\Big\}\Big)\Big), \quad (1)$$

for $1 < i \leq d$, while the first layer directly acts on the input data. 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. This enables all the layer-wise information for different receptive fields to be propagated through the network. 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 $\forall v \in V_G$ (see Eq. 7):

$$\mathbf{h}_v^{GTM(i)} = GTM_i\left(\hat{\mathbf{h}}_v^{GC(i)}\right) = p(\mathbf{u}|\hat{\mathbf{h}}_v^{GC(i)}, \mathbf{W}_i, \beta_i). \tag{2}$$

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 [29]). 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.

$$\begin{aligned}
\mathbf{h}_{readout}^{(i)} &= \sigma\Big(\mathit{GraphConv}\Big(\hat{\mathbf{h}}_v^{GTM(i)}, \Big\{\hat{\mathbf{h}}_u^{GTM(i)} \mid u \in \mathcal{N}(v)\Big\}\Big)\Big), \\
\mathbf{h'}_{readout}^{(i)} &= \mathit{aggr}\Big(\Big\{\mathbf{h}_{readout}^{(i)} \mid v \in V_G\Big\}\Big) = \\
&= \Big[\mathrm{avg}_{v\in V_G}\Big(\mathbf{h}_{readout}^{(i)}\Big), \mathrm{sum}_{v\in V_G}\Big(\mathbf{h}_{readout}^{(i)}\Big), \mathrm{max}_{v\in V_G}\Big(\mathbf{h}_{readout}^{(i)}\Big)\Big], \qquad (3) \\
\mathbf{h}_G &= \Big[\mathrm{avg}(\mathbf{X}), \mathrm{sum}(\mathbf{X}), \mathrm{max}(\mathbf{X}), \mathbf{h'}_{readout}^{(1)}, \dots, \mathbf{h'}_{readout}^{(d)}\Big], \\
\mathbf{o}_{readout} &= LogSoftMax(MLP(\mathbf{h}_G)).
\end{aligned}$$

4.2 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 (as in Eq. (3)). Then, it stacks these vectors for all the d layers and applies a linear transformation and the log softmax activation function in the end. In this way, the pre-training block learns stable node representations $\mathbf{h}_v^{GC(i)}$ for each *GraphConv* layer that are later fed to the GTM module. 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, 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. To get this output, we already mentioned that $\mathbf{h}_G$ passes over an MLP and consequently the LogSoftMax function is applied. Notice that so far each training phase has been carried out independently for each module and it is necessary to train them in this specific order, hence hampering the capability of the GTM-GNN to learn end-to-end.

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. Additional cycles of adaptation could take place by retraining the GTMs while fixing the rest of the network and so on, however, we did not investigate further this scenario. The pseudo-code that summarizes the training procedure is reported in Algorithm 1 in Appendix A.3.

5 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. An overview of the adopted datasets and baseline models is reported in the Appendix A.4 and A.5.

5.1 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 [18] 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 [15].

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 with Eq. (7) and its output 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.

5.2 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 3 in Appendix A.6 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 take the validation losses, i.e. the complete-data log-likelihood, to monitor the convergence and early stopping.

Dataset	**PTC**	**NCI1**	**PROTEINS**	**D&D**	**ENZYMES**
PSCN [19]	60.00±4.82	76.34±1.68	75.00±2.51	76.27±2.64	-
FGCNN [17]	58.82±1.80	81.50±0.39	74.57±0.80	77.47±0.86	-
DGCNN [17]	57.14±2.19	72.97±0.87	73.96±0.41	78.09±0.72	-
DGCNN [8]	-	76.4±1.7	72.9±3.5	76.6±4.3	38.9±5.7
GIN [8]	-	80.0±1.4	73.3±4.0	75.3±2.9	**59.6±4.5**
DIFFPOOL [8]	-	76.9±1.9	73.7±3.5	75.0±3.5	59.5±5.6
GraphSAGE [8]	-	76.0±1.8	73.0±4.5	72.9±2.0	58.2±6.0
DGCNN-DeepSets [17]	58.16±1.05	74.19±0.59	75.11±0.28	77.86±0.27	-
SOM-GCNN [21]	62.24±1.7	**83.30±0.45**	**75.22±0.61**	78.10±0.60	50.01±2.92
GTM-GCNN	**62.49±9.60**	82.48±1.33	72.88±4.82	**78.27±3.63**	59.03±5.92
GTM-GCNN Hyperparameters	(15×20) $q = 1$ $\lambda = 0.01$ $l = 30$	(15×20) $q = 5$ $\lambda = 0.1$ $l = 50$	(11×16) $q = 1$ $\lambda = 0.01$ $l = 20$	(12×17) $q = 3$ $\lambda = 0.1$ $l = 50$	(15×20) $q = 3$ $\lambda = 0.1$ $l = 50$

