



# SOM-based aggregation for graph convolutional neural networks

Luca Pasa<sup>1</sup> · Nicolò Navarin<sup>1</sup> · Alessandro Sperduti<sup>1</sup>

Received: 13 May 2020 / Accepted: 27 October 2020  
© The Author(s) 2020

## Abstract

Graph property prediction is becoming more and more popular due to the increasing availability of scientific and social data naturally represented in a graph form. Because of that, many researchers are focusing on the development of improved graph neural network models. One of the main components of a graph neural network is the aggregation operator, needed to generate a graph-level representation from a set of node-level embeddings. The aggregation operator is critical since it should, in principle, provide a representation of the graph that is isomorphism invariant, i.e. the graph representation should be a function of graph nodes treated as a set. DeepSets (in: *Advances in neural information processing systems*, pp 3391–3401, 2017) provides a framework to construct a set-aggregation operator with universal approximation properties. In this paper, we propose a DeepSets aggregation operator, based on Self-Organizing Maps (SOM), to transform a set of node-level representations into a single graph-level one. The adoption of SOMs allows to compute node representations that embed the information about their mutual similarity. Experimental results on several real-world datasets show that our proposed approach achieves improved predictive performance compared to the commonly adopted sum aggregation and many state-of-the-art graph neural network architectures in the literature.

**Keywords** Graph neural networks · Self-organizing maps · Node aggregation

## 1 Introduction

Neural Networks for Graphs (GNNs), while dating back to more than 20 years ago [27], have recently gained popularity due to the good results in tasks such as semi-supervised node classification [14], link prediction [13], graph classification [22] and graph generation [18]. The main component making possible the application of neural networks to graph data is the Graph Convolution (GC), for which several definitions have been proposed in the literature. The majority of GC proposals share the basic principle of generating a (fixed-size) node representation considering its local neighborhood.

When considering graph-level prediction tasks, however, these topologically enriched representations at node-level need to be aggregated in order to obtain a single (fixed-size) representation of the graph. This aggregation component is crucial since it has to transform a variable number of node-level representations into a single graph-level one. Moreover, an effective and efficient graph-level representation should be, as much as possible, invariant to different isomorphic representations of the input graph, thus letting the learning procedure to only focus on the property prediction task, with no need to worry about the way the graph in input is represented. An approach that is commonly adopted in many graph neural network architectures proposed in the literature is to consider simple aggregation schemes such as the mean, the element-wise maximum, or the sum. However, recent results [21, 32] show that using such simple aggregations inevitably results in a loss of information due to the mix of numerical values they introduce, which may hurt the overall predictive performance of the GNN.

A much better approach, from a conceptual point of view, would be to consider all the topologically enriched representations of nodes of a graph as a set, and the

---

✉ Luca Pasa  
lpasa@math.unipd.it  
Nicolò Navarin  
nnavarin@math.unipd.it  
Alessandro Sperduti  
sperduti@math.unipd.it

<sup>1</sup> Department of Mathematics, University of Padua, Padova, Italy

aggregation function to learn as a function defined on these sets. DeepSets [32] constitute a recently proposed approach to design neural networks that take sets as input. Compared to other aggregation schemes, DeepSets scheme is maximally expressive since, under certain assumptions, it can be proved to be a universal approximator for functions over sets (see Sect. 2.3). A DeepSet projects the elements of the input set in a high-dimensional space via a learned  $\phi(\cdot)$  function, usually implemented as a multi-layer perceptron [21]. It then aggregates vectorial element representations summing them up to obtain a single vector representing the set, and finally it applies the readout, i.e., the  $\rho(\cdot)$  function (another MLP) to map the set-level representation to the output of the task at hand. Navarin et al. [21] propose a graph aggregation scheme based on DeepSets implementing the  $\phi(\cdot)$  function as a multi-layer perceptron. Motivated by the theoretical properties that the  $\phi(\cdot)$  function should possess, in this paper we propose to implement  $\phi(\cdot)$  exploiting self-organizing maps (SOMs) to map the node representations in the space defined by the activations of the SOM neurons. The resulting representations consider information about the similarity between the various inputs in an unsupervised way. In fact, similar input structures are mapped in similar output representations. Using a fully unsupervised mapping for the  $\phi(\cdot)$  function may however lead to lose task-related information. We thus propose to make the  $\phi(\cdot)$  mapping supervised by stacking, after the SOM, a layer that can be trained via supervised learning. Since we are dealing with graphs we propose, instead of simply using an MLP, to stack a Graph Convolution layer after the SOM, allowing to better incorporate topological information in the mapping. We can then apply the aggregation as prescribed by DeepSets. Finally, we implement the readout ( $\rho(\cdot)$  function) as an MLP. We show on several commonly adopted benchmark datasets for graphs the effectiveness of our proposal.

The paper is organized as follows. In Sect. 2 we introduce the necessary background concepts. In Sect. 3 we propose our SOM-based aggregation. In Sect. 4 we present our experimental results, and in Sect. 5 we analyze some properties of the proposed DeepSets-based aggregation. Section 6 concludes the paper.

## 2 Background

In the following, we use italic letters to refer to variables, bold lowercase letters to refer to vectors, and bold uppercase letters to refer to matrices. The elements of a matrix  $\mathbf{A}$  are referred as  $a_{ij}$  (and similarly for vectors). We use uppercase letters to refer to sets or tuples.

Let  $G = (V, E, \mathbf{X})$  be a graph, where  $V = \{v_0, \dots, v_{n-1}\}$  denotes the set of vertices (or nodes) of the graph,  $E \subseteq V \times V$  is the set of edges, and  $\mathbf{X} \in \mathbb{R}^{n \times s}$  is a multivariate signal on the graph nodes with the  $i$ -th row representing the attributes of  $v_i$ . We define  $\mathbf{A} \in \mathbb{R}^{n \times n}$  as the adjacency matrix of the graph, with elements  $a_{ij} = 1 \iff (v_i, v_j) \in E$ . With  $\mathcal{N}(v)$  we denote the set of nodes adjacent to node  $v$ . Let also  $\mathbf{D} \in \mathbb{R}^{n \times n}$  be the diagonal degree matrix where  $d_{ii} = \sum_j a_{ij}$ , and  $\mathbf{L}$  the normalized graph laplacian defined by  $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ , where  $\mathbf{I}$  is the identity matrix.

### 2.1 Neural Networks for Graphs

The first definition of the neural network for structured data, including graphs, has been proposed by Sperduti and Starita in 1997 [27]. Later, it has been refined by Micheli [19] and Scarselli et al. [24]. The core idea is to define a neural architecture that is modeled according to the graph topology. Thanks to weights sharing, the same set of neurons is applied to each vertex in the graph, and computes its output based on the representation of the vertex and of its neighbors. As usual, the function computed by each layer is parametric. Recently [5], this approach has been referred to as *graph convolution*. After a certain number of graph convolution layers, the node-level representations are merged by an *aggregation* operator, obtaining a fixed-size graph-level representation. Finally, the *readout* layer transforms this representation to the output of the task.

In more detail, a general Graph Neural Network model is built according to the following equations. First,  $d$  graph convolution layers are stacked:

$$\mathbf{h}_v^{\text{GC}(i)} = f\left(\text{graphconv}\left(\mathbf{h}_v^{\text{GC}(i-1)}, \left\{\mathbf{h}_u^{\text{GC}(i-1)} \mid u \in \mathcal{N}(v)\right\}\right)\right), \quad (1)$$

where  $f(\cdot)$  is an element-wise non-linear activation function,  $\text{graphconv}(\cdot, \cdot)$  a graph convolution operator, and  $\mathbf{h}_v^{\text{GC}(i)}$  is the representation of node  $v$  at the  $i$ -th graph convolution layer,  $1 \leq i \leq d$ , and  $\mathbf{h}_v^{\text{GC}(0)} = \mathbf{x}_v$  (i.e. the row of  $\mathbf{X}$  corresponding to  $v$ ). Then, an aggregation function is applied:

$$\mathbf{h}^S = \text{aggr}\left(\left\{\mathbf{h}_v^{\text{GC}(i)} \mid v \in V_G, 1 \leq i \leq d\right\}\right), \quad (2)$$

where  $\text{aggr}(\cdot)$  is the aggregator function. Note that the aggregation may depend on all the hidden representations computed by the different GC layers and not just the last one.  $\mathbf{h}^S$  is the fixed-size graph-level representation. Then, the *readout*( $\cdot$ ) (implemented as a multi-layer perceptron) applies some non-linear transformation on  $\mathbf{h}^S$ . Finally, we

apply the output layer (e.g., the LogSoftMax for a classification problem)

$$\mathbf{o} = \text{LogSoftMax}(\text{readout}(\mathbf{h}^S)). \quad (3)$$

For what concerns the graph convolution, in this paper we mainly consider a particular operator inspired by the Weisfeiler-Lehman graph invariant, which has been proposed by Morris et al. [20]. This GC, named *GraphConv*, is defined as follows:

$$\mathbf{H}^{(i+1)} = \mathbf{H}^{(i)}\bar{\mathbf{W}}^{(i)} + \mathbf{A}\mathbf{H}^{(i)}\hat{\mathbf{W}}^{(i)}, \quad (4)$$

where  $\mathbf{H}^0 = \mathbf{X}$ , and  $\bar{\mathbf{W}}^{(i)}$  and  $\hat{\mathbf{W}}^{(i)}$  are two weights matrices.

A more complete discussion about the GC operators is reported in ‘‘Appendix A’’. In the following, we present different definitions of node aggregation in the literature.

### 2.1.1 Aggregation of node representations

After stacking a number of graph convolution layers, an aggregation operator maps the set of representations associated with the single vertices into a graph-level representation. Different approaches to implement this aggregation operator are possible.

*Linear operators* The simplest aggregation operators adopted in the literature are linear, namely the average and the sum of vertex representations. NN4G [19] computes, for each graph, the average graph vertex representation for each hidden layer, and concatenates them. Other approaches consider only the last graph convolution layer to compute such an average [1]. In [9], multi-layer perceptrons are applied to transform node representations before a sum aggregator is applied.

*Non-linear operators* *SortPooling* is a non-linear pooling operator [33] used in conjunction with concatenation to obtain an aggregation operator. The idea is to select a pre-determined number of vertex embeddings using a sorting function, and to concatenate them, obtaining a graph-level representation of fixed size. Notice, however, that this representation ignores some of the nodes of the graph.

Another approach consists in using a set2set model, which is a simplified Neural Turing Machine, for handling sets as inputs [9] of the readout function. The model is capable of mapping sets to other sets in output, thus it is more powerful than what is required for classification or regression tasks. This complexity makes this instantiation hard to train, introducing unneeded complexity in the model. Finally, it has been shown that DeepSets, a general formulation of a universal approximator of functions over sets [32], can be successfully adopted as aggregator

operator [21] on graph nodes. More details are reported in Sect. 2.3.

## 2.2 Self-organizing map

The main goal of the Self-Organizing Map (SOM) algorithm is to transform the incoming signal pattern of an arbitrary dimension into a two- or three-dimensional discrete map, and to perform this transformation adaptively in a topological ordered fashion. The neurons of a self-organizing map are distributed over a lattice that is usually two- or three-dimensional, and equipped with synaptic weight vectors  $\mathbf{s}_i$ ,  $i \in [1..p]$ , where  $p$  is the number of the neurons of the SOM. These neurons compete among themselves to be activated, resulting in that only one neuron, the winner of the competition dubbed best matching unit (BMU), is selected as the prototype for each input pattern. The weights of the BMU are made closer to the input vector, as well as the weights of its neighbours, although at a minor degree, with the aim of preserving at the lattice level topological relationships in the input space. Different variants for the SOM model can be considered. The one used by us is described in detail in Sect. 3.1.

