Deep Lagrangian Propagation in Graph Neural Networks

Matteo Tiezzi [1] **Giuseppe Marra** [2] **Stefano Melacci** [1] **Marco Maggini** [1]

Abstract

Graph Neural Networks (Scarselli et al., 2009) exploit an iterative diffusion procedure to compute the node states as the fixed point of the trainable state transition function. In this paper, we show how to cast this scheme as a constrained optimization problem, thus avoiding the unfolding procedure required for the computation of the fixed point. This is done by searching for saddle points of the Lagrangian function in the space of the weights, state variables and Lagrange multipliers. The proposed approach shows state-of-the-art performance in multiple standard benchmarks in graph domains.

1. Introduction

Graph Neural Networks (GNNs) (Scarselli et al., 2009) are neural models capable of learning task-dependent representations of the nodes of a graph, exploiting both the features attached to each node and the graph topology. In particular, given an input graph $G = (V, E)$, where V is the finite set of *nodes* and $E \subseteq V \times V$ collects the *arcs*, GNNs perform the *encoding* (or *aggregation*) phase yielding a state vector for each node in V by (iteratively) combining the states of neighboring nodes; then, in the *output* (or *readout*) phase, the node states are exploited to compute the model output.

Hence, the GNN processing scheme is defined by the *state transition* function f_a and the *output* function f_r, as follows:

$$x_v^{(t+1)} = f_a(x_{ne[v]}^{(t)}, l_{ne[v]}, l_v, l_{(ne[v],v)}|\theta_{f_a}) \quad (1)$$

$$y_v = f_r(x_v^{(T)}|\theta_{f_r}); \quad (2)$$

where $x_v^{(t)} \in \mathbb{R}^s$ is the state of node v at iteration t, $\text{ne}[v]$ are the *neighbors* of v and $l_v, l_{ne[v]} \in \mathbb{R}^m$, $l_{(\text{ne}[v],v)} \in \mathbb{R}^d$ encode additional information (sometimes referred as *labels*) on the node v, on its neighbors and on the arcs connecting them. The vectors θ_{f_a} and θ_{f_r} collect the GNN parameters (i.e. the weights of the neural networks implementing the two functions) that are adapted by the learning procedure. Different implementations have been proposed for f_a (Xu et al., 2018).

The application of the state transition function f_a fosters a diffusion mechanism on the graph, that is iterated for T steps to yield the node state representations. In fact, by applying t times the aggregation of 1-hop neighborhoods by f_a, the information at a given node can be propagated to the nodes that are at most t-hops away. In the original GNN model (Scarselli et al., 2009), Eq. (1) is repeated until convergence of the state representation, i.e. until $x_v^{(T)} \simeq x_v^{(T-1)}, v \in V$. This means that f_a reached its fixed point, hence satisfying the constraint[1]:

$$\forall v \in V, x_v = f_{a,v} \; . \quad (3)$$

Whilst being abandoned in recent literature (Kipf & Welling, 2017) because of its high computational burden, the fixed point formulation of GNNs is more general than executing a fixed, small, number of aggregation iterations.

In this paper, we propose a new learning mechanism for the original GNNs, which recovers the fixed point computation of the transition function. In particular, node states $x_v, v \in V$ are considered as new parameters of the problem. The learning algorithm, framed in the Lagrangian setting, seeks for node states and network parameters that maximize simultaneously the satisfaction of the fixed point constraint of Eq. (3) and the learning objective. By straightforwardly exploiting existing works in constraint-based neural networks (Carreira-Perpinan & Wang, 2014; Marra et al., 2020), the proposed model, hereafter referred to as Lagrangian Propagation GNN (LP-GNN), realizes a multi-layer GNN scheme by optimizing multiple stacked representations of each node by means of a pipeline of fixed-point constraints. The proposed method shows state-of-the-art performance in several benchmarks.