Table 1. Accuracies of GTM-GCNN and *state-of-the-art* models on the five used datasets. Values for the selected latent variable grid size, depth of the readout MLP q, regularization parameter λ and amount of hidden neurons l are reported.

5.3 Discussion

In Table 1, 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 [8]. The results reported in Table 1 were obtained by performing 5 runs of 10-fold cross-validation. The results reported in [27, 4, 28] 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 1 show that the GTM-GCNN achieved highly competitive performance in all considered datasets. In particular, on PTC and D&D the proposed method outperforms state-of-the-art results, while in NCI1, PROTEINS, and ENZYMES the accuracy results are higher than the one achieved by most of the models considered in the comparison. On NCI1 the GTM-GCNN shows the second-best performance, and only the SOM-GCNN outperforms our proposed model. While 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 2) 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 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 (see the next section).

The similarity between GTM-GCNN and SOM-GCNN makes the comparison between these two models very 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 and PROTEINS 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. In order to investigate the reason for this performance improvement, in Figure 2, we plot the heatmaps that represent the lattice representations of the SOM and GTM on ENZYMES datasets. The heatmaps were computed following the same procedure proposed in [21]. For the SOM heatmaps we selected the same best parameters reported in the original paper [21]. 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, this means 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—the node representations of the graphs benefit from the greater expressiveness and local discriminative power of the GTM in comparison with the SOM. Moreover, the local differences among representations of nodes belonging to different classes are more noticeable considering the ones obtained using the GTM (see for instance the dissimilarity between Label 2 and 3 w.r.t the SOM). The better representations obtained by the GTM are also due to less sensibility of the GTM to the values of the hyperparameters, in comparison with SOM. Indeed, as reported in Table 1, 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.

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 [21], 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

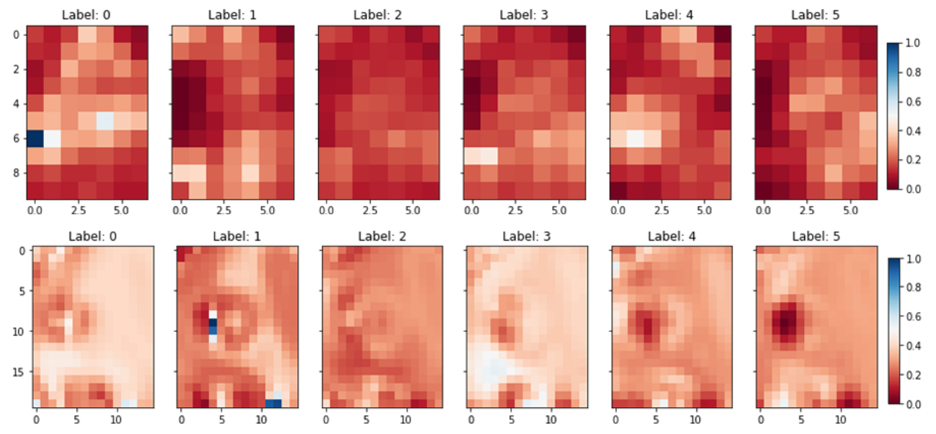

Fig. 2. Heatmaps of first-level SOM (top row) and GTM (bottom row) projected representations for the multi-class dataset ENZYMES. Heatmaps at higher levels are similar. Each heatmap is obtained by averaging the contribution of several graphs belonging to the corresponding class.