## 2.3 DeepSets

It has been proven [32] that any function  $sf(X)$  over a set  $X$ , satisfying the following two properties:

- 1 Variable number of elements in input, i.e., each input is a set  $X = \{x_1, \dots, x_m\}$  with  $x_i$  belonging to some set  $\mathfrak{X}$  (typically a vectorial space) and  $m > 0$ ;
- 2 Permutation invariance;

can be decomposed in the form:

$$sf(X) = \rho \left( \sum_{x_i \in X} \phi(x_i) \right), \quad (5)$$

for some  $\rho(\cdot)$  and  $\phi(\cdot)$  functions, if  $\mathfrak{X}$  is countable. This is the general formulation of DeepSets [32], which constitute a valid option to implement an aggregation operator. First of all, they can natively take in input the sets of topologically enriched (by graph convolutions) representations of graphs’ nodes. Moreover, being in principle universal approximators for a wide range of functions over countable sets, or uncountable sets with a fixed size, they are potentially very expressive from a functional point of view. Here we elaborate on this last capability by recalling some concepts from Zaheer et al. [32]. One of the main arguments of the universal approximation proof of DeepSets for the countable case, i.e., where the elements of the sets are countable ( $|\mathfrak{X}| \leq n$ ), relies on the fact that, given the space of input sets  $\mathcal{X} \subseteq 2^{\mathfrak{X}}$ , any function over sets can be

decomposed as  $sf(X) = \rho(e(X))$ , where  $e: \mathcal{X} \rightarrow \mathbb{R}^n$ ,  $e(X) = \sum_{x_i \in X} \phi(x_i)$ , combining the elements  $x_i \in X$  nonlinearly transformed by the  $\phi(\cdot)$  function, maps different sets in different points. Since we should be able to possibly associate (via  $\rho(\cdot)$ ) different outputs for different inputs, the  $\phi(\cdot)$  function should map its inputs (the elements of the sets) to an encoding of natural numbers, and  $e(\cdot)$  should provide a unique representation for every  $X \in \mathcal{X}$ . In the countable case, one way to achieve such property is to define the  $\phi(\cdot)$  function mapping each set element to a representation that is orthogonal to the representations of every other set element. For the uncountable case, the scenario becomes more complex, requiring  $\phi(\cdot)$  to be a homomorphism.

### 3 SOM-based DeepSets projection

In this paper, we propose to implement the  $\phi(\cdot)$  function of DeepSets by exploiting a Self-Organizing Map (SOM). A SOM can be defined to map each input embedding in the one-hot activation map of the SOM lattice, where only the winning neuron has a value different from zero. If we consider an infinitely wide SOM, it is easy to see that every input embedding activates a different winning neuron, and that different inputs are mapped in linearly independent SOM activation maps. Thus, using a SOM to implement the  $\phi(\cdot)$  function is, from a conceptual point of view, a viable approach. Following this path, however, poses some problems from the point of view of learnability. In fact, the encoding of sets in orthogonal vectors hinders the possibility to exploit similarities among examples. To solve this problem, we propose to smooth the SOM representations by exploiting relatively small lattices, and to return in output not only 1 for the winning neuron, but also a smaller adjusted value for its neighbors.

Finally, in order to compensate for the unsupervised nature of SOMs, we suggest to process the SOM output by a graph convolutional layer that, in addition to allow for supervised learning, can better preserve the topological information of the graph with respect to a simple MLP. Following this operation, the aggregation operator as prescribed by DeepSets can be applied. Even though DeepSets theoretically proves that the sum aggregation is maximally expressive, inspired by DiffPool [31] we consider the concatenation of different statistics such as the sum, average and component-wise maximum. We found that this choice leads to slightly improved overall predictive performance, probably slightly easing the training of the network. Finally, we implement the DeepSets readout function  $\rho(\cdot)$  as an MLP.

The proposed SOM-based projection is only one component of a graph neural network. In the following, we describe the overall network architecture, and we provide more details on the proposed aggregation operator.

We start considering graph convolution layers to provide a representation for each node in the graphs, using the following general equation:

$$\mathbf{h}_v^{GC(1)} = f(\text{graphconv}(\mathbf{x}_v, \{\mathbf{x}_u | u \in \mathcal{N}(v)\})), \quad (6)$$

that can be implemented with any graph convolution layer described in Appendix A. We stack  $d$  graph convolution layers as follows:

$$\mathbf{h}_v^{GC(i)} = f(\text{graphconv}(\mathbf{h}_v^{GC(i-1)}, \{\mathbf{h}_u^{GC(i-1)} | u \in \mathcal{N}(v)\})), \quad (7)$$

where  $1 \leq i \leq d$ . Let us consider the node representation generated by a graph convolution layer  $\mathbf{h}_v^{GC(i)}$ . We define our SOM-based aggregation operator as follows:

$$\mathbf{h}_v^{S(i)} = \text{som}(\mathbf{h}_v^{GC(i)}), \quad (8)$$

$$\mathbf{h}_v^{S(i)} = f(\text{graphconv}(\mathbf{h}_v^{S(i)}, \{\mathbf{h}_u^{S(i)} | u \in \mathcal{N}(v)\})), \quad (9)$$

$$\mathbf{h}^{S(i)} = \text{aggr}(\{\mathbf{h}_v^{S(i)} | v \in V_G\}), \quad (10)$$

where  $\text{som}(\cdot)$  is the function computing the SOM activations (detailed below).

In our architecture, we apply our proposed SOM-based projection operator to each graph convolution output, as shown in Fig. 1, obtaining  $d$  graph-level feature maps (one for each layer):  $\mathbf{h}^{S(1)}, \mathbf{h}^{S(2)}, \dots, \mathbf{h}^{S(d)}$ . These feature maps are then concatenated, obtaining a single graph-level representation:

$$\mathbf{h}^S = [\mathbf{h}^{S(1)}, \mathbf{h}^{S(2)}, \dots, \mathbf{h}^{S(d)}]. \quad (11)$$

We can apply the readout and the output layer (that together implement the  $\rho(\cdot)$  function in DeepSets) to the graph-level representation  $\mathbf{h}^S$ , obtaining the output of our network:

$$\mathbf{o} = \text{LogSoftMax}(\text{readout}(\mathbf{h}^S)). \quad (12)$$

The readout function is composed of several dense feed-forward layers, where we consider the number of layers and the number of neurons per layer as hyper-parameters. Each one of these layers uses the ReLU activation function, and is defined as follows:

$$\mathbf{h}^{R(j)} = \text{ReLU}(\mathbf{W}^{R(j)} \mathbf{h}^{R(j-1)} + \mathbf{b}^{R(j)}), \quad j \in [1, \dots, k-1] \quad (13)$$

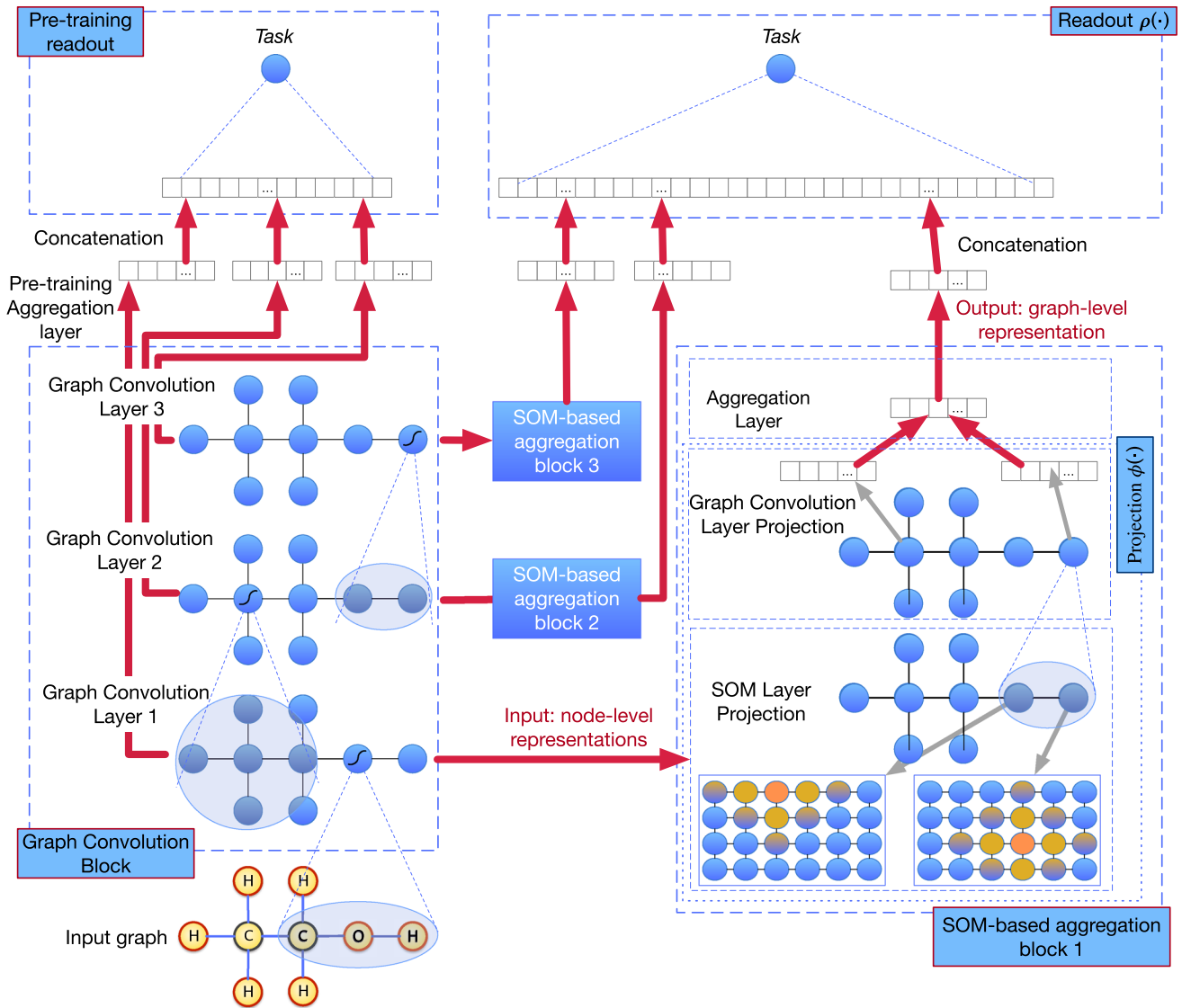


Fig. 1 A graphical layout of the proposed architecture, with an expanded view of the SOM-based aggregation block 1 (bottom right)

where  $\mathbf{h}^{R(0)} = \mathbf{h}^S$ . Finally, the output layer of the neural network for a  $c$ -class classification task is defined as follows:

$$\mathbf{o} = \text{LogSoftMax}(\mathbf{W}^o \mathbf{h}^{R(k-1)} + \mathbf{b}^o). \quad (14)$$

To reduce the *covariate shift* during training and to attenuate overfitting effects, we applied the batch normalization and dropout to the output of *each* graph convolutional layer.

