ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2.
Available from http://www.i6doc.com/en/.

Pyramidal Graph Echo State Networks

Filippo M. Bianchi[1], Claudio Gallicchio[2], Alessio Micheli[2] *

1- NORCE - the Norwegian Research Center
2- University of Pisa

Abstract. We analyze graph neural network models that combine iterative message-passing implemented by a function with untrained weights and graph pooling operations. In particular, we alternate randomized neural message passing with graph coarsening operations, which provide multiple views of the underlying graph. Each view is concatenated to build a graph embedding for graph-level classification. The main advantage of the proposed architecture is its speed, further improved by the pooling, in computing graph-level representations. Results obtained on popular graph classification benchmarks, comparing different topological pooling techniques, support our claim.

1 Introduction

A class of recent machine learning models, called Graph Neural Networks (GNNs), learns how to represent entities and how to combine them, according to arbitrary relationships given as part of the task inputs. GNNs leverage the structure of the data to perform inference, by learning simultaneously the vertex's features and its network context, i.e. the features of vertexes in its neighborhood [2].

Global graph representations can be derived by combining at once the vertex features and use them for inference on downstream tasks at graph level. However, some networks may exhibit scale-dependent behaviors, which have to be accounted for when solving the task. In these cases, pooling operations are exploited to build deep GNN architectures, which compute local summaries on the graph to gradually distill the global properties necessary for graph-level inference [4, 1, 3, 11].

So far, pooling operations have been applied only to GNNs implemented with message passing (MP) operations with parameters learned with gradient descent from a supervised loss [7]. In this work instead, we explore the synergy of graph pooling with randomized neural MP operations in terms of classification accuracy and the computing time necessary to generate the graph embeddings.

We build a deep GNN composed of Graph Echo State Network (GESN) [5] layers interleaved with graph pooling operations. Graph pooling reduces the dimension of the graph by clustering or by dropping vertices after a propagation operation (see Fig. 1). Since the adopted GNN architecture generates untrained embeddings, we consider only pooling methods that pre-compute the coarsened graphs without supervision. The potentiality of hierarchical GESN architectures in designing fast and deep models for graph classification has been recently shown in [6]. In this context, the benefit of pooling would be to further reduce

*This work is funded by the mobility grant of University of Pisa.

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2.
Available from http://www.i6doc.com/en/.

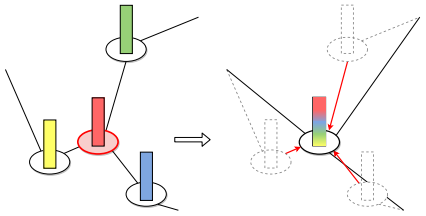

Fig. 1: After a neural message passing operation, a vertex acquires information from its neighborhood and, thus, can be used as a local representative. The other vertexes can be dropped since their features are now contained in the representative, reducing the dimensionality of the graph.

the computational complexity. In the experiments, we investigate how training time and classification accuracy change when using 3 different graph pooling approaches, and we also compare to a similar GNN architecture without pooling.

2 Methodology

Let $\mathcal{G} = \{\mathbf{A}, \mathbf{X}\}$ be an undirected graph with N vertexes, characterized by a symmetric adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$. We define the symmetrically normalized adjacency matrix $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{D}$ is a diagonal degree matrix s.t. D_{ii} is the degree of vertex i. Let $\mathbf{X} \in \mathbb{R}^{N \times F}$ be the matrix whose i-th row contains the feature vector associated to the i-th vertex.

2.1 Graph ESN

Reservoir computing (RC) is an established paradigm for modeling nonlinear temporal sequences [8]. In machine learning tasks, Echo State Networks (ESNs) are the most popular RC model, wherein the input sequence is projected to a high-dimensional space through the use of a (fixed) nonlinear recurrent reservoir. Learning is applied only to a linear readout layer, whereas the dynamical reservoir is initialized under stability constraints and then is left untrained [8]. The lack of supervised tuning in the recurrent part implies a range of advantages, including faster training compared to other neural networks.