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 five 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.

In the future, we plan to develop an end-to-end supervised training algorithm to simplify and speed up the training of the model. Moreover, we will 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. Bishop, C.M., Svensén, M., Williams, C.K.: Gtm: The generative topographic mapping. Neural computation **10**(1), 215–234 (1998)
2. Bishop, C.M., Svensén, M., Williams, C.K.: Developments of the generative topographic mapping. Neurocomputing **21**(1), 203–224 (1998)
3. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**(suppl_1), i47–i56 (2005)
4. Chen, T., Bian, S., Sun, Y.: Are powerful graph neural nets necessary? a dissection on graph classification. arXiv preprint arXiv:1905.04579 (2019)
5. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. Advances in neural information processing systems **29** (2016)
6. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B (Methodological) **39**(1), 1–38 (1977)
7. Dobson, P.D., Doig, A.J.: Distinguishing enzyme structures from non-enzymes without alignments. Journal of molecular biology **330**(4), 771–783 (2003)
8. Errica, F., Podda, M., Bacciu, D., Micheli, A.: A fair comparison of graph neural networks for graph classification. In: International Conference on Learning Representations (2020), https://openreview.net/forum?id=HygDF6NFPB
9. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
10. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 1025–1035 (2017)
11. Helma, C., King, R.D., Kramer, S., Srinivasan, A.: The predictive toxicology challenge 2000–2001. Bioinformatics **17**(1), 107–108 (2001)
12. Kipf, T.N., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. In: ICLR. pp. 1–14 (2017). https://doi.org/10.1051/0004-6361/201527329, http://arxiv.org/abs/1609.02907
13. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biological cybernetics **43**(1), 59–69 (1982)
14. Liu, C., Zhan, Y., Li, C., Du, B., Wu, J., Hu, W., Liu, T., Tao, D.: Graph pooling for graph neural networks: Progress, challenges, and opportunities. arXiv preprint arXiv:2204.07321 (2022)
15. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization (2019)
16. Micheli, A.: Neural network for graphs: A contextual constructive approach. IEEE Transactions on Neural Networks **20**(3), 498–511 (2009)
17. Navarin, N., Tran, D.V., Sperduti, A.: Universal Readout for Graph Convolutional Neural Networks. In: International Joint Conference on Neural Networks. Budapest, Hungary (2019)
18. Navarin, N., Van Tran, D., Sperduti, A.: Learning kernel-based embeddings in graph neural networks. In: ECAI 2020, pp. 1387–1394. IOS Press (2020)
19. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. In: ICML. pp. 2014–2023 (2016)
20. Park, J., Sandberg, I.W.: Universal approximation using radial-basis-function networks. Neural Computation **3**(2), 246–257 (1991)
21. Pasa, L., Navarin, N., Sperduti, A.: Som-based aggregation for graph convolutional neural networks. Neural Computing and Applications pp. 1–20 (2020)

22. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019)
23. Shervashidze, N., Schweitzer, P., Van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**(77), 2539–2561 (2011)
24. Sperduti, A., Starita, A.: Supervised neural networks for the classification of structures. IEEE Trans. Neural Networks **8**(3), 714–735 (1997). https://doi.org/10.1109/72.572108, https://doi.org/10.1109/72.572108
25. Vinyals, O., Bengio, S., Kudlur, M.: Order matters: Sequence to sequence for sets. arXiv preprint arXiv:1511.06391 (2015)
26. Wale, N., Watson, I.A., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. Knowledge and Information Systems **14**(3), 347–375 (2008)
27. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? (2019)
28. Ying, R., You, J., Morris, C., Ren, X., Hamilton, W.L., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. arXiv preprint arXiv:1806.08804 (2018)
29. Ying, R., You, J., Morris, C., Ren, X., Hamilton, W.L., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling (2019)
30. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., Smola, A.: Deep sets. arXiv preprint arXiv:1703.06114 (2017)
31. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)

