Graph Learning Network: A Structure Learning Algorithm

Darwin Saire Pilco [1] **Adín Ramírez Rivera** [1]

Abstract

Recently, graph neural networks (GNNs) have proved to be suitable in tasks on unstructured data. Particularly in tasks as community detection, node classification, and link prediction. However, most GNN models still operate with static relationships. We propose the Graph Learning Network (GLN), a simple yet effective process to learn node embeddings and structure prediction functions. Our model uses graph convolutions to propose expected node features, and predict the best structure based on them. We repeat these steps recursively to enhance the prediction and the embeddings.

1. Introduction

When working on unstructured information, commonly, graphs are employed because they can represent this information naturally. For instance, in social networks, system recommendations, and link prediction, graphs can capture the relationship between entities. In order to work on this type of information, deep models on graphs were created (Defferrard et al., 2016; Gori et al., 2005; Kipf & Welling, 2017; Scarselli et al., 2009). These models take into account the information of each node and its neighborhood relationships when extracting new information (i.e., node embedding). Unlike traditional models on graphs, which still work on a static domain (i.e., graphs without variation in the structure), Bresson & Laurent (2018); Li et al. (2016); Marcheggiani & Titov (2017); Ying et al. (2018) began to work on dynamic domains (i.e., variable graphs). However, they still do not support extreme variations; i.e., complete changes in the structure of graphs in each layer.

Related work. We classify the graph representation learning methods into two groups: *generative models* that learn the graph relationship distribution from latent variables, and *discriminative models* that predict the edge probability between pairs of vertices.

For generative models, the Variational Autoencoder (VAE) (Kingma & Welling, 2014; Sohn et al., 2015) proved to be competent at generating graphs. Thus, methods based on VAEs (Bojchevski et al., 2018; De Cao & Kipf, 2018; Grover et al., 2018; Kearnes et al., 2019; Kusner et al., 2017; Simonovsky & Komodakis, 2018) learn some probability distribution that fits and models the graph's relationships. Other methods (Li et al., 2018; You et al., 2018) propose auto-regressive models (i.e., generate node-to-node graphs) to generate graphs with a similar structure. Nevertheless, we consider relevant to contrast ourselves with the generative methods since they aim to learn the structures (regardless of the difference in the final task).

In contrast to the first group, the discriminative models do not use conditional distributions to generate edges but directly aim to learn a classifier for predicting the presence of edges. For this, a diversity of models based on GNNs (Gori et al., 2005; Scarselli et al., 2009) were explored (dgl; Battaglia et al., 2018). For example, methods for recommendation systems on bipartite graphs were proposed by Berg et al. (2018). Schlichtkrull et al. (2018) merged autoencoder and factorization methods (i.e., use of scoring function) to predict labeled edges. Besides, diverse approaches try to take advantage of recurrent neural networks (Monti et al., 2017), and heuristic methods (Donnat et al., 2018; Zhang & Chen, 2018). Different from previous methods, message-passing approaches (Battaglia et al., 2018; Gilmer et al., 2017; Kipf et al., 2018) add edge embedding for each relationship between two nodes. Similarly, we predict the edges of the graph based on an initial set of nodes and a configuration. However, we learn local and global transformations around the nodes, while transforming the features too, in turn, enhance the structure prediction.

In this paper, we predict new structures from the local and global node embedding in the graph through a recurrent set of operations. In each application of our block, we adjust the graph's structure and nodes' features. In other words, we work with variable graphs to predict new structures.

Contributions. (i) Two prediction functions (for nodes' features and adjacency) that lets us extract the most probable

Code available at https://gitlab.com/mipl/graph-learning-network.

[1]Institute of Computing, University of Campinas, Campinas, Brazil. Correspondence to: Darwin Saire Pilco <darwin.pilco@ic.unicamp.br>, Adín Ramírez Rivera <adin@ic.unicamp.br>.

Presented at the ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data

structure given a set of points and their feature embeddings, respectively. (ii) A recurrent architecture that defines our iterative process and our prediction functions. (iii) An end-to-end learning framework for predicting graphs' structure given a family of graphs. (iv) Additionally, we introduce a synthetic dataset, i.e., 3D surface functions, that contains patterns that can be controlled and mapped into graphs to evaluate the robustness of existing methods.

2. Graph Learning Network

Given a set of vertices $V = \{v_i\}$, such that every element v_i is a feature vector, we intend to predict its structure as a set of edges between the vertices, $E = \{(v_i, v_j) : v_i, v_j \in V\}$. In other words, we want to learn the edges of the graph $G = (V, E)$ that maximize the relations between the vertices given some prior patterns, i.e., a family of graphs.

To achieve this, we perform two alternating tasks for a given number of times, akin to an expectation-maximization process. At each step, we transform the nodes' features through convolutions on the graph (Kipf & Welling, 2017) using multi-kernels to learn better representations to predict their structure. Then, we merge the multiple node embedding and apply function transform (Bai et al., 2019) on them that combines the local and global contexts for the embeddings. Next, we use these transformed features (local and global) in a pairwise node method to predict the next structure, which is represented through an adjacency matrix. The learned convolutions on the graph represent a set of responses on the nodes that will reveal their relations. These responses are combined to create or delete connections between the nodes, and encoded into the adjacency matrix. The sequential application of these steps recover effective relations on nodes, even when trained on families of graphs. We represent this process in Fig. 1.