An RC model for graphs has been introduced under the name of Graph Echo State Network (GESN) [5]. A GESN updates the vertex features according to:

$$\mathbf{H}^{(t+1)} = tanh\left(\tilde{\mathbf{A}}\mathbf{H}^{(t)}\mathbf{W} + \mathbf{X}\mathbf{V}\right), \tag{1}$$

where $\mathbf{H} \in \mathbb{R}^{N \times H}$ collects the reservoir states computed for all the vertices of an input graph, $\mathbf{W} \in \mathbb{R}^{H \times H}$ and $\mathbf{V} \in \mathbb{R}^{H \times F}$ are the recurrence and input weight matrix, respectively. To build the graph embedding, (1) is iterated until convergence to a steady state. Conditions to ensure the existence and uniqueness of such steady state have been studied in [6]. Accordingly, elements in $\mathbf{W}$ are

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2.
Available from http://www.i6doc.com/en/.

randomly initialized from a uniform distribution in $[-0.5, 0.5]$, and then are re-scaled to get a spectral radius $\rho < 1$. Weights in $\mathbf{V}$ are initialized similarly and then are re-scaled by a hyper-parameter ω. In our model, the MP operations are implemented as in (1). When more GESN layers are stacked, input weights of layers > 1 are re-scaled by a hyper-parameter ω_{in}.

2.2 Graph pooling methods

Differentiable pooling operators compute cluster assignments by means of functions that depend on the vertex features and are parameterized by weights learned end-to-end, according to the data and the task at hand [11]. On the other hand, topological pooling methods account only for the graph topology ($\mathbf{A}$) and are usually unsupervised, as they define how to coarsen the graph outside of the learning procedure. After each MP layer, the current vertex representations are fit to these pre-determined structures. Obviously, GNNs with pre-computed coarsened graphs are much faster to train. In the following, we describe the three main approaches used to perform topological graph pooling.
Graclus pooling [4] is the first approach that has been proposed to perform pooling within a GNN architecture and consists in coarsening the graph with Graclus, a hierarchical spectral clustering algorithm. At each layer l of the GNN, two vertices $x_i^{(l)}$ and $x_j^{(l)}$ are clustered together and form a new vertex $x_z^{(l+1)}$. Afterwards, max pooling is applied on the vertex features to be clustered together to halve the size of the graph. If the number of vertexes in the graph is not divisible by 2^L, where L is the number of pooling operations, fake vertices must be added so that the number of vertexes can be halved after each time.
Non-negative Matrix Factorization (NMF) pooling [1] aggregates the vertices by clustering the rows (columns) of the adjacency matrix, rather than performing spectral clustering. The approach is based on the NMF of the adjacency matrix $\mathbf{A} \approx \mathbf{QS}$, which has the inherent property of clustering the rows (or the columns). It is possible to interpret $\mathbf{Q} \in \mathbb{R}^{N \times K}$ as the cluster representatives matrix and $\mathbf{S} \in \mathbb{R}^{K \times N}$ as a soft-clustering of the columns in $\mathbf{A}$. The pooled vertex features and the coarsened graph are computed as $\mathbf{X}^{\text{pool}} = \mathbf{S}^T\mathbf{X}$ and $\mathbf{A}^{\text{pool}} = \mathbf{S}^T\mathbf{XS}$, respectively.
Node Decimation Pooling (NDP) [3] reduces the graph by dropping the vertices that, after an MP operation, end up containing the most similar features. Specifically, the vertices are partitioned in two maximally similar groups via MAXCUT. One group is dropped, while the other is kept and a new graph is built by connecting the retained vertices by using the Kron reduction [10]. Finally, a sparsification procedure is used to remove from the coarsened graphs edges with small weights. Vertex pooling is performed by multiplying a decimation matrix $\mathbf{S}$, obtained by keeping in the identity matrix the rows corresponding to the retained vertices, with the vertex features: $\mathbf{X}^{\text{pool}} = \mathbf{S}^T\mathbf{X}$.

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2. Available from http://www.i6doc.com/en/.

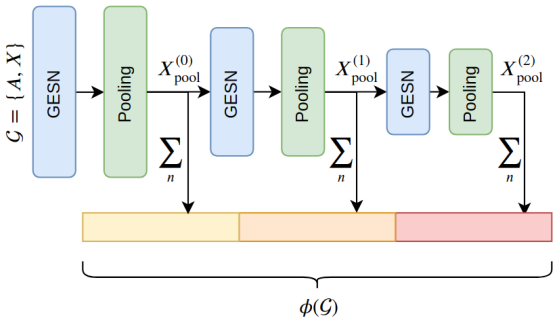

Fig. 2: Architecture of the proposed model. After each pooling layer, all the vertex features are summed into single vector. In turn, such vectors are concatenated to obtain a graph level representation $\phi(\mathcal{G})$.