A Appendix

A.1 The Generative Topographic Mapping

The 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 [20]), 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

$$\mathbf{y}(\mathbf{u}, \mathbf{W}) = \mathbf{W}\boldsymbol{\phi}(\mathbf{u}), \tag{4}$$

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

$$\mathbf{m}_i = \mathbf{W}\boldsymbol{\phi}(\mathbf{u}), \tag{5}$$

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 $p(\mathbf{u})$

$$p(\mathbf{x}|\mathbf{W}, \beta) = \int p(\mathbf{x}|\mathbf{u}, \mathbf{W}, \beta) p(\mathbf{u}) d\mathbf{u}. \tag{6}$$

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:

$$R_i(\mathbf{x}; \mathbf{W}, \beta) = p(\mathbf{u}_i|\mathbf{x}, \mathbf{W}, \beta) = \frac{\exp\left\{-\frac{\beta}{2}\|\mathbf{m}_i - \mathbf{x}\|^2\right\}}{\sum_{j=1}^{K} \exp\left\{-\frac{\beta}{2}\|\mathbf{m}_j - \mathbf{x}\|^2\right\}}, \tag{7}$$

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 $\lambda\in\mathbb{R}$

$$p(\mathbf{W}|\lambda)=\left(\frac{\lambda}{2\pi}\right)^{MD/2}\exp\left\{-\frac{\lambda}{2}\sum_{j=1}^{M}\sum_{d=1}^{D}w_{jd}^{2}\right\}.$$

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 [6]. 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. 6, 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.