Node Embeddings. At a given step l on the alternating process, we have d_l hidden features, $H^{(l)} \in \mathbb{R}^{n \times d_l}$, for each of our n nodes, and the set of edges (structure) encoded into an adjacency matrix $A^{(l)} \in [0, 1]^{n \times n}$ that represents our graph. As introduced, our first step is to produce the features of the next step, $H^{(l+1)}$, through the embedding

$$H^{(l+1)} = \lambda_l \left(\eta_l \left(H^{(l)}, A^{(l)} \right), A^{(l)} \right). \tag{1}$$

Our embedding comprises to steps: extracting k features for the nodes, and combining them into an intermediary embedding (2); and creating a local representation (5). For the first step, we use convolutional graph operations (Kipf & Welling, 2017)

$$H^{(l)}_{\text{int}} = \eta_l \left(H^{(l)}, A^{(l)} \right) = \sum_{i=1}^{k} \sigma_l \left(\tau \left(A^{(l)} \right) H^{(l)} W_i^{(l)} \right), \tag{2}$$

where k is the number of kernels, $W_i^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ is the learnable weight matrix for the ith convolutional kernel at

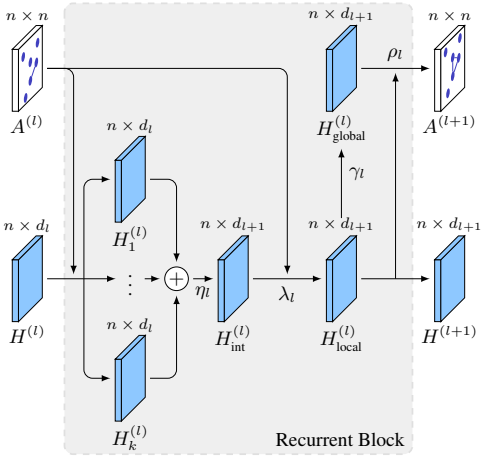

Figure 1. Our proposed method is a recurrent block. We create a set of node embeddings $\{H_i^{(l)}\}_{i=1}^{k}$ that are later combined to produce an intermediary representation $H^{(l)}_{\text{int}}$. Then, we use the updated node information with the adjacency information to produce a local embedding of the nodes information $H^{(l)}_{\text{local}}$ that is also the output $H^{(l+1)}$. We also broadcast the information of the local embedding to produce a global embedding $H^{(l)}_{\text{global}}$. We combine the local and global embeddings to predict the next layer adjacency $A^{(l+1)}$.

the lth step, $\sigma_l(\cdot)$ is a non-linear function, and $\tau(\cdot)$ is a symmetric normalization transformation of the adjacency matrix, defined by

$$\tau \left(A^{(l)} \right) = \left(\hat{D}^{(l)} \right)^{-\frac{1}{2}} \left(A^{(l)} + I_n \right) \left(\hat{D}^{(l)} \right)^{-\frac{1}{2}}, \tag{3}$$

where $\hat{D}^{(l)}$ is the degree matrix of the graph plus identity, that is,

$$\hat{D}^{(l)} = D^{(l)} + I_n, \tag{4}$$

where $D^{(l)}$ is the degree matrix of $A^{(l)}$, and I_n is the identity matrix of size $n \times n$. Unlike previous work (Kipf & Welling, 2017), we are computing convolutions that will have different neighborhoods at each step defined by the changing $A^{(l)}$, in addition to multiple learnable kernels per layer. In summary, this step allows us to learn a response function, defined by the weights $W_i^{(l)}$ of the ith kernel, that embed the node's features into a suitable form to predict the structure of the graph.

The second step corresponds to create a local-context embedding from the intermediary representation (2) that depends on the current adjacency. We define our local context λ_l as

$$H^{(l)}_{\text{local}} = \lambda_l \left(H^{(l)}_{\text{int}}, A^{(l)} \right) = \sigma_l \left(\tau \left(A^{(l)} \right) H^{(l)}_{\text{int}} U^{(l)} \right), \tag{5}$$

where $U^{(l)} \in \mathbb{R}^{d_{l+1} \times d_{l+1}}$ is the learnable weight matrix for the linear combinations of the nodes' features $H^{(l)}_{\text{int}}$.

Adjacency Matrix Prediction. After obtaining the nodes embeddings, $H^{(l)}_{\text{local}}$ (5), we use them to predict the next

adjacency matrix $A^{(l+1)}$ through

$$A^{(l+1)} = \rho_l\left(H^{(l)}_{\text{local}}\right) = \sigma_l\left(M^{(l)}\alpha_l\left(H^{(l)}_{\text{local}}\right)M^{(l)\top}\right), \quad (6)$$

where $M^{(l)} \in \mathbb{R}^{n\times n}$ is the weight matrix that produces a symmetric adjacency, α_l is a transformation that mixes global and local information within the graph, and $\cdot^\top$ denotes the transposition operator.

We broadcast the local information to all the nodes by assuming that all the nodes are connected, i.e., the adjacency on the graph would be $A^{(l)}=\mathbb{1}$, and then using a convolution operation. We define the global context as

$$H^{(l)}_{\text{global}} = \gamma_l\left(H^{(l)}_{\text{local}}\right) = \sigma_l\left(H^{(l)}_{\text{local}}Z^{(l)}\right), \quad (7)$$

where $Z^{(l)} \in \mathbb{R}^{d_{l+1}\times d_{l+1}}$ is the learnable weight matrix. This operation is similar to attention mechanisms previously used (Bai et al., 2019), yet, we use it as a broadcasting mechanism instead.