2.3 Proposed architecture

We propose an architecture composed of L blocks. The l-th block consists of a GESN layer, which computes the new vertex features, and a pooling layer, which yields a pooled version of the vertex features $\mathbf{X}^{(l)}_{\text{pool}}$ and the coarsened adjacency matrix $\mathbf{A}^{(l)}_{\text{pool}}$. After pooling, we obtain a single vector $\bar{\mathbf{x}}^{(l)} = \sum_n \mathbf{X}^{(l)}_{\text{pool}}[n, :]$ by summing all the vertex features. By concatenating L of these vectors, we obtain the final graph embedding $\phi(\mathcal{G})$. Fig. 2 reports a schematic depiction of the whole procedure for $L = 3$.

The final graph classification is performed by a commonly used classifier for vectorial data [9], which implements the function $y_i = tanh\left(\left(\mathbf{W}_\phi \phi(\mathcal{G})\right)\right)\mathbf{W}_{\text{out}}$, where $\mathbf{W}_\phi$ is a randomized matrix, projecting the graph embedding into high dimensionality. $\mathbf{W}_{\text{out}}$ are the only trainable parameters of the whole model and learned by ridge regression with hyperparameter λ.

3 Experiments

We consider different graph classification tasks, where the i-th datum is a graph described by the pair $\{\mathbf{A}_i, \mathbf{X}_i\}$ and must be associated to the correct label y_i. We test the models on 5 benchmark datasets for graph classification[1], i.e. MUTAG, PROTEINS, DD, NCI1, COLLAB. For featureless graphs, we used the vertex degree information and the clustering coefficient as surrogate vertex features. We evaluate the model with nested cross-validation with 10 folds in the inner and outer loop. We fixed $\rho = 0.5$, $\omega = 0.5$, $\omega_{in} = 0.8$, $L = 3$ and the size of the reservoir to $H = 100$, except for MUTAG where $H = 50$ since the samples are few and the average number of vertices is small. In the inner loop, the regularization parameter λ is cross-validated and chosen by model selection.

The objective of our analysis is to evaluate the trade-off between classification

[1] `https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets`

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2. Available from http://www.i6doc.com/en/.

accuracy and training time, when equipping a deep GESN model with pooling layers. As baseline, we consider a deep state-of-the-art model [6], which we instantiate by stacking 3 GESN layers. We compare models where the GESN layers are interleaved by the 3 pooling operations: Graclus, NMF, and NDP. Classification accuracy and training times computed, for each dataset, as the average on the test folds are reported in Tab. 1. For each dataset, we also report the relative differences in percentage with respect to the lowest training time and highest classification accuracy. A graphical aggregation of the results is in Fig. 3, that compares the deviations from the lowest training time (X-axis, the smaller the better) and from the highest accuracy (Y-axis, the higher the better) obtained as an average on all datasets.

Table 1. Classification accuracy and training time (seconds). Higher accuracy and lower times for each dataset are in bold font.

	No-pool		**Pool** (Graclus)		**Pool** (NMF)		**Pool** (NDP)	
	Acc.	**Time**	**Acc.**	**Time**	**Acc.**	**Time**	**Acc.**	**Time**
MUTAG	**91.4**$_{\pm 5.5}$ -0%	0.3s +50%	88.9$_{\pm 6.8}$ -2.7%	0.5s +150%	86.2$_{\pm 6.3}$ -5.6%	0.4s +100%	89.4$_{\pm 6.0}$ -2.2%	**0.2s** +0%
PROTEINS	**77.9**$_{\pm 2.7}$ -0%	5.3s +152%	76.2$_{\pm 3.8}$ -2.2%	6.2s +195%	73.6$_{\pm 5.8}$ -5.8%	2.7s +28%	76.9$_{\pm 2.5}$ -1.3%	**2.1s** +0%
DD	**80.9**$_{\pm 4.1}$ -0%	171.8s +92.4%	78.8$_{\pm 6.0}$ -2.6%	119.0s +33.2%	78.5$_{\pm 3.2}$ -3.0%	143.2s +60.3%	79.6$_{\pm 2.8}$ -1.6%	**89.3s** +0%
NCI1	**78.3**$_{\pm 1.4}$ -0%	49.0s +54.5%	72.6$_{\pm 2.7}$ -7.2%	**31.7s** +0%	69.2$_{\pm 1.6}$ -11.6%	37.2s +15.8%	74.9$_{\pm 1.7}$ -4.3%	35.4s +10%
COLLAB	77.5$_{\pm 1.6}$ -0.4%	110.9s +57.1%	**77.8**$_{\pm 1.2}$ -0%	64.5s +26.2%	74.4$_{\pm 1.4}$ -4.3%	51.3s +7.2%	75.7$_{\pm 1.1}$ -2.7%	**47.6s** +0%