The paper is structured as follows. In Section 2, we present related approaches. Then, in Section 3 the LP-GNN formulation is presented. Section 4 reports the experimental evaluation of the proposed scheme. Finally, Section 5 draws

[1]Department of Information Engineering and Mathematical Sciences, University of Siena, Italy [2]KU Leuven, Leuven, Belgium. Correspondence to: Matteo Tiezzi <mtiezzi@diism.unisi.it>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 108, 2020.

[1]Henceforth, $f_{a,v}$ is used to denote the state transition function applied to a node $v \in V$ as in Eq. (1).

the conclusions.

2. Related Works

A constrained fixed-point formulation for neural networks is used in other approaches, such as SSE (Dai et al., 2018) where the *policy iteration* algorithm is applied for the interleaved evaluation of the fixed point equation and the improvement of the transition and output functions.

The costly iterative procedure aimed at computing the fixed point of the transition function has been simplified (Li et al., 2016) and then removed in the recent literature, which is now extremely heterogeneous. *Spectral approaches* exploit particular embeddings of the graph and the spectral convolution (Bruna et al., 2014). Other works are based on smooth reparametrization (Henaff et al., 2015) or approximation of the spectral filters (Defferrard et al., 2016). Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017) are based on filters considering a 1-hop neighborhood of each node. *Spatial methods*, like PATCHY-SAN (Niepert et al., 2016; Duvenaud et al., 2015), DCNNs (Atwood & Towsley, 2016), GraphSAGE (Hamilton et al., 2017), GATs (Veličković et al., 2017), DGCNN (Zhang et al., 2018), GIN (Xu et al., 2018), exploit directly the graph topology, without the need of an intermediate representation. AWE (Ivanov & Burnaev, 2018) exploits kernel-based methods with a learning-based approach to learn graph embeddings.

In a more general context, neural network learning can be cast as a Lagrangian optimization problem (LeCun et al. (1988), Carreira-Perpinan & Wang (2014), Taylor et al. (2016), Marra et al. (2020)), by a formulation that requires the minimization of the classical data fitting loss and the satisfaction of a set of *architectural* constraints that describe the computation performed on the data. Then, the solution is computed by finding the *saddle points* of the associated Lagrangian in the space defined by the original network parameters and the *Lagrange multipliers*.

3. Deep Lagrangian Propagation GNNs

Let us introduce a set of K states for each node $v \in V$, organized into K layers, $\{x_{v,k}, k = 0, \dots, K-1\}$. At the layers $k > 0$, the state computed at the previous layer $k-1$ is considered as an additional input of the state transition function f_a^k (Bianchini et al., 2018). The states x_v^k of layer k can be defined as the fixed point of

$$x_{v,k} = f_a^k(x_{ne[v],k}, x_{v,k-1}, l_{ne[v]}, l_v, l_{(v,ne[v])}|\theta_{f_a}) \quad (4)$$

and they can be computed layer by layer from layer 0 to layer $K-1$, being $x_{v,k-1}$ constant when computing $x_{v,k}$. For compactness, in the following Eq. 4 will be denoted as $x_v^k = f_{a,v}^k$.

By adding free variables corresponding to the node states x_v^k, Eq. 4 defines a constraint on x_v^k as follows:

$$\mathcal{G}(x_{v,k} - f_{a,v}^k) = 0, \quad \forall\, v \in V,\ \forall\, k \in [0, K-1] \quad (5)$$

where $\mathcal{G}(x)$ is a function characterized by $\mathcal{G}(0) = 0$, such that the satisfaction of the constraints implies the solution of Eq. (4). Possible choices are $\mathcal{G}(x) = x$, $\mathcal{G}(x) = x^2$, or $\mathcal{G}(x) = \max(||x||_1 - \epsilon, 0)$, where $\epsilon \geq 0$ is a parameter that, when set to a small positive value, allows a given tolerance in the satisfaction of the constraint.