Finally, we merge both local (5) and global (7) contexts using a transformation function

$$\alpha_l\left(H^{(l)}_{\text{local}}\right) = H^{(l)}_{\text{local}}Q^{(l)}\gamma_l\left(H^{(l)}_{\text{local}}\right)^\top, \quad (8)$$

where $Q^{(l)}\in\mathbb{R}^{d_{l+1}\times d_{l+1}}$ is the learnable weight matrix. The intuition is that nodes similar to the global and local context should receive higher attention weights for the projection of a new adjacency graph within the graph creation (6).

In other words, the ρ_l function broadcasts the information of the nodes' neighborhoods (as determined by the adjacency on the previous step, $A^{(l)}$, and embedded in the local context), and, at each edge, creates a score of the possible adjacency as a linear combination of the nodes' features restricted to the existing structure.

3. Learning Framework

We are assuming that we have a family of undirected graphs, $\mathcal{G}=\{G_i\}_i$, that have a particular structure pattern that we are interested in. We will use each of the graphs, $G_i = (V_i, A_i)$, to learn the parameters, Θ, of our model that minimize the loss function (12) on each of them. The structure of each graph is used as ground truth, $A^*_i = A_i$. The graph is predicted by the set of node embeddings, λ_l (5), and the adjacency prediction, ρ_l (6), functions that depend on the weight matrices (i.e., Θ) that are learnable, defined in Section 2.

Our input comprises the vertices, $H^{(0)} = V_i$, and some structure for training. In our experiments, we used the identity, $A^{(0)} = I$. However, other structures can be used as well. In the following, we describe our learning framework to obtain the parameters $\theta_l \in \Theta$ of our functions for every l. For brevity, we will omit the parameters on the losses and in their functions.

Given the combinations of pairs of vertices on a graph, the total number of pairs with an edge (positive class) is, commonly, fewer than pairs without an edge (negative class). In order to handle the imbalance between the two binary classes (edge, no edge), we used the HED-loss function (Xie & Tu, 2015) that is a class-balanced cross-entropy function. Then we consider the edge-class objective function as

$$\mathcal{L}_c = -\beta\sum_{i\in Y_+}\log P\left(A^o_i\right) - (1-\beta)\sum_{j\in Y_-}\log P\left(A^o_j\right), \quad (9)$$

where A^o_i is the indexed predicted edge (output) for the ith pair of vertices. The proportion of positive (edge) and negative (no edge) pairs of vertices on the A^* graph are $\beta = |Y_+|/|Y|$ and $1-\beta = |Y_-|/|Y|$, where $Y = Y_+ \cup Y_-$. And $P(\cdot)$ is the probability of a pair of vertices to have an edge, predicted at the last layer L, such that

$$P\left(A^o_i\right) = A^{(L)}_i. \quad (10)$$

Individually penalizing the (class) prediction of each edge is not enough to model the structure of the graph. Hence, we compare the whole structure of the predicted graph, A^o, with its ground truth, A^*. By treating the edges on the adjacency matrices as regions on an image, we maximize the intersection-over-union (Milletari et al., 2016) of the structural regions. Then we consider the objective function,

$$\mathcal{L}_s = 1 - \frac{2|A^o\cap A^*|}{|A^o|^2+|A^*|^2} = 1 - \frac{2\sum_{i,j}A^o_{i,j}A^*_{i,j}}{\sum_{i,j}(A^o_{i,j})^2+\sum_{i,j}(A^*_{i,j})^2}. \quad (11)$$

Finally, we aim to minimize the total loss that is the sum of all of the previous ones, defined by

$$\mathcal{L} = \psi_1\mathcal{L}_c + \psi_2\mathcal{L}_s, \quad (12)$$

where ψ_1 and ψ_2 are hyper-parameters that define the contribution of each loss to the learning process.

4. Results and Discussion

In this work, we evaluate our model as an edge classifier, and simulate its performance as a graph generator by inputting noise as features and predicting on them. We perform experiments on three synthetic datasets that consist of images with Geometric Figures for segmentation, 3D surface function, and Community dataset (see Appendices A.1, A.2, and A.3, respectively). For our experiments, we used 80% of the graphs in each dataset for training, and test on the rest. Our evaluation metric is the Maximum Mean Discrepancy (MMD) measure (You et al., 2018), which measures the Wasserstein distance over three statistics of the graphs: degree (Deg), clustering coefficients (Clus), and orbits (Orb).

We report our results contrasted against existing methods on Table 1. Additionally, we show more experiments using accuracy (Acc), intersection-over-union (IoU), and dice coefficient (Dice) in Appendix C.

Table 1. Comparison of GLN against deep generative models, GraphRNN (G.RNN), Kronecker (Kron.), and MMSB, on the Community ($C = 2$ and $C = 4$), on all sequences of Surf100 and Surf400, and Geometric Figures datasets. The evaluation metric is MMD for degree (D), cluster (C), and orbits (O) shown row-wise per method, where smaller numbers denote better performance.