### 3.1 SOM details

To define the SOM-based aggregation block, we adopted a SOM model that exploits a two-dimensional map,  $p' \times p''$ . We recall that we have an aggregation operator, and thus a SOM, associated with each graph convolution. The neurons

in the SOM at the  $k$ -th convolution layer  $s_{ij}^{(k)}$  are thus identified by two indices,  $i \in \{1, \dots, p'\}$  and  $j \in \{1, \dots, p''\}$ . As for the similarity measure, we use the 2-norm of the difference between the input and the SOM synaptic weights  $\mathbf{s}_{ij}^{(k)}$ . Thus, the distance between a SOM neuron and the input embedding for node  $v$  at layer  $k$  is defined as

$$d_{v,ij}^{(k)} = \left\| \mathbf{h}_v^{\text{GC}(k)} - \mathbf{s}_{ij}^{(k)} \right\|_2. \quad (15)$$

This measure is used to compute the BMU for each forward pass of the SOM, identified by the tuple of its position:

$$\left( i_v^{*(k)}, j_v^{*(k)} \right) = \underset{i,j}{\text{argmin}} \left( d_{v,ij}^{(k)} \right). \quad (16)$$



As the output of the SOM module, and in accordance with our discussion about the DeepSets approach, we propose to exploit the distances  $d_{v,i,j}^{(k)}$  to compute a similarity measure in the interval  $[0, 1]$  for each SOM neuron:

$$\zeta_{v,i,j}^{S(k)} = \alpha_{i,j}^{\sigma,i^*j^*} e^{-d_{v,i,j}^{(k)} + d_{v,i^*j^*}^{(k)}}, \tag{17}$$

where we omit the dependency from  $v$  and  $k$  from  $i^*$  and  $j^*$  for the ease of notation. In this way, each BMU will always output the value 1. The neighborhood function  $\alpha_{i,j}^{\sigma,i^*j^*}$  is defined following a Gaussian distribution over the topological distance between a neuron and the BMU as follows:

$$\alpha_{i,j}^{\sigma,i^*j^*} = e^{-((i^*-i)^2 + (j^*-j)^2) / \sigma^2}, \tag{18}$$

where  $\sigma$  is a hyper-parameter. Combining Eqs. (17) and (18), we finally get

$$\zeta_{v,i,j}^{S(k)} = e^{-d_{v,i,j}^{(k)} + d_{v,i^*j^*}^{(k)} - ((i^*-i)^2 + (j^*-j)^2) / \sigma^2}. \tag{19}$$

We flatten the two-dimensional output of the SOM obtaining a vectorial representation of the activation map of the SOM for each node, i.e.,  $\mathbf{h}_v^{S(k)}$  where each element of such vector is defined as  $\{\mathbf{h}_v^{S(k)}\}_{i:p'+j} = \zeta_{v,i,j}^{(k)}$ .

Finally, we implement the aggregator function  $\text{aggr}(\cdot)$  computing global statistics over the nodes in a graph as follows:

$$\mathbf{h}^{S(i)} = \text{aggr}\left(\left\{\mathbf{h}_v^{S(i)} \mid v \in V_G\right\}\right) \tag{20}$$

$$= \left[\text{avg}_{v \in V_G}\left(\mathbf{h}_v^{S(i)}\right), \max_{v \in V_G}\left(\mathbf{h}_v^{S(i)}\right), \text{sum}_{v \in V_G}\left(\mathbf{h}_v^{S(i)}\right)\right]. \tag{21}$$

### 3.2 Training procedure

To train a GNN that exploits the proposed SOM-based aggregation we have to deal with the fact that the SOM model requires an unsupervised training algorithm, while we consider supervised learning problems. For this reason, we developed a four-step training procedure. The basic idea is to, first of all, learn initial embeddings for nodes without using the SOM (pre-training). The aim of this step

is to learn stable representations to use for training the SOM. In the second step, the SOM is trained using an unsupervised method. The representations developed by the SOM are then fed for supervised training of the rest of the network (post-SOM GC and readout). Training is concluded with a fine-tuning training involving all network parameters, except for the SOM weights.

Let us define the set of parameters of the Graph Convolution Block in equation (7) as  $\theta^{\text{GCB}}$ , the SOM parameters as  $\theta^{\text{SOM}}$ , and all the other parameters (post-SOM GC and readout) as  $\theta^{\text{rest}}$ .

The first step consists in training the  $\theta^{\text{GCB}}$  parameters of the Graph Convolution Block. We add, only for this first step, an ad-hoc readout layer, which we refer to as pre-training readout, to perform supervised learning with backpropagation (see Fig. 1, top left). That allows to train this part of the model separately from the rest of the network. The pre-training readout layer is defined as follows:

$$\begin{aligned} \mathbf{h}_{\text{pre}}^{S(i)} &= \text{aggr}\left(\left\{\mathbf{h}_v^{\text{GC}(i)} \mid v \in V_G\right\}\right) \\ &= \left[\text{avg}_{v \in V_G}\left(\mathbf{h}_v^{\text{GC}(i)}\right), \max_{v \in V_G}\left(\mathbf{h}_v^{\text{GC}(i)}\right), \text{sum}_{v \in V_G}\left(\mathbf{h}_v^{\text{GC}(i)}\right)\right], \end{aligned} \tag{22}$$

$$\mathbf{h}_{\text{pre}}^S = \left[\mathbf{h}_{\text{pre}}^{S(1)}, \mathbf{h}_{\text{pre}}^{S(2)}, \dots, \mathbf{h}_{\text{pre}}^{S(d)}\right], \tag{23}$$

$$\mathbf{o}_{\text{pre}} = \text{LogSoftMax}\left(\mathbf{W}_{\text{opre}} \mathbf{h}_{\text{pre}}^S + \mathbf{b}_{\text{opre}}\right). \tag{24}$$

The second step consists in training the  $\theta^{\text{SOM}}$  weights of the SOM layers of the SOM-based aggregation blocks. We adopted the well-known unsupervised training method proposed by Kohonen [16]. In the third phase, the aim is to train  $\theta^{\text{rest}}$  parameters. This phase exploits again supervised training and the part of the model involved in these phases is represented by Eqs. (9)–(12). In this phase, the SOMs output will not change. The last training step consists in a fine-tuning phase. The aim is to tune the model parameters  $\theta^{\text{GCB}}$  and  $\theta^{\text{rest}}$ , maintaining  $\theta^{\text{SOM}}$  fixed. Further cycles of adaptation could take place by retraining the SOMs and so on. In this paper, however, we do not apply these further adaptation cycles. The pseudo-code that summarizes the training procedure is reported in Algorithm 1.

**Algorithm 1:** Training Procedure

---

```

/* Input: (X,y) */
/* Output: { $\theta^{GCB}$ ,  $\theta^{SOM}$ ,  $\theta^{rest}$ } */
/*  $V_{Tr}$ : set of all vertices in the Training Set (X,y) */
/* Functions: */
/* PreTraining: */
/* algorithm that minimizes the Negative Log-Likelihood  $NLL(y, \mathbf{o}_{pre})$ . */
/* Returns:  $\theta^{GCB}$  and  $\mathbf{W}_{opre}$  */
/* SOMTraining: */
/* SOM training algorithm [16]. Returns:  $\theta^{SOM}$  */
/* ReadOutTraining: */
/* algorithm that minimizes  $NLL(y, \mathbf{o})$ . Returns:  $\theta^{rest}$  */
/* FineTuning: */
/* algorithm that minimizes  $NLL(y, \mathbf{o})$ . Returns:  $\theta^{GCB}$  and  $\theta^{rest}$  */
/* Parameter initialization */
/*  $\theta^{GCB}$ ,  $\theta^{rest}$ , and  $\mathbf{W}_{opre}$  are initialized using Uniform distribution */
/*  $\theta^{SOM}$  is initialized using normal distribution  $\mathcal{N}(0, 1)$  */
/* First step: pre-training Graph Convolution Block */
1  $\theta^{GCB}, \mathbf{W}_{opre} \leftarrow PreTraining(\mathbf{X}, \mathbf{y}; \theta^{GCB}, \mathbf{W}_{opre})$ 
/* Second step: Unsupervised training of SOM Layer Projection */
2  $\forall \mathbf{v} \in V_{Tr} : \mathbf{h}_v^{GC(i)} \leftarrow f(graphconv(\mathbf{h}_v^{GC(i-1)}, \{\mathbf{h}_u^{GC(i-1)} | u \in \mathcal{N}(v)\}; \theta^{GCB}))$ 
3  $\mathbf{h}^{GC(i)} \leftarrow [\mathbf{h}_v^{GC(i)}, \forall \mathbf{v} \in V_{Tr}]$ 
4  $\mathbf{h}^{GC} \leftarrow [\mathbf{h}^{GC(1)}, \mathbf{h}^{GC(2)}, \dots, \mathbf{h}^{GC(d)}]$ 
5  $\theta^{SOM} \leftarrow SOMTraining(\mathbf{h}^{GC}; \theta^{SOM})$ 
/* Third step: train Graph Convolution Layer Projection and Readout */
6  $\mathbf{h}^{IS} \leftarrow [SOM^{(1)}(\mathbf{h}^{GC(1)}; \theta^{SOM(1)}), \dots, SOM^{(d)}(\mathbf{h}^{GC(d)}; \theta^{SOM(d)})]$ 
7  $\theta^{rest} \leftarrow ReadOutTraining(\mathbf{h}^{IS}, \mathbf{y}; \theta^{rest})$ 
/* Forth step: fine-tuning */
8  $\theta^{GCB}, \theta^{rest} \leftarrow FineTuning(\mathbf{X}, \mathbf{y}; \theta^{GCB}, \theta^{rest})$ 
9 return  $\theta^{GCB}, \theta^{SOM}, \theta^{rest}$ 

```

---

## 4 Experimental results

In this section, we report and discuss the results obtained by our proposed model, which we refer to as SOM-GCNN. We start introducing the adopted datasets, our model set up, and the hyper-parameter selection strategy. We then present and discuss our experimental results.

### 4.1 Datasets

The proposed model, and the other models compared with it, were empirically validated on commonly adopted graph classification benchmarks. Specifically, we used five

datasets modeling bioinformatics problems: PTC [12], NCI1 [29], PROTEINS, [3], D&D [6] and ENZYMES [3]. PTC, and NCI1 involve chemical compounds represented by their molecular graph, where the atom type is represented by node labels, and bonds correspond to edges. The prediction task for PTC concerns the carcinogenicity for male rats by chemical compounds. In NCI1 the graphs represent anti-cancer screens for cell lung cancer. The remaining datasets, PROTEINS, D&D and ENZYMES, involve graphs that represent proteins. Amino acids are represented by nodes while edges connect amino acids that are less than 6 Å apart. All the prediction tasks are binary classification tasks, except for the ENZYMES dataset, where a multi-class classification of chemical compounds

**Table 1** Datasets statistics

Dataset	#Graphs	#Node	#Edge	Avg #Nodes/graph	Avg. #Edges/graph
PTC	344	4915	10,108	14.29	14.69
NCI1	4110	122,747	265,506	29.87	32.30
Proteins	1113	43,471	162,088	39.06	72.82
D&D	1178	334,925	1,686,092	284.32	715.66
Enzymes	600	19,580	74,564	32.63	124.27

(6 classes) is represented. Relevant statistics about the datasets are reported in Table 1.