Consider a node-focused task, such that for some (or all) nodes $v \in S \subseteq V$ of the input graph G, a target output y_v is provided as a supervision[2]. Let $L(f_r(x_v|\theta_{fr_r}), y_v)$ be the loss function used to measure the target fitting approximation for node $v \in S$. Then, we can define the learning task as the following constrained optimization problem:

$$\begin{aligned} &\min_{\Theta_{f_a}, \theta_{f_r}, X} \sum_{v \in S} L(f_r(x_{v,K-1} \mid \theta_{f_r}), y_v) \\ &\text{s.t.} \quad \mathcal{G}(x_{v,k} - f_{a,v}^k) = 0, \quad \forall\, v \in V,\ \forall\, k \in [0, K-1] \end{aligned} \quad (6)$$

where $\Theta_{f_a} = \left[\theta_{f_a^0}, \dots, \theta_{f_a^{K-1}}\right]$ collects the weights of the neural networks implementing the transition function of each layer, and X collects all the states $x_{v,k}$. By introducing a Lagrange multiplier λ_v^k for each constraint, we define the Lagrangian associated to the problem of Eq. (6), as

$$\begin{aligned} \mathcal{L}(\theta_{f_a}, \theta_{f_r}, X, \Lambda) = \sum_{v \in S} \Big[& L(f_r(x_v \mid \theta_{f_r}), y_v) + \\ & + \sum_{k=0}^{K-1} \lambda_v^k \mathcal{G}\left(x_v - f_{a,v}\right) \Big] \end{aligned} \quad (7)$$

where Λ is the set of the Lagrangian multipliers λ_v^k. Finally, the unconstrained optimization problem is defined as the search for saddle points in the adjoint space $(\Theta_{f_a}, \theta_{f_r}, X, \Lambda)$ as

$$\min_{\Theta_{f_a}, \theta_{f_r}, X} \max_{\Lambda} \ \mathcal{L}(\Theta_{f_a}, \theta_{f_r}, X, \Lambda) \quad (8)$$

that can be solved by gradient descent with respect to the variables $\Theta_{f_a}, \theta_{f_r}, X$ and gradient ascent with respect to the Lagrange multipliers Λ (Platt & Barr, 1988). The gradient can be computed locally to each node, given the local variables and those of the neighboring nodes (see Appendix A).

Even if the proposed formulation adds the free state variables x_v and the Lagrange multipliers $\lambda_v^k, \forall\, v \in V, \forall\, k \in$

[2] We consider only the case when a single graph is provided for learning. The extension for more graphs is straightforward, since they can be considered as a single graph composed by the given graphs as disconnected components.

$[0, K-1]$, there is no significant increase in the memory requirements. The persistent state variable matrix requires $\mathcal{O}(K|V|)$, where $|V|$ denotes the number of nodes. Similarly to other common graph neural models, we exploit synchronous updates among all nodes and a stacked diffusion process for the node state embedding computation, with a computational complexity for each parameter update of $\mathcal{O}(K(|V|+|E|))$, where $|E|$ is the number of edges. In the proposed algorithm, the diffusion process is turned itself into an optimization process that must be carried out both when learning and when making predictions. This is deeply different both from original GNNs (Scarselli et al., 2009), where it is realized by the unrolling of the transition function, and from Graph Convolutional Networks (GCN) (Kipf & Welling, 2017), where it is embedded into the layer-wise projection. Differently from (Carreira-Perpinan & Wang, 2014), the constraint scheme is exploited only to enforce the fixed-point computation, while backpropagation is still exploited to efficiently update the weights of the neural networks f_a and f_r. Finally, in the proposed approach, there is a strict distinction between the deep feature extraction of the architecture and the diffusion mechanism. Diffusion is realised by looking for a fixed point of the state transition function, while deep feature extraction is realised by stacking multiple layers of node states, enabling a separate diffusion process at each layer. These two aspects are instead mixed in GCNs, where deep architectures are needed both to spread information among nodes and to extract more meaningful features for the task at hand.

4. Experiments

We implemented the algorithm described in the previous sections using TensorFlow. For the experimental setting see Appendix B.