		C2	C4	Surf400							Surf100							Geo
				T	EP	S	E	EH	O	A	T	EP	S	E	EH	O	A	
GLN	*D*	0.0121	0.0022	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0016	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0005	0.0062
	C	0.0098	0.0026	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0006	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0003	0.0002
	O	0.6248	0.9952	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0005	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0002	0.0053
G.RNN	*D*	0.0027	0.2843	0.0287	0.0232	0.0303	0.0286	0.0436	0.0155	0.0388	0.0478	0.0506	0.1845	0.0664	0.0321	0.0880	0.0628	0.0023
	C	0.0052	0.2272	1.6302	1.6690	1.7358	1.8362	1.8313	1.8057	1.7734	1.8271	1.0961	1.5689	1.7155	1.8379	1.9252	1.8962	0.0001
	O	0.0033	1.9987	1.3684	1.3304	1.7337	1.5440	1.6709	1.5646	1.4736	0.4124	0.3705	0.8566	0.7786	0.9005	0.5702	1.5494	0.0015
Kron.	*D*	1.0295	1.3741	0.9231	0.8922	0.9301	0.8873	0.8890	0.8987	0.9028	0.7361	0.8012	0.7279	0.7453	0.6382	0.8655	0.8515	0.5817
	C	1.2837	1.3962	1.7836	1.7955	1.8163	1.8791	1.8814	1.8123	1.8945	1.9098	1.7722	1.7869	1.8981	1.9020	1.9297	1.9063	0.3815
	O	1.1846	1.3283	1.5621	1.5875	1.7834	1.6223	1.7027	1.6928	1.6338	0.4299	0.6013	0.5674	0.5655	0.6731	0.5827	1.3719	0.5052
MMSB	*D*	1.7610	1.7457	1.1160	1.0256	1.1054	1.0513	1.0628	1.0589	1.0435	1.0124	1.0122	0.9940	1.0583	0.9334	1.1648	0.9825	0.6163
	C	1.8817	1.9876	1.9987	1.9916	1.9985	1.9959	1.9975	1.9969	1.9951	1.9526	1.9417	1.9642	1.9744	1.9489	1.9332	1.9369	0.2855
	O	1.4524	1.5095	1.7501	1.7851	1.8254	1.7863	1.7606	1.7480	1.7286	0.4303	0.7118	0.2466	0.6605	0.1209	0.7368	1.1789	0.6066

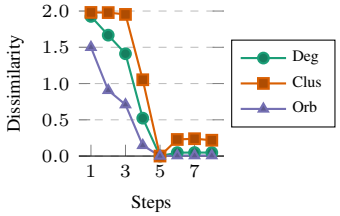

Figure 2. Results of the dissimilarity (MMD) between the prediction and ground truth (smaller values are better) while varying the number of recurrent steps, on the 3D Surface dataset (Surf400).

Table 2. Ablation of GLN using Geometric Figures. Note, in the first three metrics, high values are better, and opposite in the rest.

Losses			Metrics					
IoU	HED	Reg	Acc↑	IoU↑	Dice↑	Deg↓	Clus↓	Orb↓
–	✓	–	0.999 69	0.974 66	0.987 17	0.006 77	0.001 11	0.106 86
–	✓	✓	0.999 70	0.974 89	0.987 25	0.006 48	0.001 02	0.097 24
✓	–	–	0.799 68	0.052 40	0.099 59	1.862 43	1.998 03	0.982 72
✓	–	✓	0.893 78	0.095 27	0.173 96	1.768 95	1.949 12	1.186 16
✓	✓	–	0.999 70	**0.974 90**	**0.987 23**	0.006 27	0.000 19	0.061 87
✓	✓	✓	**0.999 70**	0.974 89	0.987 25	**0.006 22**	**0.000 19**	**0.005 32**

Knowing the depth of the recursive model (i.e., the number of iterations) is not a trivial task since we must find a trade-off between the efficiency and effectiveness of the model. In Fig. 2, we show the dissimilarity metrics (MMD) while varying the number of applications of our proposed block on the 3D Surface dataset. According to our experiment, using five recurrent steps provides the right trade-off.

Additionally, in Table 2, we present an ablation analysis of our model's loss functions and regularization components on the Geometric Figures dataset. We emphasize a stable training and a fast convergence when we minimize both loss functions simultaneously.

Finally, we examined the robustness to structural inputs by randomly changing the proportion of the initial connections

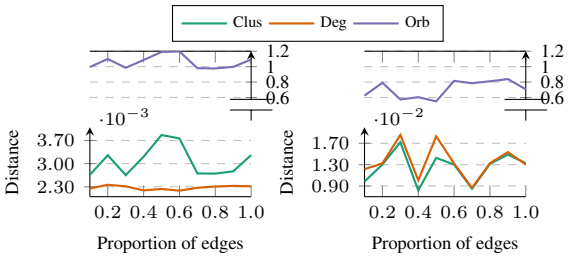

Figure 3. MMD metrics on GLN when varying the input structure on Community $C = 4$ (left) and $C = 2$ (right). The input corresponds to an adjacency matrix with different proportions of connections.

(i.e., 10%, 20%, . . . , 100%) in our input $A^{(0)}$. Fig. 3 shows the average results (of five executions) of this experiment on the Community ($C = 2$, and $C = 4$). We obtained minimum variation on the prediction capabilities of the network. Hence, the best option is to select a minimal graph as input, i.e., the identity matrix. We present our models' qualitative results on the different databases in Appendices D, E, and F.

5. Conclusions

We proposed a simple yet effective method to predict the structure of a set of vertices. Our method works by learning node embedding and adjacency prediction functions and chaining them. This process produces expected embeddings which are used to obtain the most probable adjacency. We encode this process into the neural network architecture. Our experiments demonstrate the prediction capabilities of our model on three databases with structures with different features (the communities are densely connected on some parts, and sparse on others, while the images are connected with at most four neighbors). Further experiments are necessary to evaluate the robustness of the proposed method on larger graphs, with more features and more challenging structures.