## 4.2 Model selection and experimental setup

In order to test the effectiveness of the proposed SOM aggregation block, we applied it to a simple GNN model inspired by the FGCNN model [21], which obtained very good results in the considered datasets. Our model is reported in Fig. 1. In particular the base model (*Graph Convolution Block* in the figure) is composed of three Graph Convolution layers (therefore  $d$  is set to 3 in Eq. (11)). In FGCNN, the number of neurons of each layer increases with increasing depth. More precisely,  $\mathbf{h}^{\text{GC}(1)} \in \mathbb{R}^l$ ,  $\mathbf{h}^{\text{GC}(2)} \in \mathbb{R}^{2l}$ , and  $\mathbf{h}^{\text{GC}(3)} \in \mathbb{R}^{3l}$ . As graph convolution operator we used the *GraphConv* [20]. As activation function  $f$  for the various convolution layers we choose the ReLU.

For what concerns the readout part of the model, we experimented with a shallow setting, where  $k = 1$ , therefore only the output layer defined in Eq. (14) is used after the aggregation layer  $\mathbf{h}^S$  in Eq. (11). We also tested a model that exploits a deeper version of the readout composed of three fully connected feed-forward layers before the output layer ( $k = 3$ ). In any case, the pre-training readout exploited a single output layer, i.e.,  $k = 1$ . The results reported in Table 4 were obtained by performing five runs of tenfold cross-validation.

The goal of this work is to evaluate the benefit of using the SOM-based aggregation. For this reason, we focused our attention on the adopted SOM parameters, in particular, we carefully validated the dimensions of the lattice, and the learning rate used during the self-organizing phase. These two parameters were selected using a limited grid search where the explored set of values changes based on the considered dataset (see Table 2). Due to the high time requirements of performing an extensive grid search, we decided to limit the number of values taken into account for each hyper-parameter, by performing several preliminary tests. As score, we used the average accuracy computed over the 10-fold cross-validation on the validation sets, and we used the same set of selected hyper-parameters for each fold. The selection of the epoch was performed for each fold independently based on the accuracy on the

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

Dataset	SOM learning-rate	SOM sizes
PTC	0.1, 0.05, 0.005	(12, 9), (16, 12), (20, 15)
NCI1	0.1, 0.05, 0.005	(10,7), (15, 10), (15, 20)
Proteins	0.1, 0.05, 0.005	(10, 7), (12, 9), (15, 10)
D&D	0.1, 0.05, 0.005	(7, 10), (15, 10), (17, 12)
Enzymes	0.1, 0.05, 0.005	(10, 7), (12, 9), (15, 12)

validation set. Another parameter that influences the SOM performance is  $\sigma$  that modifies the impact of the neighborhood function (see Eq. (18)). For this value, we use a heuristic solution by making the value dependent on the maximum dimension of the SOM:

$$\sigma = \max(p', p'')/2. \quad (25)$$

During training the trend of neighborhood function is influenced not only by the value of  $\sigma$ , but also by the learning rate. Indeed at each training iteration the  $\sigma$  value used by the neighborhood function to update the weights is defined as follows:

$$\sigma_{\text{training}} = \sigma \cdot lr \cdot ((1 - i_c)/i_t), \quad (26)$$

where  $lr$  is the learning rate,  $i_c$  is the current iteration, and  $i_t$  is the total number of training iterations.

An issue related to the proposed SOM-based aggregation is the high number of hyper-parameters. This aspect makes time-consuming to perform a complete grid search considering all hyper-parameters. For this reason, for what concerns the other parameters of the model, we adopted a random search strategy. Therefore, we believe that all reported results may be slightly improved by a more systematic search. The selected parameters for each dataset/configuration are reported in Table 3. For further details about the experimental setup, it is possible to consult the publicly available code.<sup>1</sup>

<sup>1</sup> <https://github.com/lpasa/SOMBasedGraphAggregation>.



**Table 3** Hyper-parameters selected via random search

Dataset	Readout	$l$	LR pre-training	LR readout	LR fine-tuning	Batch size
PTC	Shallow	30	$5 \times 10^{-4}$	$5 \times 10^{-4}$	$1 \times 10^{-4}$	32
PTC	Deep	20	$5 \times 10^{-4}$	$5 \times 10^{-4}$	$1 \times 10^{-4}$	32
NCII	Shallow	75	$5 \times 10^{-4}$	$5 \times 10^{-4}$	$1 \times 10^{-4}$	32
NCII	Deep	50	$5 \times 10^{-4}$	$5 \times 10^{-4}$	$1 \times 10^{-4}$	32
Proteins	Shallow	30	$1 \times 10^{-4}$	$1 \times 10^{-4}$	$1 \times 10^{-4}$	32
Proteins	Deep	20	$1 \times 10^{-4}$	$1 \times 10^{-4}$	$1 \times 10^{-4}$	32
D&D	Shallow	40	$5 \times 10^{-4}$	$5 \times 10^{-4}$	$1 \times 10^{-4}$	16
D&D	Deep	40	$5 \times 10^{-4}$	$5 \times 10^{-4}$	$1 \times 10^{-4}$	16
Enzymes	Shallow	30	$5 \times 10^{-4}$	$5 \times 10^{-4}$	$1 \times 10^{-4}$	32
Enzymes	Deep	20	$5 \times 10^{-4}$	$5 \times 10^{-4}$	$1 \times 10^{-4}$	32

For what concerns dropout probability and weight decay all datasets/configurations use 0.5 and  $5 \times 10^{-4}$ , respectively

**Table 4** Accuracies by SOM-GCNN and *state-of-the-art* models on the five used datasets

Dataset	PTC	NCII	Proteins	D&D	Enzymes
PSCN [23]	60.00 ± 4.82	76.34 ± 1.68	75.00 ± 2.51	76.27 ± 2.64	–
FGCNN [21]	58.82 ± 1.80	81.50 ± 0.39	74.57 ± 0.80	77.47 ± 0.86	–
DGCNN [21]	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	<b>59.6 ± 4.5</b>
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 [21]	58.16 ± 1.05	74.19 ± 0.59	75.11 ± 0.28	77.86 ± 0.27	–
SOM-GCNN shallow	<b>62.24 ± 1.7</b>	<b>83.30 ± 0.45</b>	75.01 ± 0.65	74.65 ± 1.23	50.01 ± 2.92
SOM-GCNN deep	61.70 ± 0.97	82.32 ± 0.52	<b>75.22 ± 0.61</b>	<b>78.10 ± 0.60</b>	44.10 ± 1.87

The best results are reported in bold

### 4.3 Baselines

We compare SOM-GCNN with several GNN architectures, which achieved state-of-the-art results on the different datasets. In Table 4 we report the results of such comparison. Specifically, we consider PSCN [23], Funnel GCNN (FGCNN) [21], DGCNN [34], GIN [30], DIFFPOOL [31] and GraphSage [11]. More details about these architectures are provided in “Appendix B”. For FGCNN, we did not consider the additional loss term based on the Weisfeiler-Lehman graph kernel to ensure a fair comparison.

The results reported in the GIN paper [30] cannot be compared with the other results reported in Table 4, because the authors state “*The hyper-parameters we tune for each dataset are [...] the number of epochs, i.e., a single epoch with the best cross-validation accuracy averaged over the ten folds was selected.*”. Similarly, for the result reported in Chen et al. [4] for the GCN and the GFN models, the authors state “*We run the model for 100*

*epochs, and select the epoch in the same way as [30], i.e., a single epoch with the best cross-validation accuracy averaged over the ten folds is selected.*” In both cases, the model selection strategy is different from the one we adopted. This makes the results not comparable. Moreover, in the GIN and Diffpool papers [30, 31] the node descriptors are augmented with structural features (a one-hot representation of the node degree in both works, and a clustering coefficient in the latter). We decided to use a common setting for the chemical domain, where the nodes are labeled with a one-hot encoding of their atom type only. The only exception is ENZYMES, where it is common to use 18 additional available features, thus we decided to adopt this setting. These issues, and the importance of the validation strategy, are highlighted and discussed in [8]. The same paper reports the results of a fair comparison between the considered baseline models. We follow this setting as much as possible.

For the sake of comparison we also reported the results of the DGCNN-DeepSets [21]. We want to point out that these results are not completely comparable with the ones obtained by our models for two reasons: (i) the DeepSets was implemented by MLPs on a different base GNN architecture, i.e., a DGCNN model; (ii) the adopted validation strategy is different from the one we applied in this study.

#### 4.4 Discussion of experimental results

The results reported in Table 4 show that the predictive performance of our proposed SOM-GCNN architecture are highly competitive compared to the other considered methods in all the considered datasets, with the exception of ENZYMES. Specifically, the proposed method shows the best performances on PTC, NCI1, PROTEINS and D&D datasets, significantly outperforming state-of-the-art methods in many cases. NCI1 and D&D are the two datasets that have higher numbers of nodes/graphs. From this point of view, it is also interesting to notice that the experimental results on D&D are the only case where the deep version of the readout shows improved results compared to the shallow one. Indeed, in the other datasets, the higher number of parameters of the deep readout and the limited size of the datasets favor the onset of overfitting in this setting. That is more visible in the fine-tuning phase. In this regard, it is important to point out that the selection of the type of output highly depends on the considered dataset. For the seek of analysis, we report the results of both

types of output for all datasets. In some cases (like DD and PROTEINS) we consider the model with a deep readout, while for the other datasets is clearly more convenient to use a shallow readout. Note that the selection of the readout type is part of the validation process. Moreover, the proposed base aggregation method turns out to be completely detached from the adopted readout architecture. The second best performing method on D&D is DGCNN in the validation setting of Navarin et al. [21]. However, in the more rigorous validation setting of Errica et al. [8] that we followed, the difference gap compared to our proposed SOM-GCNN is higher. On PROTEINS, SOM-GCNN achieves slightly improved predictive performances compared to PSCN [23], but with a significantly lower variance. Moreover, its performance are close to DGCNN-DeepSets [21], for which we recall there is a difference in the validation setting that can favor DGCNN-DeepSets. The ENZYMES dataset involves a 6-class classification task, and in this case, the proposed SOM-based aggregation method seems very efficient in increasing the graph convolutional accuracy. Indeed, the results obtained after the SOM-based aggregation improve significantly the ones achieved after the first training phase. Unfortunately, the accuracy of the base GNN model (Table 5) is significantly lower than the ones reached by the state-of-the-art models. We argue that this difference in accuracy performance is highly related to the readout part of the model. In fact, the model considered during the first training phase turns out to be similar to the FGCNN network reported in Table 4 except for the readout part, which in our case is shallow.

**Table 5** Accuracy obtained after the various training steps

Dataset	Pre-training	Readout type	Readout	Fine-tuning
PTC	57.75 ± 1.51	Shallow	62.24 ± 1.71	61.15 ± 1.54
	58.05 ± 1.62	Deep	61.70 ± 0.97	61.19 ± 0.65
4 layers GC Block	58.77 ± 1.76			
NCI1	81.53 ± 0.51	Shallow	82.32 ± 0.52	83.30 ± 0.45
	81.69 ± 0.30	Deep	82.32 ± 0.52	82.27 ± 0.18
4 layers GC Block	82.09 ± 0.51			
Proteins	72.07 ± 1.16	Shallow	73.88 ± 0.37	75.01 ± 0.65
	73.15 ± 0.88	Deep	74.92 ± 0.97	75.22 ± 0.61
4 layers GC Block	69.93 ± 1.40			
D&D	72.00 ± 0.82	Shallow	74.52 ± 1.19	74.65 ± 1.23
	70.24 ± 1.22	Deep	77.57 ± 0.41	78.10 ± 0.6
4 layers GC Block	70.34 ± 1.57			
Enzymes	43.7 ± 0.84	Shallow	46.77 ± 1.50	50.01 ± 2.92
	41.81 ± 1.81	Deep	43.93 ± 2.19	44.10 ± 1.87
4 layers GC Block	43.30 ± 3.41			

Note that the pre-training phase always uses a shallow pre-training readout, while the hyper-parameters could be different (see Table 3). The last row of each dataset reports the accuracy of the pre-training readout using a Graph Convolution Block composed of four GC layers

The choice of using a simple and shallow readout layer for pre-training the first part of the model is due to the fact that the readout part (Eqs. (22)–(24)) will be discarded after the pre-training phase. Moreover, the first training step aims to initialize properly the weights of the Graph Convolution Block to provide meaningful inputs to the second training step that involves the SOMs.