4.1. Artificial Tasks

We consider two classical graph processing tasks, *subgraph matching*, i.e. identifying nodes belonging to a specific subgraph, and *clique localization*, i.e. detecting nodes belonging to a clique. We compared different functions $\mathcal{G}(x)$ to enforce the constraints: *ϵ-insensitive functions*, i.e $\mathcal{G}(x) = 0,\ \forall x : -\epsilon \leq x \leq \epsilon$; *unilateral functions*, i.e. $\mathcal{G}(x) \in \mathbb{R}^+$; and *bilateral functions*, i.e. $\mathcal{G}(x) \in \mathbb{R}$ (a $\mathcal{G}$ function is either unilateral or bilateral). In particular, we considered: $\max(x, \epsilon) - \max(-x, \epsilon)$ (*lin*), $\max(|x| - \epsilon, 0)$ (*abs*), and x^2 (*squared*). Results are presented in Table 1. Unilateral functions usually yield better performances than equivalent bilateral constraints. This might be due to the fact that keeping constraints positive (as in unilateral constraints) provides a more stable learning process. Moreover, smoother constraints (i.e *squared*) or ϵ–insensitive constraints tend to perform slightly better than the hard versions. This can be due to the fact that as the constraints are close to 0 they tend to give a small or null contribution, for *squared* and *abs*-ϵ respectively, acting as regularizers.

Model	ϵ	Subgraph		Clique	
LP-abs	0.00	96.25	0.96	88.80	4.82
	0.01	**96.30**	0.87	88.75	5.03
	0.10	95.80	0.85	85.88	4.13
LP-lin	0.00	95.94	0.91	84.61	2.49
	0.01	95.94	0.91	85.21	0.54
	0.10	95.80	0.85	85.14	2.17
LP-squared	-	96.17	1.01	**93.07**	2.18
GNN	-	95.86	0.64	91.86	1.12

Table 1. Accuracy on the artificial datasets, for the proposed model Lagrangian Propagation (LP)-GNN and the original GNN model (Scarselli et al., 2009) for different settings.

4.2. Karate club dataset

A very interesting analysis (Kipf & Welling, 2017) consists in how latent representations of nodes (states) evolve during the learning process. When there is no dependence on node labels (l_v^0), the states are continuous representations of topological features of the nodes in the graph. In order to perform this evaluation, we exploit the Zachary Karate Club dataset (Zachary, 1977). Each node (labeled by one out of four classes via modularity-based clustering) represents one of the 34 members of the club, and each of the 154 edges represents a tie between two members. We trained a layered Lagrangian Propagation GNN (LP-GNN) with three state layers ($K = 3$), having a two–dimensional state in the last layer. A shallow softmax regressor is used as output function, to force a linear separation among classes. Moreover, node-attached features of the given data were totally removed in order to force the algorithm to exploit only structural properties in the solution of the classification task. Figure 1 shows how the node states evolve over time.

4.3. Graph Classification

We considered 4 benchmarks in bioinformatics (MUTAG, PTC, NCI1, PROTEINS) and 2 in social network analysis (IMDB-BINARY, IMDB-MULTI) (Yanardag & Vishwanathan, 2015). In MUTAG, PTC, NCI1, and PROTEINS, the graph nodes have categorical input features (e.g. the atom symbol). In the social network datasets, there are no node features and we followed the approach in Xu et al. (2018), where the nodes were labeled by one-hot encodings of their degrees. Dataset statistics are summarized in Table 2.

We compared the proposed LP-GNN model with other state-of-the-art neural models for graph classification. LP-GNN is proposed both in a shallow version ($K = 1$, LP-GNN-SINGLE) and in a deep version ($K > 1$, LP-GNN-MULTI).