Acknowledgements

This work was financed in part by the São Paulo Research Foundation (FAPESP) under grants No. 2016/19947-6 and No. 2017/16597-7, the Brazilian National Council for Scientific and Technological Development (CNPq) under grant No. 307425/2017-7, and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior—Brasil (CAPES)—Finance Code 001. We acknowledge the support of NVIDIA Corporation for the donation of a Titan X Pascal GPU used in this research.

References

Deep graph library. URL https://www.dgl.ai/.

Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., and Wang, W. SimGNN: A neural network approach to fast graph similarity computation. In *ACM Inter. Conf. Web Search Data Min. (WSDM)*, WSDM '19, pp. 384–392, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-5940-5.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv*, (1806.01261v3), 2018.

Berg, R. v. d., Kipf, T. N., and Welling, M. Graph convolutional matrix completion. *ACM Conf. Knowl. Discov. Data Min. (ACM SIGKDD)*, 2018.

Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. NetGAN: Generating graphs via random walks. In *Inter. Conf. Mach. Learn. (ICML)*, 2018.

Bresson, X. and Laurent, T. Residual gated graph convnets, 2018. URL https://openreview.net/forum?id=HyXBcYg0b.

De Cao, N. and Kipf, T. MolGAN: An implicit generative model for small molecular graphs. *arXiv*, (1805.11973), 2018.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Adv. Neural Inf. Process. Sys. (NeurIPS)*, pp. 3844–3852, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9.

Donnat, C., Zitnik, M., Hallac, D., and Leskovec, J. Learning structural node embeddings via diffusion wavelets. In *ACM Conf. Knowl. Discov. Data Min. (ACM SIGKDD)*, pp. 1320–1329. ACM, 2018.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In Precup, D. and Teh, Y. W. (eds.), *Inter. Conf. Mach. Learn. (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *IEEE Inter. Joint Conf. Neural Netw. (IJCNN)*, volume 2, pp. 729–734. IEEE, 2005.

Grover, A., Zweig, A., and Ermon, S. Graphite: Iterative generative modeling of graphs. *arXiv*, (1803.10459v3), 2018.

Kearnes, S., Li, L., and Riley, P. Decoding molecular graph embeddings with reinforcement learning. *arXiv*, (1904.08915), 2019.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *Inter. Conf. Learn. Represent. (ICLR)*, 1050:1, 2014.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. *arXiv*, (1802.04687v2), 2018.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *Inter. Conf. Learn. Represent. (ICLR)*, 2017.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. Grammar variational autoencoder. In *Inter. Conf. Mach. Learn. (ICML)*, pp. 1945–1954, 2017.

Li, Y., Zemel, R., and Brockschmidt, M. a. Gated graph sequence neural networks. In *Inter. Conf. Learn. Represent. (ICLR)*, April 2016.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs. *Inter. Conf. Learn. Represent. (ICLR)*, 2018.

Marcheggiani, D. and Titov, I. Encoding sentences with graph convolutional networks for semantic role labeling. In *Conf. Empir. Methods Nat. Lang. Process. (EMNLP)*, pp. 1506–1515, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

Milletari, F., Navab, N., and Ahmadi, S.-A. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *IEEE Inter. Conf. 3D Vis. (3DV)*, pp. 565–571. IEEE, 2016.

Monti, F., Bronstein, M., and Bresson, X. Geometric matrix completion with recurrent multi-graph neural networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Adv. Neural Inf. Process. Sys. (NeurIPS)*, pp. 3697–3707. Curran Associates, Inc., 2017.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. Computational capabilities of graph neural networks. *IEEE Trans. Neural Netw.*, 20(1):81–102, 2009.

Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In Gangemi, A., Navigli, R., Vidal, M.-E., Hitzler, P., Troncy, R., Hollink, L., Tordai, A., and Alam, M. (eds.), *Semantic Web Conf. (ESWC)*, pp. 593–607, Cham, 2018. Springer International Publishing.

Simonovsky, M. and Komodakis, N. GraphVAE: Towards generation of small graphs using variational autoencoders. In *Int. Conf. Artif. Neural Netw. (ICANN)*, pp. 412–422. Springer, 2018.

Sohn, K., Lee, H., and Yan, X. Learning structured output representation using deep conditional generative models. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Adv. Neural Inf. Process. Sys. (NeurIPS)*, pp. 3483–3491. Curran Associates, Inc., 2015.

Watts, D. J. Networks, dynamics, and the small-world phenomenon. *Amer. J. Soc.*, 105(2):493–527, 1999.

Xie, S. and Tu, Z. Holistically-nested edge detection. In *IEEE Inter. Conf. Comput. Vis. (ICCV)*, pp. 1395–1403, 2015.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *Adv. Neural Inf. Process. Sys. (NeurIPS)*, pp. 4800–4810, 2018.

You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Inter. Conf. Mach. Learn. (ICML)*, pp. 5694–5703, 2018.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. In *Adv. Neural Inf. Process. Sys. (NeurIPS)*, 2018.

Graph Learning Network: A Structure Learning Algorithm

SUPPLEMENTARY MATERIAL

A. Datasets

A.1. Geometric Figures Dataset

We made the Geometric Figures dataset for the task of image segmentation within a controlled environment. Segmentation is given by the connected components of the graph ground-truth. Here, we provide RGB images and their expected segmentations.

The Geometric Figures dataset contains 3000 images of size $n \times n$, that are generated procedurally.[1] Each image contains circles, rectangles, and lines (dividing the image into two parts). We also add white noise to the color intensity of the images to perturb and mixed their regions.