In Table 5, we reported the accuracy results reached in the different training phases (we merge the SOM training and the readout part training since both steps are necessary to obtain a fully trained model). In Table 4 we reported for each dataset/configuration the best result with or without fine-tuning, selected according to the validation error. Indeed the fine-tuning phase does not always improve the model classification performance. It turns out to be more useful when the size of the training set is large. Indeed, also, in this case, the problem is related to the already mentioned overfitting phenomenon. The results reported in Table 5 show that the application of the SOM-based aggregation blocks always allows to reach improved results compared to the base model (pre-training). In particular, the comparison between the pre-training and the readout accuracy in the shallow model (where the exploited readouts are very similar) highlights the benefit of adopting the proposed aggregation technique. Table 6 reports the results obtained on the validation set by using different SOM dimensions. In three out of five datasets, the SOMs with highest dimensions obtain better results. One exception is

**Table 6** Accuracy obtained on the validation sets by using SOM with different lattice sizes

Dataset	Som size		
PTC	(12, 9)	(16, 12)	(20, 15)
Shallow readout	<b>79.14</b>	76.00	74.29
Deep readout	73.430	<b>74.00</b>	71.43
NCII	(10, 7)	(15, 10)	(20, 15)
Shallow readout	84.38	85.11	<b>86.30</b>
Deep readout	83.65	<b>84.64</b>	84.54
Proteins	(10, 7)	(12, 9)	(15, 10)
Shallow readout	78.30	78.48	<b>79.91</b>
Deep readout	79.46	78.482	<b>80.09</b>
D&D	(10, 7)	(15, 10)	(17, 12)
Shallow readout	80.08	80.42	<b>81.10</b>
Deep readout	82.46	81.78	<b>83.05</b>
Enzymes	(10, 7)	(12, 9)	(15, 12)
Shallow readout	<b>61.00</b>	56.17	57.50
Deep readout	<b>54.5</b>	52.83	52.17

The best results are reported in bold

Each reported result was obtained by selecting the best epoch and using the best learning rate based on validation test accuracy results

the deep readout version of the SOM-GCNN on NCII, where the results of the (15,10) SOM are very close to the one with the lattice of (20,15). Moreover, in this case, also the overfitting influences the results. In our preliminary experiments, we noticed that using larger lattices drives the model to overfit quickly. The other exceptions are the PTC and ENZYMES datasets. These datasets are the ones with the smallest number of nodes/graphs (see Table 1), which reduces the advantage of using larger lattices. Having larger SOMs increases also the number of parameters of the model, and considering the limited size of these two datasets, using a larger lattice may boost the rise of the overfitting issues. In the next section, we analyze in more detail the SOM's behavior in the proposed aggregation mechanism.

In Table 5, we also report the results of a pre-trained graph convolutional network with four GC layers. These results turn out to be crucial to assess the benefit of using the SOM-based aggregation blocks. Indeed, considering that a SOM-based aggregation block includes a GC layer after the SOM, it is important to verify that the observed improvements with respect to the pre-trained model are not due to that additional GC layer. The comparison with a pre-trained model with four GC layers is thus intended to provide this information. The results show that, in all cases, the use of a further GC layer does not significantly improve the accuracy results, which turn out to be similar to the results obtained with the three layers graph convolutional block. Therefore we can conclude that the SOM plays a crucial role in the proposed aggregation method.

## 5 SOM-based aggregation block analysis

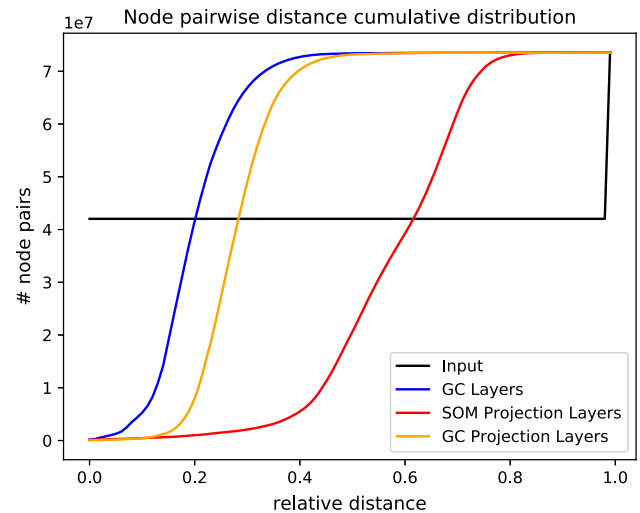
In this section, we analyze how the use of the SOM layer modifies the representations learned by a GNN. In particular, we start by analyzing the benefits of using a trained SOM vs. using an untrained one. Subsequently, we study the global distribution of the representations generated by the SOM. We do this by comparing the normalized distribution of the distances between all couples of nodes' embeddings in the whole dataset in the representation computed by the SOM, versus the ones obtained by the first set of GC layers and by the post-SOMs CG layers. As a result of this analysis, we question whether the GC projection layers are actually useful. For this reason we perform an ablation study that shows that these layers are beneficial for the final performance. Finally, we explore how nodes belonging to positive/negative graphs are mapped onto the SOM lattice. This last analysis is quite interesting since it allows, in principle, to *explain* the output of the model. This is an additional advantage of integrating SOM modules into the model.

## 5.1 Advantage in using SOM-based aggregation block

In order to evaluate the positive effect of using SOM projection, we defined a model by randomly initializing the SOM synaptic weights  $s_{i,j}^{(k)}$ , and trained it without performing the second step, where the SOMs weights are trained using the unsupervised method. Thus, the SOMs of the obtained models have all the weights set to random values. The results of these experiments are reported in Table 7, and they can be directly compared versus the ones reported in Table 5, since we used the same hyperparameters. Also in this case the results were obtained by performing five runs of 10-fold cross-validation. It is possible to notice that in all cases the obtained accuracy drops significantly using random SOM synaptic weights. Note that, in some cases the results after the readout training are even lower than the ones computed in the pre-training phase. It is important to point out that also in this case the readout training phase is performed. These results suggest that the SOMs allow to obtain a node embedding that makes it somehow easier to perform the classification task. In the following, we analyze how the SOM influences the obtained node embeddings.

## 5.2 Node embedding distances

In Fig. 2 we report the cumulative distribution of the (normalized) pairwise nodes distances of all nodes of the training set's graphs of the NCI1 dataset. We observed a similar behavior for the other datasets, thus we decided to omit the plots for brevity. The black line represents the cumulative distribution of the input nodes representations (node labels). In the figure, it is possible to notice that in NCI1 dataset there are almost  $10^4$  nodes that share the same input representation.



**Fig. 2** Cumulative distribution of the pairwise nodes distances of all nodes of the training set's graphs of the NCI1 dataset. Note that in the y-axis each value is written in scientific notation and is multiplied by  $10^7$

The blue line represents the cumulative distribution of the representations returned by the Graph Convolution block of the model. In this specific case, the considered node representation is the concatenation of the three graph convolutional layers outputs:

$$\mathbf{h}_v^{\text{GC}} = [\mathbf{h}_v^{\text{GC}(1)}, \mathbf{h}_v^{\text{GC}(2)}, \mathbf{h}_v^{\text{GC}(3)}]. \quad (27)$$

The SOM in the  $i$ -th SOM-based aggregation block receives in input the corresponding node embeddings  $\mathbf{h}_v^{\text{GC}(i)}$  and computes in output  $\mathbf{h}_v^{\text{S}(i)}$ . The red curve represents the cumulative distance between the concatenation of the various SOM Layers Aggregation outputs:

$$\mathbf{h}_v^{\text{S}} = [\mathbf{h}_v^{\text{S}(1)}, \mathbf{h}_v^{\text{S}(2)}, \mathbf{h}_v^{\text{S}(3)}]. \quad (28)$$

Finally, we analyze the representation obtained by the

**Table 7** Accuracy obtained using random initialized SOMs, and without performing the unsupervised learning phase on them

Dataset	Pre-training	Readout type	Readout	Fine-tuning
PTC	59.61 ± 2.48	Shallow	57.04 ± 1.25	57.14 ± 1.05
	59.49 ± 1.84	Deep	58.16 ± 1.73	59.14 ± 0.74
NCI1	81.29 ± 0.38	Shallow	81.17 ± 0.24	81.65 ± 0.15
	81.57 ± 0.46	Deep	81.18 ± 0.49	81.45 ± 0.36
Proteins	71.29 ± 1.09	Shallow	71.91 ± 0.96	72.39 ± 0.75
	71.51 ± 1.02	Deep	72.63 ± 0.65	73.43 ± 0.86
D&D	70.44 ± 1.56	Shallow	71.85 ± 1.08	72.58 ± 1.21
	71.39 ± 0.86	Deep	75.21 ± 0.44	75.76 ± 0.86
Enzymes	43.77 ± 1.64	Shallow	40.83 ± 2.16	45.03 ± 2.03
	41.17 ± 1.38	Deep	42.07 ± 1.22	42.10 ± 1.96

Note that the pre-training phase always uses a shallow pre-training readout, while the hyper-parameters could be different (see Table 3)

*Graph Convolution Layer Projection*. Also in this case we considered the distances of the concatenation of the various layer outputs:

$$\mathbf{h}_v^{iS} = \left[ \mathbf{h}_v^{iS(1)}, \mathbf{h}_v^{iS(2)}, \mathbf{h}_v^{iS(3)} \right]. \quad (29)$$

The yellow curve represents the correspondent cumulative distribution.

The first observation is that SOM tends to increase the pairwise distance between nodes. We think this is consistent with the SOM property to exploit several neurons in the lattice for dense populated zones in the input space. The second observation is that the application of the *Graph Convolution Layer Projection* after the SOM Layers Aggregation reduces the distances among the representations. However, the resulting representations are less close than the ones computed by the first Graph Convolution Layers. Note that the representations  $\mathbf{h}_v^{iS(i)}$  encapsulate more information than  $\mathbf{h}_v^{GC(i)}$  (and consequently, by  $\mathbf{h}_v^{S(i)}$ ), because the additional Graph Convolution layer extends of one hop the node neighbor.

As a final remark, we think it is fair to claim that the SOM seems to help the model to develop more separable internal representations, which may explain the better observed performances.

### 5.3 Graph convolution layer projection

As reported in Fig. 1, the proposed SOM-based aggregation block is composed of three parts: *SOM Layer Projection*, *Graph Convolution Layer Projection* and *Aggregation Layer*. In the previous section, we showed that *Graph Convolution Layer Projection* reduces the distances among the representations obtained by the *SOM Layer Projection*. In order to study whether the contraction made by Graph Convolution Layer is useful, we trained the model removing these specific components. The results of

this ablation study are reported in Table 8, and they can be directly compared versus the ones reported in Table 5, since we used the same hyperparameters.