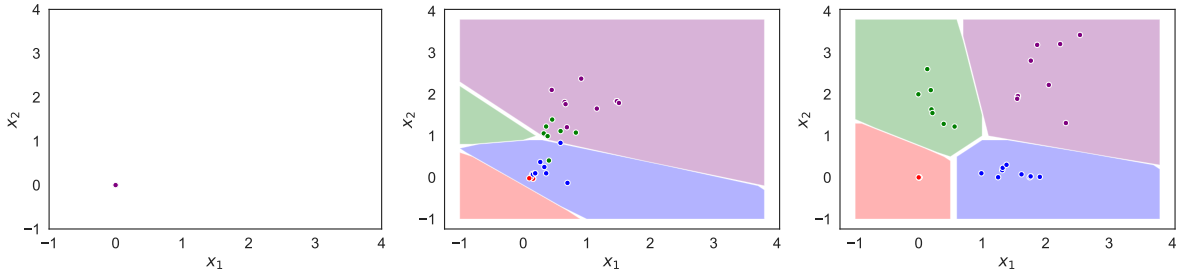

Figure 1. Evolution of the node state embeddings: beginning, after 200 epochs and at convergence. Point colors correspond to the ground truth class, while background colors denote the regions of the predicted class.

Datasets	IMDB-B	IMDB-M	MUTAG	PROT.	PTC	NCI1
# graphs	1000	1500	188	1113	344	4110
# classes	2	3	2	2	2	2
Avg # nodes	19.8	13.0	17.9	39.1	25.5	29.8
DCNN	49.1	33.5	67.0	61.3	56.6	62.6
PATCHYSAN	71.0 ± 2.2	45.2 ± 2.8	**92.6** ± 4.2	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9
DGCNN	70.0	47.8	85.8	75.5	58.6	74.4
AWE	74.5 ± 5.9	51.5 ± 3.6	87.9 ± 9.8	–	–	–
GRAPHSAGE	72.3 ± 5.3	50.9 ± 2.2	85.1 ± 7.6	75.9 ± 3.2	63.9 ± 7.7	77.7 ± 1.5
GIN	75.1 ± 5.1	**52.3** ± 2.8	89.4 ± 5.6	76.2 ± 2.8	64.6 ± 7.0	**82.7** ± 1.7
GNN	60.9 ± 5.7	41.1 ± 3.8	88.8 ± 11.5	76.4 ± 4.4	61.2 ± 8.5	51.5 ± 2.6
LP-GNN-SINGLE	71.2 ± 4.7	46.6 ± 3.7	90.5 ± 7.0	77.1 ± 4.3	64.4 ± 5.9	68.4 ± 2.1
LP-GNN-MULTI	**76.2** ± 3.2	51.1 ± 2.1	**92.2** ± 5.6	**77.5** ± 5.2	**67.9** ± 7.2	74.9 ± 2.4

Table 2. Average and standard deviation of the classification accuracy on the graph classification benchmarks, evaluated on the test set, for different GNN models. The models used in the comparison (first column) are reported in Section 2.

All the GNN-like models have a number of layers/iterations equal to 5 (as in the original papers). For graph-focused tasks the *readout* function exploits an aggregated representation obtained, in the specific implementation, by summing or averaging the top-most layer node states, x_v^{K-1}. Results are shown in Table 2. The deep version of the proposed model (LP-GNN-MULTI) compares similarly or favourably to all the other methods. Despite the fact that LP-GNN-SINGLE is the unique neural model not relying on a deep stack of layers, it already offers performances that, on average, are preferable or on-par to the ones obtained by more complex models that exploit a larger amount of parameters. Hence, it seems that some tasks suffice a shallow representation of the nodes, but still need a diffusion process to take place. LP-GNN can naturally model this diffusion, without the need of deep architectures. On the other side, GCN-like models need to stack multiple layers to achieve this result.

To investigate further this aspect, we compared LP-GNNs with the state-of-the-art GIN (Xu et al., 2018) model on the IMDB-B dataset (since it contains no node features), when increasing the number of state layers. The goal is to show that shallow LP-GNNs (one layer) can still yield good performances, since the diffusion process is independent of the depth of the network. Conversely, we expect the GIN model, as other GCNs, to need deep architectures with a larger number of parameters for the diffusion process to take place. Results are shown in Table 3. For a number of layers

Model	**Number of State Layers**			
	1	2	3	5
GIN (Xu et al., 2018)	52	72.6	72.7	75.1
LP-GNN	**71.2**	**73.7**	**73.9**	**76.2**

Table 3. Average test accuracy on the IMDB-B dataset for LP-GNN and GIN model with state layers $K \in [1, 5]$.

greater than 1, the two models perform similarly, reaching good performances just for $K \geq 2$. However, with only 1 layer, the GIN model exploits information available in 1-hop neighbors, reaching a 52% accuracy (which is close to random in a binary classification task). On the contrary, LP-GNN yields 71.2% of accuracy. This suggests that GCN architectures need a second layer (and thus a larger number of parameters) to perform the diffusion at 2-hop neighbors, rather than for exploiting a higher representational power.