The geometrical figures are of different dimensions, within $[1, n]$, and positioned randomly on the image (taking care in maintaining the geometric figure). There is no specific color for each geometric shape and their background.

For our experiments we use a version of dimension $n = 20$.

A.2. 3D Surfaces Dataset

To evaluate our method we needed a highly structured dataset with intricate relations and with easily understandable features. Hence, we convert parts of 3D surfaces into a mesh by sampling them. Each point in the mesh is translated into a node of the graph, with its position as a feature vector. We have a generator[2] that creates different configurations for this dataset based on a number of nodes per surface, and transformation on it.

We considered the following surfaces:

- **Ellipsoid:** defined by the 3D-function $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$, where the semi-axes are of lengths a, b, and c.
- **Elliptic hyperboloid:** defined by the 3D-function $\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1$, where the semi-axes are of lengths a, b, and c.
- **Elliptic paraboloid:** defined by the 3D-function $\frac{x^2}{a^2} + \frac{y^2}{b^2} = z$, where a and b are the level of curvature in the xz and yz planes respectively.
- **Saddle:** defined by the 3D-function $\frac{x^2}{a^2} - \frac{y^2}{b^2} = z$, where a and b are the level of curvature in the xz and yz planes respectively.
- **Torus:** defined by the 3D-function $\left(\sqrt{x^2 + y^2} - R\right)^2 + z^2 = r^2$, where R is the major radius and r is the minor radius.
- **Another:** defined by the 3D-function $h \sin(\sqrt{x^2 + y^2}) = z$, where h is the height above z-axis.

We generated 200 versions of each surface by randomly applying a set of transformations (from scaling, translation, rotation, reflection, or shearing) to the curve, moreover, two versions of the Surface dataset were created, *Surf100* and *Surf400* that use 100 and 400 vertices per surface, respectively.

A.3. Community Dataset

We perform experiments on a synthetic dataset (Community dataset) that comprises two sets with $C = 2$ and $C = 4$ communities with 40 and 80 vertices each, respectively, created with the caveman algorithm (Watts, 1999), where each community has 20 people. Besides, Community $C = 4$ and $C = 2$ have 500 and 300 samples respectively.

B. Architecture

For our experiments, we used 80% of the graphs in each dataset for training, and test on the rest. For both models, we use the following settings. Our activation functions, σ_l, are sigmoid for all layers, except for the Eq. 7 where σ_l is a hyperbolic tangent. We use $L = 5$ layers to extract the final adjacency and embeddings. The feature dimension, d_l, is 32 for all layers. The learning rate is set 10^{-5} for the Community dataset, and in the rest of datasets, the learning rate is set 5×10^{-6}. Additionally, the number of epochs changes depending on the experiment. Thus in the experiments of Communities, Surfaces and Geometrical Figures we use 150, 200 and 150 times respectively and, the number

[1] Code available at https://gitlab.com/mipl/graph-learning-network.

[2] Code available at https://gitlab.com/mipl/graph-learning-network.

of kernel using is $k = 3$. To convert the prediction of the adjacency into a binary edge, we use a fixed threshold of $\epsilon = 0.5$. The hyper-parameters in our loss function (12) are $\psi_1 = 1$ and $\psi_2 = 1$. In our experiments, we did not needed the regularization our GLN model. Finally, for training, we used the ADAM optimization algorithm on Nvidia GTX Titan X GPU with 12 GB of memory.

C. More Measure of Prediction

Unlike Table 1, where dissimilarity measures are used, such as our metric evaluation on graphs, in Table C.1 we present similarity measures such as accuracy (Acc), intersection-over-union (IoU), Recall (Rec), and Precision (Prec).

D. Prediction of 3D Surface

In Fig. D.1, we show the qualitative result of GLN for the 3D Surface dataset. We show the prediction on the elliptic hyperboloid, elliptic paraboloid, torus, saddle, and ellipsoid, all using 100 nodes (Surf100). We normalized the graphs (w.r.t. scale and translation) for better visualization. Besides, the red edges represent false negatives (i.e., not predicted edges) and black edges are correctly predicted ones.

E. Prediction of Community

In Fig. E.1, we predict the adjacency matrix the of Community dataset on two and four communities, $C = 2$ and $C = 4$ respectively (even rows). Note, our node embedding obtained after apply the λ_l function, shows a good grouping of individuals in the hyperspace (odd rows). Furthermore, the red edges represent false negatives (i.e., not predicted edges), and black edges are correctly predicted ones.

F. Prediction of Geometric Image

Finally, in Fig. F.1, we present an application, even fundamental, on segmentation where each of the connected components represents different objects. For this, we apply our GLN model on Geometric Image dataset, using size image of 20×20. Besides, the white edges represent correct predictions, and light blue dashed edges are false negatives (i.e., not predicted edges).