The results show the benefits of having a Graph Convolutional layer that allows to aggregate the *SOM Layer Projection* outputs according to the graph topology. In all datasets, the accuracy decreases significantly compared to the results obtained with the full model (Table 5). Note that for datasets that have a higher average number of edges per graph (DD, ENZYMES), the accuracy drop is even more pronounced.

### 5.4 SOM lattice representations

In Figs. 3, 4, 5 and 6, we report heatmaps with entries computed as follows. Let  $G_+$  and  $G_-$  be set of graphs belonging to class + and -, respectively. For each layer level  $k$ , and class  $c \in \{+, -\}$ , we define the entry  $(i, j)$  of heatmap  $(c)^{(k)}$  as follows:

$$\text{heatmap}(c)_{ij}^{(k)} = \frac{1}{|G_c|} \sum_{g \in G_c} \sum_{v \in V_g} \zeta_{v,ij}^{(k)}. \quad (30)$$

Notice that the inner summation computes a graph-based contribution of each neuron, since it aggregates the outputs of the neuron for all the nodes of a single graph. Then, the outer summation computes an average over the set of graphs belonging to the same class  $c$ .

Each figure reports the heatmap of each SOM layer (one per SOM-based aggregation block). We also report the differences between the two heatmaps in order to highlight how the SOM output conveys information related to the classification task. Our analysis is limited to the datasets that model a binary classification task.

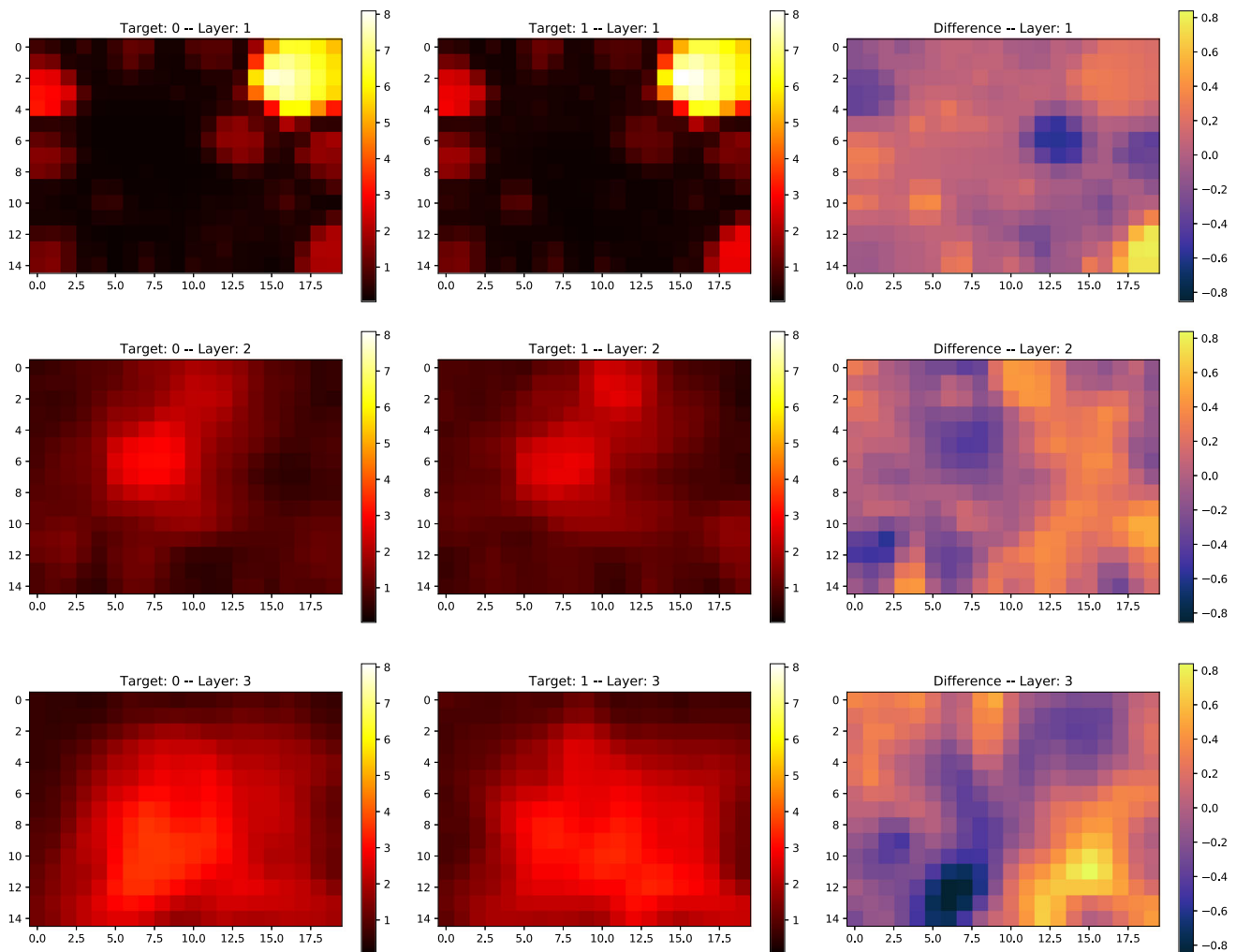
In all datasets, the area where the neurons are activated by the input widens with increasing depth, and in the first layer, it tends to be close to the lattice borders. This

**Table 8** Accuracy obtained training the model where the *Graph Convolution Layer Projection* were removed from the SOM-based aggregation blocks

Dataset	Pre-training	Readout type	Readout	Fine-tuning
PTC	57.71 ± 4.46	Shallow	58.00 ± 3.35	59.48 ± 2.44
	60.51 ± 1.61	Deep	59.89 ± 0.26	60.40 ± 0.37
NCII	81.51 ± 0.65	Shallow	78.05 ± 0.52	80.88 ± 0.82
	81.60 ± 0.23	Deep	81.67 ± 0.45	81.63 ± 0.25
Proteins	71.89 ± 0.91	Shallow	71.97 ± 0.08	72.44 ± 1.74
	72.29 ± 0.83	Deep	73.66 ± 0.60	73.48 ± 0.69
D&D	71.64 ± 1.37	Shallow	71.81 ± 1.27	72.78 ± 0.94
	71.40 ± 2.21	Deep	75.75 ± 0.41	76.18 ± 0.49
Enzymes	42.67 ± 0.73	Shallow	42.44 ± 3.58	44.72 ± 1.46
	42.16 ± 2.73	Deep	45.72 ± 0.75	46.61 ± 1.11

Note that the pre-training phase always uses a shallow pre-training readout, while the hyper-parameters could be different (see Table 3)





**Fig. 3** Heatmaps computed according to Eq. (30) on a batch of the NCI1 test set. Each image row presents from left to right: the heatmap obtained by considering only nodes that belong to graphs with negative target, the heatmap obtained by considering only nodes that

belong to graphs with positive target, and (on a different scale) the difference between the two heatmaps (positive minus negative). Each row reports the heatmaps relative to a single SOM-based aggregation block, from first (top) to third (bottom)

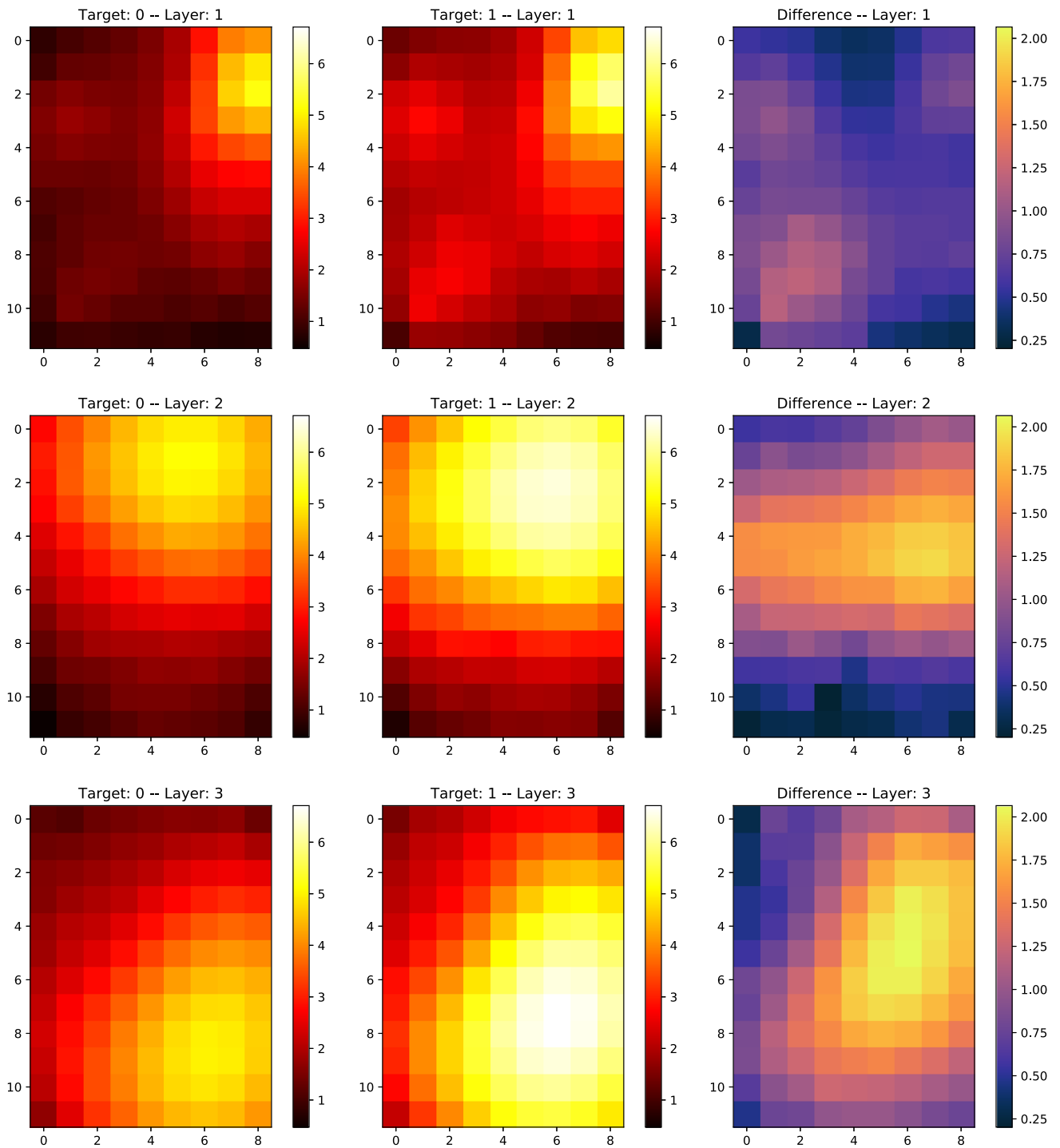
behavior could be due to the limited size of the input vocabulary. In NCI1 (Fig. 3) the representation of the positive and negative classes is quite similar in the first layer, while a more prominent differentiation arises in deeper layers. In PTC and in DD (Figs. 4 and 5, respectively) the visible distinction between the two classes could be noticed in the magnitude of the SOM activations, while the area of activation is partially overlapping. A very sharp difference between the two classes representations can be noticed in the PROTEINS dataset (Fig. 6). In fact, in this case both the magnitude and the distribution of the active SOM neurons seems to be very different.

In PTC, D&D and PROTEINS, the activations of the positive and the negative classes differ both in terms of the width of the area and in the magnitude of activations. Indeed, all these datasets have a ratio between the positive and the negative class different from 1. We argue that this

difference is related to the imbalanced class distribution of these datasets. Moreover, NCI1 is the only dataset that is almost perfectly balanced and the positive and negative heatmaps turn out to be more similar in terms of output magnitude.