5. Conclusions

We proposed an approach for training GNN models cast as a constrained optimization problem, avoiding the explicit computation of the fixed point needed to encode the graph. The resulting algorithm, Lagrangian Propagation GNN, yields state-of-the-art performances in multiple tasks, showing the tradeoff between the information diffusion process and the depth of the model.

References

Atwood, J. and Towsley, D. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 1993–2001, 2016.

Bianchini, M., Dimitri, G. M., Maggini, M., and Scarselli, F. *Deep Neural Networks for Structured Data*, pp. 29–51. Springer International Publishing, Cham, 2018. ISBN 978-3-319-89629-8. doi: 10.1007/978-3-319-89629-8_2. URL https://doi.org/10.1007/978-3-319-89629-8_2.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6203.

Carreira-Perpinan, M. and Wang, W. Distributed optimization of deeply nested systems. In *Artificial Intelligence and Statistics*, pp. 10–19, 2014.

Dai, H., Kozareva, Z., Dai, B., Smola, A., and Song, L. Learning steady-states of iterative algorithms over graphs. In *International Conference on Machine Learning*, pp. 1114–1122, 2018.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3837–3845, 2016.

Duvenaud, D. K., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 2224–2232, 2015.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *NIPS*, 2017.

Henaff, M., Bruna, J., and LeCun, Y. Deep convolutional networks on graph-structured data. *CoRR*, abs/1506.05163, 2015. URL http://arxiv.org/abs/1506.05163.

Ivanov, S. and Burnaev, E. Anonymous walk embeddings. *arXiv preprint arXiv:1805.11921*, 2018.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

LeCun, Y., Touresky, D., Hinton, G., and Sejnowski, T. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pp. 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. S. Gated graph sequence neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05493.

Marra, G., Tiezzi, M., Melacci, S., Betti, A., Maggini, M., and Gori, M. Local propagation in constraint-based neural network. *arXiv preprint arXiv:2002.07720*, 2020.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 2014–2023, 2016. URL http://jmlr.org/proceedings/papers/v48/niepert16.html.

Platt, J. C. and Barr, A. H. Constrained differential optimization. In *Neural Information Processing Systems*, pp. 612–621, 1988.

Rossi, A., Tiezzi, M., Dimitri, G. M., Bianchini, M., Maggini, M., and Scarselli, F. Inductive–transductive learning with graph neural networks. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pp. 201–212. Springer, 2018.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605. URL https://doi.org/10.1109/TNN.2008.2005605.

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, pp. 2722–2731, 2016.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *CoRR*, abs/1810.00826, 2018. URL http://arxiv.org/abs/1810.00826.

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374. ACM, 2015.

Zachary, W. W. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 4438–4445, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17146.

A. Lagrangian derivatives

For simplicity, we consider the case of a single layer LP-GNN, i.e. $K = 1$, and $\Theta_{f_a} = [\theta_{f_a^0}]$. The derivatives of the Lagrangian with respect to the considered parameters are:

$$\frac{\partial \mathcal{L}}{\partial x_v} = L' f'_{r,v} + \lambda_v \mathcal{G}'_v (1 - f'_{a,v}) - \sum_{w : v \in ne[w]} \lambda_w \mathcal{G}'_w f'_{a,w} \tag{9}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_{f_a}} = - \sum_{v \in S} \lambda_v \mathcal{G}'_v f'_{a,v} \tag{10}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_{f_r}} = \sum_{v \in S} L' f'_{r,v} \tag{11}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_v} = \mathcal{G}_v \tag{12}$$

where $f'_{a,v}$ is the derivative of $f_{a,v}$ computed with respect to the same argument as in the partial derivative on the left side, $f_{r,v} = f_r(x_v | \theta_{f_r})$, $f'_{r,v}$ is its first derivative, $\mathcal{G}_v = \mathcal{G}\left(x_v - f_{a,v}\right)$ and $\mathcal{G}'_v$ is its first derivative, and, finally, L' is the first derivative of L. Note that when parameters are vectors, the reported gradients should be considered element-wise.