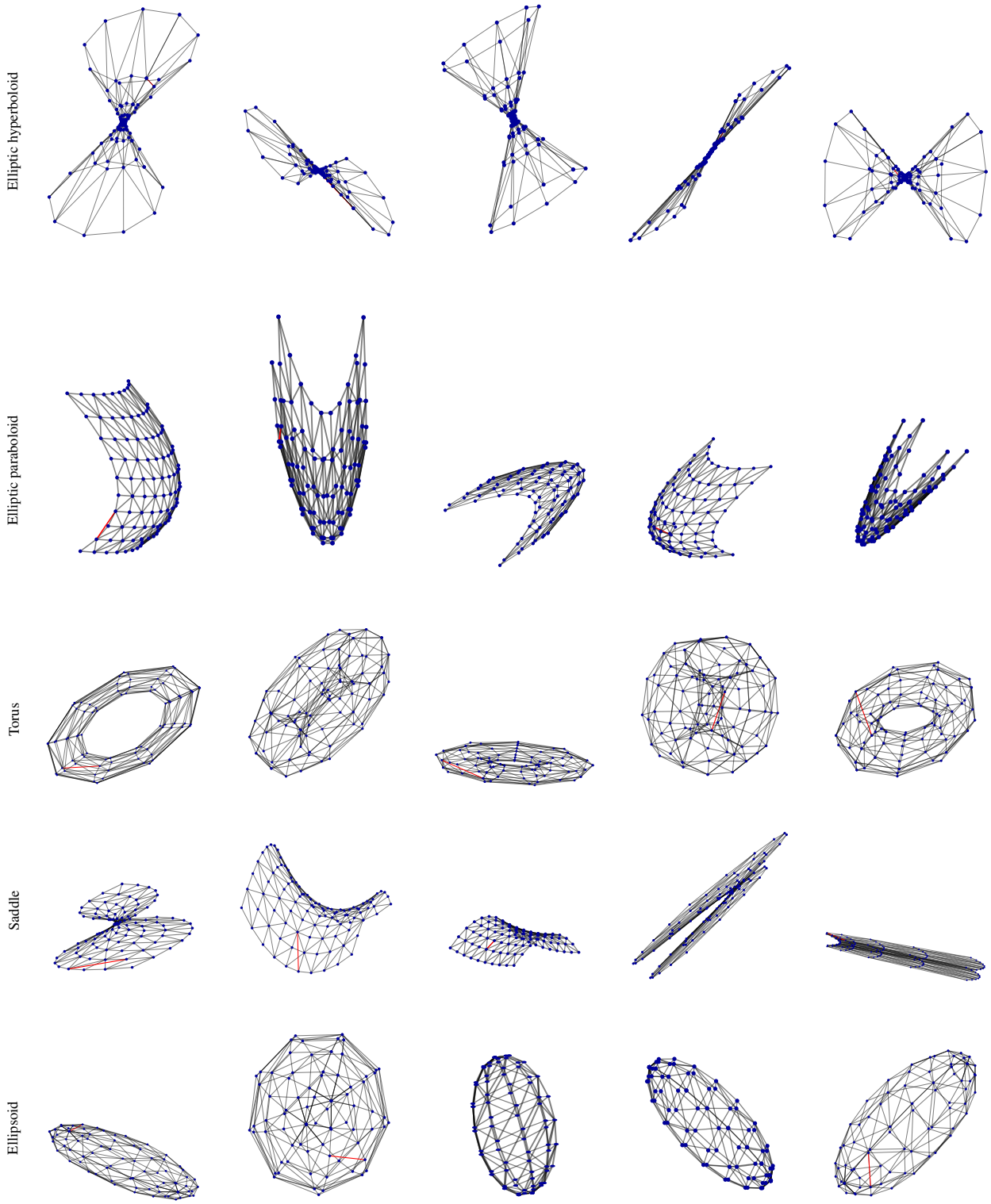

Figure D.1. Results on 3D Surface dataset predictions for the proposed methods, and the learned latent space, used for build adjacency matrix in the prediction. The blue edges represent false negatives (i.e., not predicted edges), red edges represent false positives (i.e., additional predicted edges), and black edges are correctly predicted ones. The graphs were normalized (w.r.t. scale and translation) for better visualization.