## 6 Conclusion and future works

In this paper, we proposed a node aggregation scheme for graph convolutional neural networks inspired by DeepSets [32] that exploits self-organizing maps followed by graph convolutions to transform the node embeddings, and then aggregates them according to the DeepSets formulation in a fixed-size graph-level representation. Due to the unsupervised nature of the SOM training algorithm, we developed an ad-hoc training algorithm for supervised



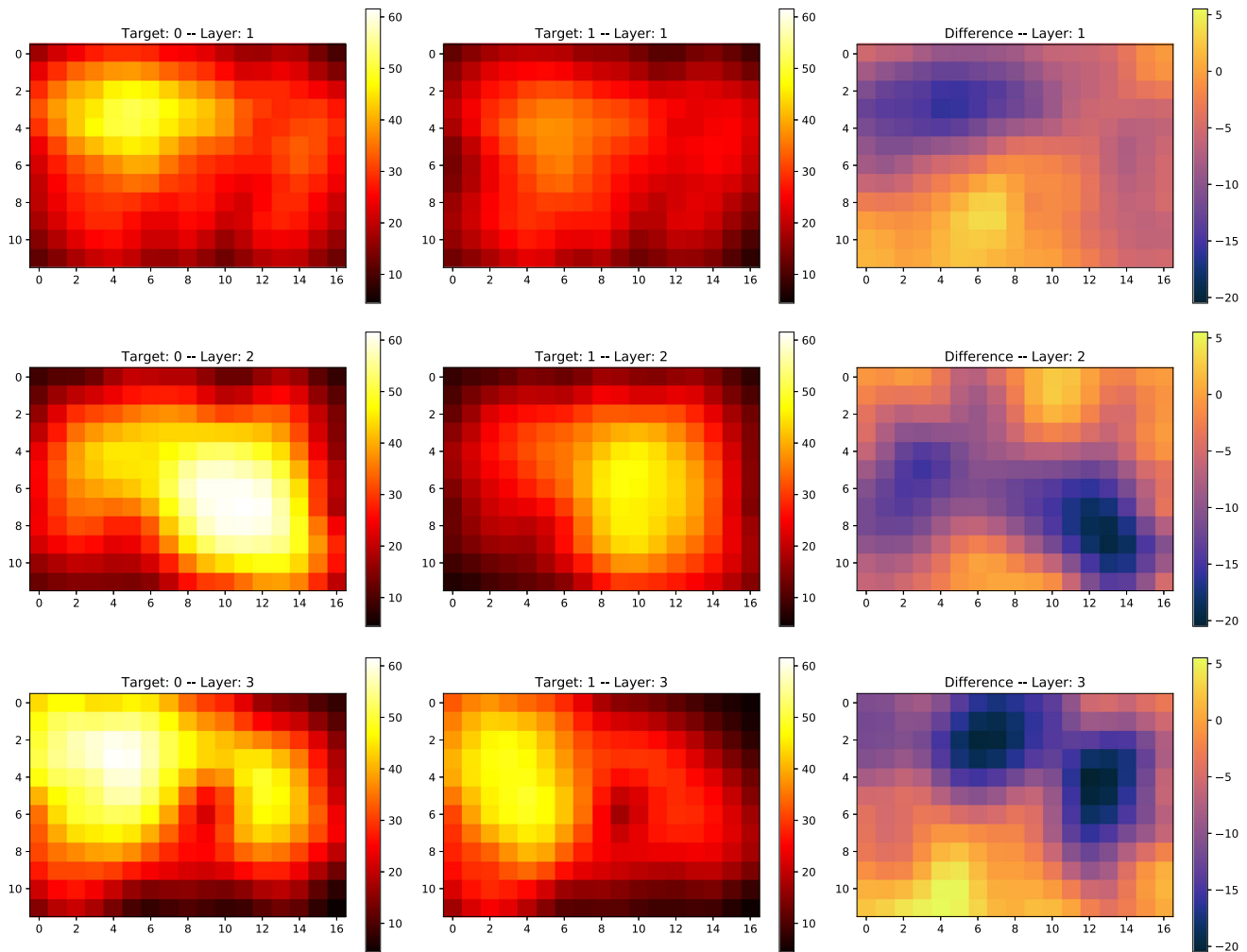
**Fig. 4** Heatmaps computed according to Eq. (30) on a batch of the PTC test set. Each image row presents from left to right: the heatmap obtained by considering only nodes that belong to graphs with negative target, the heatmap obtained by considering only nodes that

belong to graphs with positive target, and (on a different scale) the difference between the two heatmaps (positive minus negative). Each row reports the heatmaps relative to a single SOM-based aggregation block, from first (top) to third (bottom)

learning tasks to learn the parameters of the resulting graph neural network.

We empirically validated the proposed SOM-based aggregation method on five commonly adopted graph classification benchmarks modeling bioinformatics

problems. Experimental results show that our proposal achieves improved predictive performance compared to competing methods in the majority of the considered datasets. We investigated how much the SOM component helps in reaching these results. We showed that training the



**Fig. 5** Heatmaps computed according to Eq. (30) on a batch of the D&D test set. Each image row presents from left to right: the heatmap obtained by considering only nodes that belong to graphs with negative target, the heatmap obtained by considering only nodes that

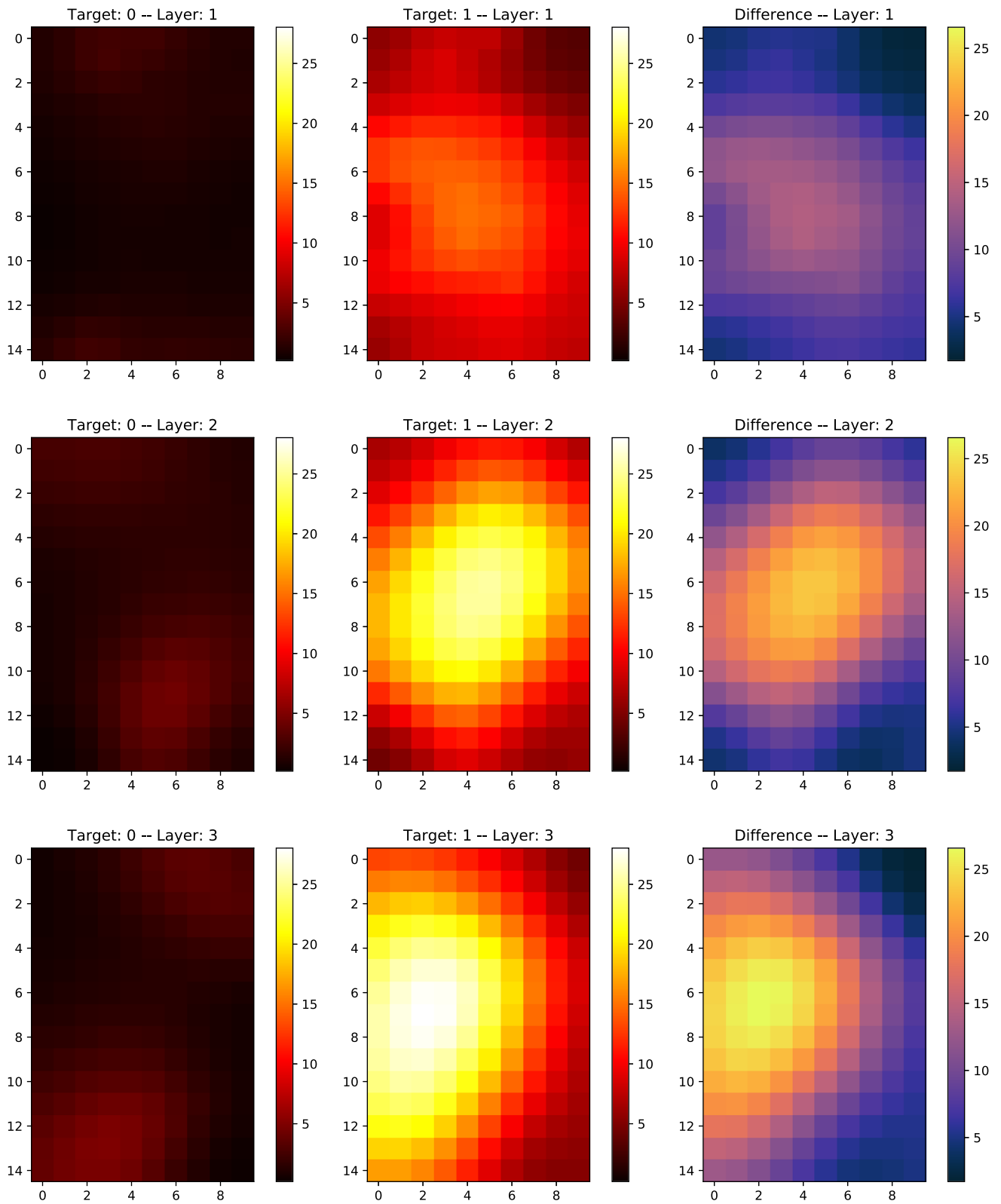
belong to graphs with positive target, and (on a different scale) the difference between the two heatmaps (positive minus negative). Each row reports the heatmaps relative to a single SOM-based aggregation block, from first (top) to third (bottom)

SOM is beneficial for the predictive performance, as well as stacking a GC projection layer after the SOM. Moreover, we studied the distances between node representations after the first graph convolution block, comparing them with the distances after applying the SOM mapping and the subsequent *Graph Convolution Layer Projection*. The comparison shows the benefit of using the SOM-based aggregation block to increase the distances between the node embeddings, thus potentially making the classification task easier.

Finally, thanks to the use of SOMs in the proposed model, we were also able to produce heatmaps representing each graph. By taking the average of such heatmaps over a

set of graphs, we studied how SOMs represent graphs belonging to different classes.

A limit of the proposed approach is its relatively high number of hyper-parameters, which makes the application of a complete grid search time-consuming. In the future, we plan to study and develop a simpler version of the SOM-based aggregation that depends on a significantly lower number of hyper-parameters. We will extend the study to different types of lattice configurations as well as to more advanced types of SOM models (e.g., the Generative Topographic Map (GTM) [2]). Finally, we will explore a fully supervised extension of our proposal exploiting supervised SOM training algorithms, e.g. the algorithm proposed by Hagenbuchner et al. [10].



**Fig. 6** Heatmaps computed according to Eq. (30) on a batch of the PROTEINS test set. Each image row presents from left to right: the heatmap obtained by considering only nodes that belong to graphs with negative target, the heatmap obtained by considering only nodes

that belong to graphs with positive target, and (on a different scale) the difference between the two heatmaps (positive minus negative). Each row reports the heatmaps relative to a single SOM-based aggregation block, from first (top) to third (bottom)

**Acknowledgements** This work has been supported by the University of Padova, Department of Mathematics, DEEP project.

**Funding** Open access funding provided by Università degli Studi di Padova within the CRUI-CARE Agreement.

**Compliance with ethical standard**

**Conflict of interest** The authors declare that they have no conflict of interest.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

**Appendix A: Graph convolutions**

In this section, we review some definitions of graph convolution in the literature, besides the ones used in the proposed architecture. In the following definitions, for the sake of simplicity, we ignore the bias terms. Scarselli et al. [24] proposed a recurrent neural network for graphs that adopts the following general form of convolution:

$$\mathbf{h}_v^{t+1} = \sum_{u \in \mathcal{N}(v)} f_{\theta}(\mathbf{h}_u^t, \mathbf{x}_v, \mathbf{x}_u), \tag{31}$$

where  $t \geq 0$  is the time of the recurrence,  $\mathbf{h}_v^{t+1}$  is the representation of node  $v$  at timestep  $t + 1$ ,  $f_{\theta}$  is a neural network whose parameters  $\theta$  have to be learned, and are shared among all the vertices. If edge labels are available, they can be easily included in Eq. (31). The recurrent system is defined as a contraction mapping, thus it is guaranteed to converge to a fixed point. Li et al. [17] extended the work of Scarselli et al. [24], removing the constraint for the recurrent system to be a contraction mapping.