B. Experimental settings

B.1. Artificial Tasks

We tuned the hyperparameters on the validation data, by selecting the node state dimension from the set $\{5, 10, 35\}$; the dropout drop-rate from the set $\{0, 0.7\}$; the state *transition function* from $\{f_{a,v}^{(\text{SUM})}, f_{a,v}^{(\text{AVG})}\}$ where:

$$f_{a,v}^{(\text{SUM})} = \sum_{u \in ne[v]} h(x_u, l_u, l_v, l_{(u,v)} \mid \theta_h)$$
$$f_{a,v}^{(\text{AVG})} = \frac{1}{|ne[v]|} \sum_{u \in ne[v]} h(x_u, l_u, l_v, l_{(u,v)} \mid \theta_h),$$

with the function $h()$ computed by a feedforward neural network with s outputs, whose input is the concatenation of its arguments (i.e. the input is a vector of $s + 2m + d$ entries); their number of hidden units in $\{5, 20, 50\}$. We used the Adam optimizer (TensorFlow). The learning rate for the parameters Θ_{f_a} and θ_{f_r} is selected from the set $\{10^{-5}, 10^{-4}, 10^{-3}\}$, and the learning rate for the variables x_v and λ_v from the set $\{10^{-4}, 10^{-3}, 10^{-2}\}$.

Subgraph Matching. Given a graph G and a graph S such that $|S| \leq |G|$, the subgraph matching problem consists in finding the nodes of a subgraph $\hat{S} \subset G$ which is isomorphic to S. The task is that of learning a function τ, such that $\tau_S(G, n) = 1, n \in V$, when the node n belongs to the given subgraph S, otherwise $\tau_S(G, n) = 0$. Our dataset is composed of 100 different graphs, each one having 7 nodes. The number of nodes of the target subgraph S is instead 3.

Clique localization. A clique is a complete graph, i.e. a graph in which each node is connected with all the others. In a network, overlapping cliques (i.e. cliques that share some nodes) are admitted. Clique localization is a particular instance of the subgraph matching problem, with S being complete. In the experiments, we consider a dataset composed by graphs having 7 nodes each, where the dimension of the maximal clique is 3 nodes.

B.2. Graph Classification

The models used in the comparison are: Diffusion-Convolutional Neural Networks (DCNN) (Atwood & Towsley, 2016), PATCHY-SAN (Niepert et al., 2016), Deep Graph CNN (DGCNN) (Zhang et al., 2018), AWE (Ivanov & Burnaev, 2018), GraphSAGE (Hamilton et al., 2017) , GIN-GNN (Xu et al., 2018), original GNN (Scarselli et al., 2009; Rossi et al., 2018). Apart from original GNN, we report the accuracy as reported in the referred papers.

The *readout* function exploits an aggregated representation obtained, in the specific implementation, by summing the top-most layer node states, x_v^{K-1}:

$$y_G = f_r \left(\sum_{v \in V} x_v^{K-1} \mid \theta_{f_r} \right).$$

We tuned the hyperparameters by searching: (1) the number of hidden units for both the f_a and f_r functions from the set $\{5, 20, 50, 70, 150\}$; (2) the state transition function from $\{f_{a,v}^{(\text{SUM})}, f_{a,v}^{(\text{AVG})}\}$; (3) the dropout ratio from $\{0, 0.7\}$; (4) the size of the node state x_v from $\{10, 35, 50, 70, 150\}$; (5) learning rates for both the θ_{f_a}, θ_{f_r}, x_v and λ_v from $\{0.1, 0.01, 0.001\}$.