A.2 GTM-GNN Architecture Scheme

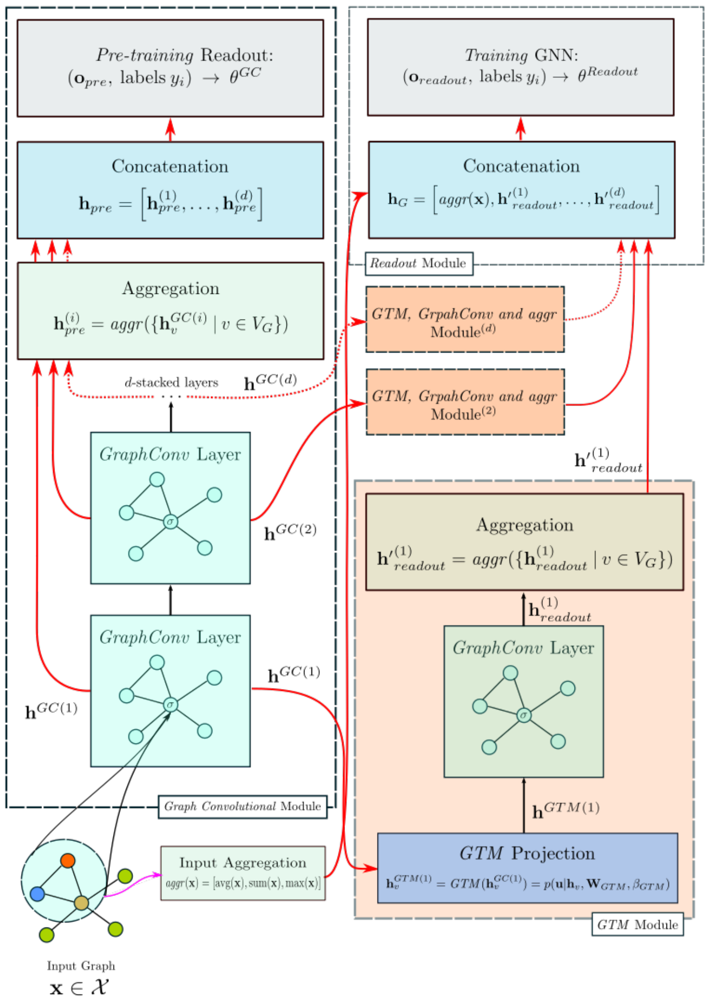
Pre-training Readout: $(\mathbf{o}_{pre}$, labels $y_i) \rightarrow \theta^{GC}$
Concatenation
$\mathbf{h}_{pre} = \left[\mathbf{h}^{(1)}_{pre}, \dots, \mathbf{h}^{(d)}_{pre}\right]$
Aggregation
$\mathbf{h}^{(i)}_{pre} = aggr(\{\mathbf{h}^{GC(i)}_v \mid v \in V_G\})$
d-stacked layers
$\mathbf{h}^{GC(d)}$
GraphConv Layer
$\mathbf{h}^{GC(2)}$
GraphConv Layer
$\mathbf{h}^{GC(1)}$
$\mathbf{h}^{GC(1)}$
Graph Convolutional Module
Training GNN: $(\mathbf{o}_{readout}$, labels $y_i) \rightarrow \theta^{Readout}$
Concatenation
$\mathbf{h}_G = \left[aggr(\mathbf{x}), \mathbf{h}'^{(1)}_{readout}, \dots, \mathbf{h}'^{(d)}_{readout}\right]$
Readout Module
GTM, GrpahConv and aggr Module$^{(d)}$
GTM, GrpahConv and aggr Module$^{(2)}$
$\mathbf{h}'^{(1)}_{readout}$
Aggregation
$\mathbf{h}'^{(1)}_{readout} = aggr(\{\mathbf{h}^{(1)}_{readout} \mid v \in V_G\})$
$\mathbf{h}^{(1)}_{readout}$
GraphConv Layer
$\mathbf{h}^{GTM(1)}$
GTM Projection
$\mathbf{h}^{GTM(1)}_v = GTM(\mathbf{h}^{GC(1)}_v) = p(\mathbf{u}|\mathbf{h}_v, \mathbf{W}_{GTM}, \beta_{GTM})$
GTM Module
Input Aggregation
$aggr(\mathbf{x}) = [\text{avg}(\mathbf{x}), \text{sum}(\mathbf{x}), \text{max}(\mathbf{x})]$
Input Graph
$\mathbf{x} \in \mathcal{X}$