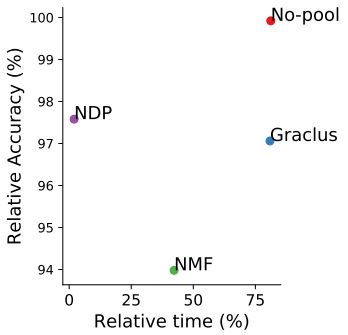

Fig. 3. Average increment (in %) in computing time (X-axis, the lower the better) and average decrement (in %) in classification accuracy (Y-axis), compared to the best performing model on each dataset. For each model M: X-axis = $(time_M - time_{min})/time_{min} * 100$, Y-axis = $(acc_M/acc_{max}) * 100$. Results are computed individually on each dataset, and then averaged.

While in fully trainable GNN architectures alternating graph convolutions with pooling layers helps, in several tasks, to improve the classification accuracy [3], the same does not happen when using randomized architectures such as GESN. However, the results show that when using pooling it is possible to obtain a significant speed-up in computing the graph embeddings $\phi(\mathcal{G})$; this is

ESANN 2020 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Online event, 2-4 October 2020, i6doc.com publ., ISBN 978-2-87587-074-2. Available from http://www.i6doc.com/en/.

naturally obtained by the proposed construction as the 2nd and 3rd GESN layer process a graph that has 1/2 and 1/4 of the size of the original one, respectively. In particular, when the pooling procedure is implemented by NDP, it is possible to obtain the highest speed-up and, at the same time, loosing only a few percentage points in the classification accuracy. Indeed, NDP is based on node decimation, which generates sparser yet well-formed coarsened graphs, compared to the other pooling approaches based on vertex clustering.

4 Conclusions

In this paper, we performed the first exploratory experimental study on combining randomized graph convolutions with pooling operations, to obtain a graph representations from multiple views on the graph at different level of resolutions. The results show that, contrarily to fully-trainable GNNs, the proposed models with pooling often do not improve the classification accuracy but, on the other hand, consistently diminish the training time. In particular, when using the Node Decimation Pooling strategy, it is possible to significantly reduce the model complexity at the cost of loosing only few points in accuracy. The proposed model provides an advantageous trade-off in those situations where computational resources are scarce, such as in embedded systems. Finally, our work paved the road for further research in pooling strategies for randomized architectures, such as the Graph Echo State Network.

References

[1] D. Bacciu and L. Di Sotto. A non-negative factorization approach to node pooling in graph convolutional neural networks. In *AI*IA 2019 – Advances in Artificial Intelligence*, pages 294–306. Springer International Publishing, 2019.

[2] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. Graph neural networks with convolutional arma filters. *arXiv preprint arXiv:1901.01343*, 2019.

[3] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi. Hierarchical representation learning in graph neural networks with node decimation pooling. *arXiv:1910.11436*, 2019.

[4] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3844–3852, 2016.

[5] C. Gallicchio and A. Micheli. Graph echo state networks. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8. IEEE, 2010.

[6] C. Gallicchio and A. Micheli. Fast and deep graph neural networks. In *Proceedings of AAAI*, 2020. arXiv preprint arXiv:1911.08941.

[7] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[8] M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 2009.

[9] S. Scardapane and D. Wang. Randomness in neural networks: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2017.

[10] D. Shuman, M. J. Faraji, and P. Vandergheynst. A multiscale pyramid transform for graph signals. *IEEE Transactions on Signal Processing*, 64(8):2119–2134, 2016.

[11] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. *NeurIPS*, 2019.