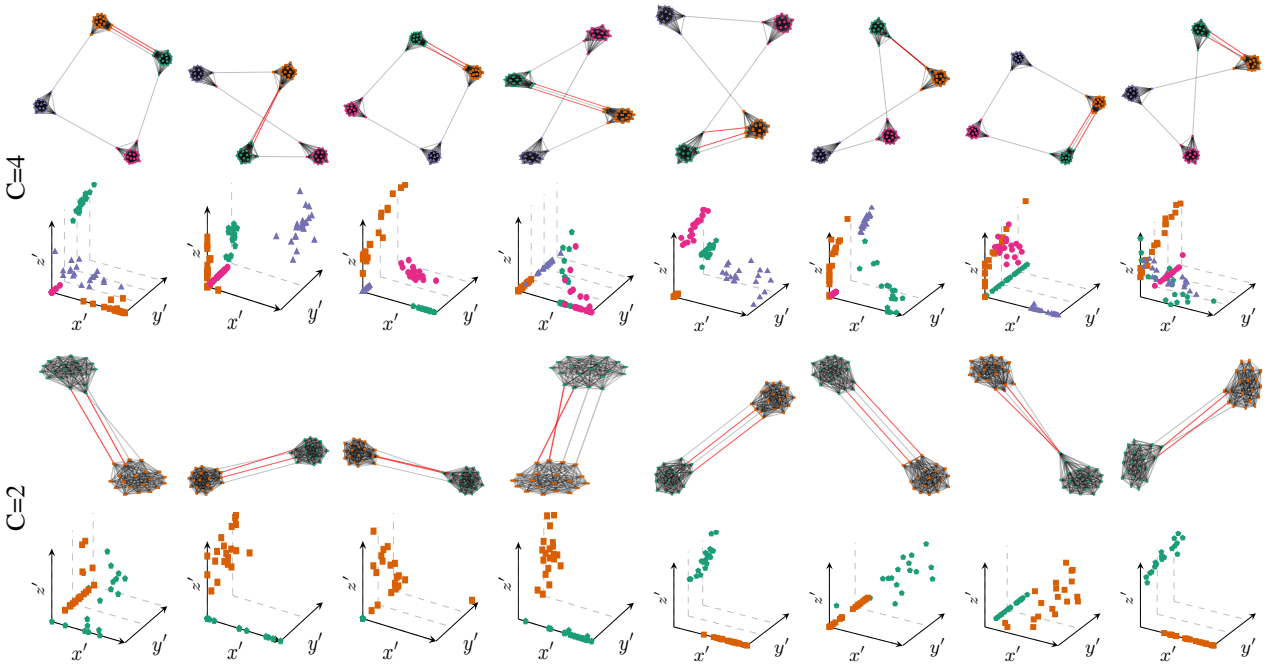

Figure E.1. Results on Community dataset predictions for the proposed methods, and the learned latent space, used for build adjacency matrix in the prediction. The blue edges represent false negatives (i.e., not predicted edges), red edges represent false positives (i.e., additional predicted edges), and black edges are correctly predicted ones. The graphs were normalized (w.r.t. scale and translation) for better visualization.

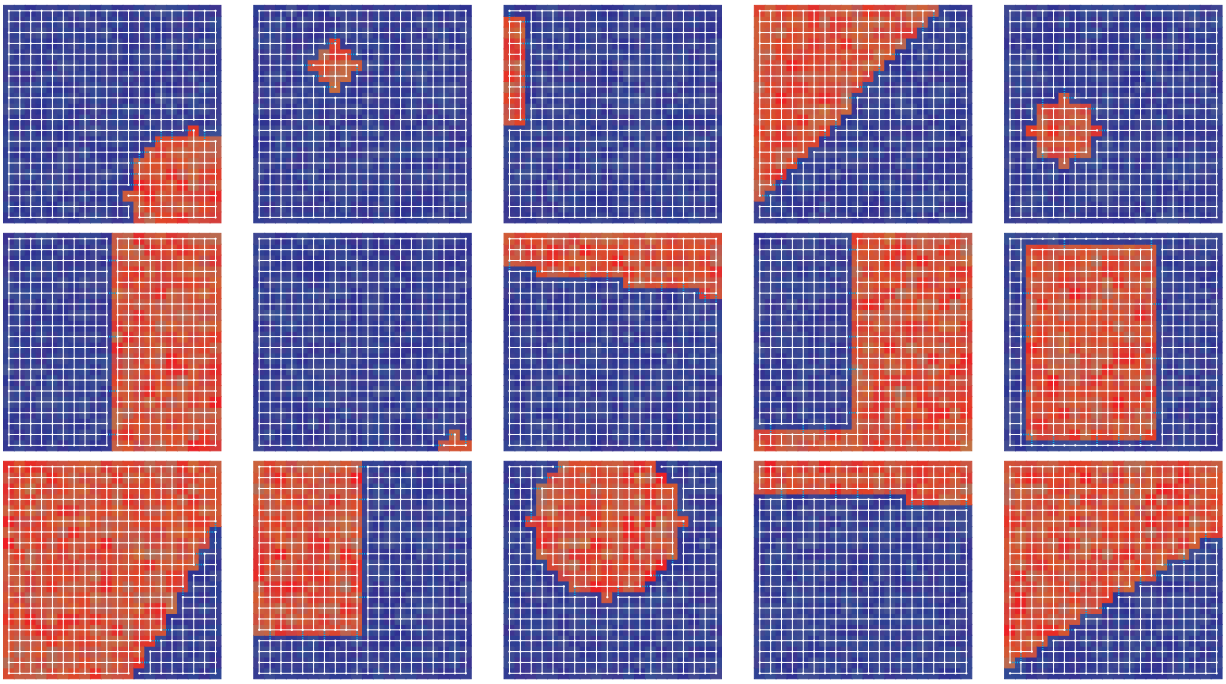

Figure F.1. Predicted graphs using GLN on images with geometric shape of 20×20 pixels. The image behind the graph corresponds to the input values at each node (RGB values), the white edges represent correct predictions, yellow dashed edges are false negatives (i.e., not predicted edges), and light blue dashed edges are false positives (i.e., additional predicted edges).

Table C.1. Comparison of GLN, on the Community ($C = 2$ and $C = 4$), on all sequences of Surf100 and Surf400, and Geometric Figures datasets. The evaluation metric are accuracy (Acc), intersection-over-union (IoU), Recall (Rec), and Precision (Prec) shown row-wise per method, where larger numbers denote better performance.

		C2	C4	Surf400							Surf100							Geo
				T	EP	S	E	EH	O	A	T	EP	S	E	EH	O	A	
GLN	*Acc*	0.997	0.997	0.999	0.999	0.999	0.999	0.999	0.999	0.997	0.999	0.999	0.999	0.999	0.999	0.999	0.993	0.999
	IoU	0.993	0.992	0.991	0.982	0.999	0.981	0.989	0.999	0.865	0.999	0.999	0.999	0.999	0.999	0.999	0.877	0.974
	Rec	0.994	0.997	0.999	0.999	0.999	0.999	0.999	0.999	0.928	0.999	0.999	0.999	0.999	0.999	0.999	0.934	0.986
	Prec	0.997	0.997	0.991	0.982	0.999	0.981	0.989	0.999	0.927	0.999	0.999	0.999	0.999	0.999	0.999	0.934	0.976