Fig. 3. Graphical representation of the GTM-based GNN architecture. On the left side, the input graph goes first ot the *Graph Convolutional* module, that is trained independently on the *Pre-training* Readout layer. Then its representations are used to train the GTMs (bottom right) and their projection are learned by a *Readout* module. Finally, all the parameters but the GTMs' are adjusted in a fine-tuning step.

A.3 Training Procedure Pseudo-code

Algorithm 1 GTM-based GNN Training Procedure

```
Input: Graphs Dataset $\mathcal{X}$ with associated labels $y$
 1: $\boldsymbol{\theta}^{GC}, \mathbf{o}_{pre} \leftarrow \mathit{Pretraining}(\mathcal{X}, y)$    ▷ Pre-training of GraphConv Module
 2: $\forall$ stacked layer $i$ : $\mathbf{h}^{GC(i)} \leftarrow f\Big(\mathit{GraphConv}\Big(\mathbf{h}_v^{GC(i-1)}\Big)\Big)$    ▷ Get stable representations from pre-training module
 3: $\boldsymbol{\theta}^{GTM(i)} \leftarrow \mathit{GTM\_training}\Big(\mathit{tanh}\Big(\mathbf{h}^{GC(i)}\Big)\Big)$    ▷ Expectation Maximization
 4: $\mathbf{h}^{GTM(i)} \leftarrow \mathit{GTM}\Big(\mathit{tanh}\Big(\mathbf{h}^{GC(i)}\Big), \boldsymbol{\theta}^{GTM}\Big)$    ▷ Projected representations
 5: $\forall v : \hat{\mathbf{h}}_v^{GTM(i)} \leftarrow \frac{\mathbf{h}_v^{GTM(i)}}{\xi_v^{GTM(i)}}$, where $\xi_v^{GTM(i)}$ is the maximum of $\mathbf{h}_v^{GTM(i)}$ components
 6: $\forall i : \mathbf{h}_{readout}^{(i)} \leftarrow f\Big(\mathit{GraphConv}\Big(\hat{\mathbf{h}}_v^{GTM(i)}\Big)\Big)$    ▷ Readout Module
 7: $\forall i : \mathbf{h'}_{readout}^{(i)} \leftarrow \mathit{aggr}\Big(\Big\{\mathbf{h}_{readout}^{(i)} \mid v \in V_G\Big\}\Big)$    ▷ Aggregation
 8: $\mathbf{h}_G \leftarrow \Big[\mathrm{avg}(\mathbf{X}), \mathrm{sum}(\mathbf{X}), \mathrm{max}(\mathbf{X}), \mathbf{h'}_{readout}^{(1)}, \ldots, \mathbf{h'}_{readout}^{(d)}\Big]$    ▷ Concatenation
 9: $\mathbf{o}_{readout} \leftarrow f(\mathit{MLP}(\mathbf{h}_G))$
10: $\boldsymbol{\theta}^{readout} \leftarrow \mathit{LogSoftMax}(\mathcal{X}, y; \mathbf{o}_{readout})$    ▷ Readout training
11: $\boldsymbol{\theta}^{GC}, \boldsymbol{\theta}^{readout} \leftarrow \mathit{FineTuning}\big(\mathcal{X}, y; \boldsymbol{\theta}^{GC}, \boldsymbol{\theta}^{readout}\big)$    ▷ Fine-Tuning training
Output: GTM-based GNN$\big(\boldsymbol{\theta}^{GC}, \boldsymbol{\theta}^{GTM}, \boldsymbol{\theta}^{readout}\big)$
```

A.4 Datasets

We empirically validated the proposed GTM-GNN on five widely adopted datasets of graph classification: PTC [11], NCI1 [26], PROTEINS, [3], D&D [7] and ENZYMES [3]. All these datasets are modeling chemical/bioinformatics problems. PTC and NCI1 contains molecular graphs that represent chemical compounds. Each node is labeled with an atom type, and the edges represent bonds between them. The task in PTC consists in the prediction of the carcinogenicity of the compounds for male rats. In NCI1 the graphs represent anti-cancer screens for cell lung cancer. PROTEINS, D&D and ENZYMES, contain graphs that represent proteins. Each node corresponds to an amino acid and an edge connects two of them if they are less then 6Å (Angstrom) apart. In particular ENZYMES, differently than the other considered datasets (that model binary classification problems) allows testing the model on multi-class classification over 6 classes. Relevant statistics about the datasets are reported in Table 2.

A.5 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

Dataset	#Graphs	#Node	#Edge	Avg #Nodes/Graph	Avg.#Edges/Graph	Avg. Degree
PTC	344	4915	10108	14.29	14.69	2.06
NCI1	4110	122747	265506	29.87	32.30	2.16
PROTEINS	1113	43471	162088	39.06	72.82	3.73
D&D	1178	334925	1686092	284.32	715.66	5.03
ENZYMES	600	19580	74564	32.63	124.27	3.81

Table 2. Datasets statistics.

consider in our experimental comparison is the PSCN proposed by Niepert *et al.* [19]. 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 [10] 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 [10] has been extended by GIN [27], 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 [31] extends the GCN proposed by Kipf et al. [12] 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 [17].
The Funnel GCNN (FGCNN) model [18] does rely on the similarity of the adopted graph convolutional operator to the way the features of the Weisfeiler-Lehman (WL) Subtree Kernel [23] 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.

A.6 Hyperparameters and Configuration

Hyperparameter	List of values
Latent variables 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 Regularization λ	$10, 1, 0, 0.1, 0.01, \beta^{-1}$
MLP depth q	$1, 3, 5$
Hidden neurons l	$20, 30, 50$

Table 3. 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.

The GTM-based GNN has been implemented with `Python 3.8.8` and `PyTorch 1.8.1` [22], 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.