Micheli [19] proposed the Neural Network for Graphs (NN4G) model. It is based on a graph convolution that is defined for the first and the  $(i + 1)$ -th layer (for  $i > 0$ ), respectively, as follows:

$$\mathbf{h}_v^{(1)} = f(\bar{\mathbf{W}}^{(1)} \mathbf{x}_v), \tag{32}$$

$$\mathbf{h}_v^{(i+1)} = f\left(\bar{\mathbf{W}}^{(i+1)} \mathbf{x}_v + \sum_{k=1}^i \hat{\mathbf{W}}^{(i+1,k)} \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^k\right), \tag{33}$$

where  $1 \leq v \leq n$  is the vertex index,  $\hat{\mathbf{W}}^{(l,m)}$  are the weights connecting the representations at the  $m$ -th and  $l$ -th layer, and  $\bar{\mathbf{W}}^{(i+1)}$  are weights transforming the input representations for the  $i + 1$ -th layer. Note that in this formulation, skip connections are present, to the  $(i + 1)$ -th layer, from layer 1 to layer  $i$ . Duvenaud et al. [7], proposed a hierarchical approach, similar to NN4G and inspired by circular fingerprints in chemical structures. While NN4G [19] adopted Cascade-Correlation for training, Duvenaud et al. used end-to-end back-propagation. ECC [26] developed an improvement of the approach introduced by Duvenaud et al. [7], weighting the sum over the neighbors of a vertex by weights conditioned by the edge labels.

Kipf and Welling [15] proposed a graph convolution that closely resembles NN4G in Eq. (33). Motivated by a first-order approximation of localized spectral filters on graphs introduced by Defferrard et al. [5], the graph convolutional filter is defined as follows:

$$\mathbf{H}^{(i+1)} = f(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(i)} \mathbf{W}^{(i)}), \tag{34}$$

where  $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ ,  $\tilde{d}_{ii} = \sum_j \tilde{a}_{i,j}$ ,  $f$  is any activation function applied element-wise, and  $\mathbf{H}^0 = \mathbf{X}$ , the matrix obtained by collecting as rows the labels of the graph. Note that in this equation, we compute the representation of all the vertices in the graph at once. It is easy to see that Eq. (34) is very similar to Eq. (33) with no skip connections. DGCNN [33] proposed a slight modification of the graph convolution in Eq. (34), defined as follows:

$$\mathbf{H}^{(i+1)} = f(\tilde{\mathbf{D}}^{-1} \tilde{\mathbf{A}} \mathbf{H}^{(i)} \mathbf{W}^{(i)}), \tag{35}$$

where  $\mathbf{H}^0 = \mathbf{X}$ , and  $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ . The difference between Eqs. (35) and (34) is the use of a different propagation scheme (different normalization of the graph Laplacian) to propagate vertices’ representations. In fact, both equations can be seen as first-order approximations of the polynomially parameterized spectral graph convolution [5]. Tran et al. [28], extended DGCNN with a hyper-parameter controlling the neighborhood distance considered in the convolution operation.

PATCHY-SAN [23] follows a more straightforward approach to define convolutions on graphs, which 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 vertex ordering. A standard CNNs is then applied since the dimension of the resulting representation is fixed. This approach requires the vertices of each input graph to be in a canonical ordering, which is as complex as the graph isomorphism problem (no polynomial-time



algorithm is known), which introduces a computational bottleneck for the method.

Diffusion CNN [1] defines a different graph convolution (i.e., diffusion-convolution) that incorporates in the definition of graph convolution the diffusion operator, i.e., the multiplication of the input representation with a power series of the degree-normalized transition matrix.

## Appendix B: State-of-the-art graph neural network architectures

In this section, we briefly review SOTA Graph Neural Network architectures.

The Funnel GCNN (FGCNN) model [21] aims to enhance the gradient propagation using a simple aggregation function and LeakyReLU activation functions. Hinging on the similarity of the adopted graph convolutional operator to the way feature space representations by Weisfeiler-Lehman (WL) Subtree Kernel [25] are generated, it introduces a loss term for the output of each convolutional layer to guide the network to reconstruct the corresponding explicit WL features. Moreover, the number of filters used at each convolutional layer is based on a measure of the WL kernel complexity. GraphSage [11] aggregates neighbor representations in the graph convolution using sum, mean or max-pooling operators, and then performs a linear projection in order to update the node representations. In addition to that, GraphSage exploits a particular neighbors sampling scheme. GIN [30] is an extension of GraphSage that defines a maximally expressive graph convolution. DiffPool [31] combines a differentiable graph encoder with a trainable pooling mechanism. Indeed, the method learns an adaptive pooling strategy to collapse nodes on the basis of a supervised criterion.

## References

- Atwood J, Towsley D (2016) Diffusion-convolutional neural networks. In: Advances in neural information processing systems, pp 1993–2001. [arXiv:1511.02136](#)
- Bishop CM, Svensén M, Williams CK (1998) GTM: The generative topographic mapping. *Neural Comput* 10(1):215–234
- Borgwardt KM, Ong CS, Schönauer S, Vishwanathan S, Smola AJ, Kriegel HP (2005) Protein function prediction via graph kernels. *Bioinformatics* 21(1):47–56
- Chen T, Bian S, Sun Y (2019) Are powerful graph neural nets necessary? A dissection on graph classification. [arXiv:1905.04579](#)
- Defferrard M, Bresson X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Neural information processing systems (NIPS) (2016). [arXiv:1606.09375](#)
- Dobson PD, Doig AJ (2003) Distinguishing enzyme structures from non-enzymes without alignments. *J Mol Biol* 330(4):771–783
- Duvenaud D, Maclaurin D, Aguilera-Iparraguirre J, Gómez-Bombarelli R, Hirzel T, Aspuru-Guzik A, Adams RP (2015) Convolutional networks on graphs for learning molecular fingerprints. In: Advances in neural information processing systems. Montreal, Canada, pp 2215–2223. [arXiv:1509.09292](#)
- 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>
- Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE (2017) Neural message passing for quantum chemistry. In: Proceedings of the 34th international conference on machine learning, pp 1263–1272. [arXiv:1704.01212](#)
- Hagenbuchner M, Tsoi AC, Sperduti A (2001) A supervised self-organizing map for structured data. In: Allinson NM, Yin H, Allinson L, Slack JM (eds) Advances in self-organising maps, WSOM 2001, Lincoln, UK, 13–15 June. Springer, pp 21–28. [https://doi.org/10.1007/978-1-4471-0715-6\\_4](https://doi.org/10.1007/978-1-4471-0715-6_4)
- Hamilton W, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. In: NIPS, pp 1024–1034
- Helma C, King RD, Kramer S, Srinivasan A (2001) The predictive toxicology challenge 2000–2001. *Bioinformatics* 17(1):107–108
- Kipf TN, Welling M (2016) Variational graph auto-encoders. In: NIPS workshop on Bayesian deep learning. [arXiv:1611.07308](#)
- Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: ICLR, pp 1–14. <https://doi.org/10.1051/0004-6361/201527329>. [arXiv:1609.02907](#)
- Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: ICLR. <https://doi.org/10.1051/0004-6361/201527329>. [arXiv:1609.02907](#)
- Kohonen T (1990) The self-organizing map. *Proc IEEE* 78(9):1464–1480
- Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks. In: ICLR (2016). <https://doi.org/10.1103/PhysRevLett.116.082003>. [arXiv:1511.05493](#)
- Liu, Q., Allamanis, M., Brockschmidt, M., Gaunt, A.L.: Constrained graph variational autoencoders for molecule design. In: NeurIPS (2018). [arXiv:1805.09076](#)
- Micheli A (2009) Neural network for graphs: a contextual constructive approach. *IEEE Trans. Neural Netw.* 20(3):498–511. <https://doi.org/10.1109/TNN.2008.2010350>
- Morris C, Ritzert M, Fey M, Hamilton WL, Lenssen JE, Rattan G, Grohe M (2019) Weisfeiler and Leman Go Neural: higher-order graph neural networks. In: Proceedings of the AAAI conference on artificial intelligence, vol. 33, pp 4602–4609. <https://doi.org/10.1609/aaai.v33i01.33014602>. [arXiv:1810.02244](#)
- Navarin N, Tran DV, Sperduti A (2019) Universal readout for graph convolutional neural networks. In: International joint conference on neural networks. Budapest, Hungary
- Navarin N, Tran DV, Sperduti A (2020) Learning kernel-based embeddings in graph neural networks. In: European conference on artificial intelligence
- Niepert M, Ahmed M, Kutzkov K (2016) Learning convolutional neural networks for graphs. In: ICML, pp 2014–2023
- Scarselli F, Gori M, Ah Chung Tsoi AC, Hagenbuchner M, Monfardini G (2009) The graph neural network model. *IEEE Trans Neural Netw* 20(1):61–80. <https://doi.org/10.1109/TNN.2008.2005605>
- Shervashidze N, Schweitzer P, Van Leeuwen EJ, Mehlhorn K, Borgwardt KM (2011) Weisfeiler–Lehman graph kernels. *J Mach Learn Res* 12(77):2539–2561

26. Simonovsky M, Komodakis N (2017) Dynamic edge-conditioned filters in convolutional neural networks on graphs. [arxiv:1704.02901](https://arxiv.org/abs/1704.02901). Comment: Accepted to CVPR 2017; extended version
27. Sperduti A, Starita A (1997) Supervised neural networks for the classification of structures. *IEEE Trans Neural Netw* 8(3):714–735. <https://doi.org/10.1109/72.572108>
28. Tran DV, Navarin N, Sperduti A (2018) On filter size in graph convolutional networks. In: *IEEE SSCI*. IEEE, Bengaluru, pp 1534–1541. <https://doi.org/10.1109/SSCI.2018.8628758>. <https://ieeexplore.ieee.org/document/8628758/>
29. Wale N, Watson IA, Karypis G (2008) Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl Inf Syst* 14(3):347–375
30. Xu K, Hu W, Leskovec J, Jegelka S (2019) How powerful are graph neural networks? In: *International conference on learning representations*
31. Ying Z, You J, Morris C, Ren X, Hamilton W, Leskovec J (2018) Hierarchical graph representation learning with differentiable pooling. In: *Advances in neural information processing systems*, pp 4800–4810
32. Zaheer M, Kottur S, Ravanbakhsh S, Póczos B, Salakhutdinov R, Smola AJ (2017) Deep sets. In: *Advances in neural information processing systems*, pp 3391–3401 (2017)
33. Zhang M, Cui Z, Neumann M, Chen Y (2018) An end-to-end deep learning architecture for graph classification. In: *AAAI conference on artificial intelligence*
34. Zhang M, Cui Z, Neumann M, Chen Y (2018) An end-to-end deep learning architecture for graph classification. In: *Thirty-second AAAI conference on artificial intelligence*

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.