Graph Structure Learning

Based on 'Differentiable Graph Module (DGM) for Graph Convolutional Networks' Paper

Yavar Taheri Yeganeh

October 2020

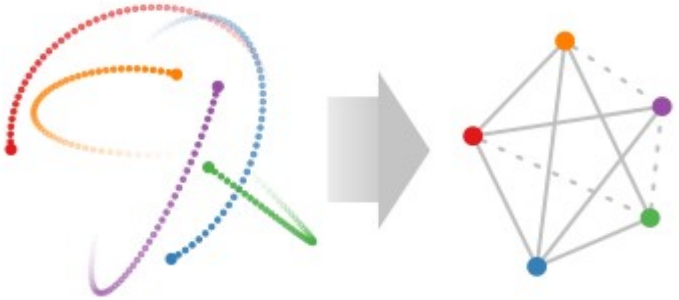
Introduction

- **Geometric deep learning (GDL)** is a novel emerging branch of deep learning attempting to generalize deep neural networks to non-Euclidean structured data such as graphs and manifolds (Bronstein et al., 2017; Hamilton et al., 2017b; Battaglia et al., 2018)
- Graphs: general abstract descriptions of relation and interaction systems
- GNNs can have have enormous applications in a wide variety of domains, e.g., applied in social networks, link prediction, human- object interaction, computer vision and graphics, physics, chemistry, health care and medicine , drug repositioning , protein science, IOT, ...
- This paper is devoted to the following applications: healthcare (disease prediction), brain imaging (gender and age prediction), computer graphics (3D point cloud segmentation), and computer vision (zero-shot learning)
- Please Note: References are not listed in the slides

Introduction

- The most GNN models assume that the **underlying graph is given and fixed** and the graph convolution-like operations typically amount to modifying the node-wise features.
- **Message passing neural networks** (Gilmeret al.) or **primal-dual convolutions** (Monti et al., 2018) allow also to **update the edge features**, but the graph **topology is always kept the same.** This often happens to be a limiting assumption.
- In many application an underlying graph can be expected or example, in medical and healthcare applications, where the graph may be **noisy, partially or even completely unknown**.
- One of the main objectives can be inferring the graph from data.

Introduction



. Physical simulation of 2D particles coupled by invisible springs (*left*) according to a latent interaction graph (*right*). In this example, solid lines between two particle nodes denote connections via springs whereas dashed lines denote the absence of a coupling. In general, multiple, directed edge types – each with a different associated relation – are possible.

Neural Relational Inference for Interacting Systems

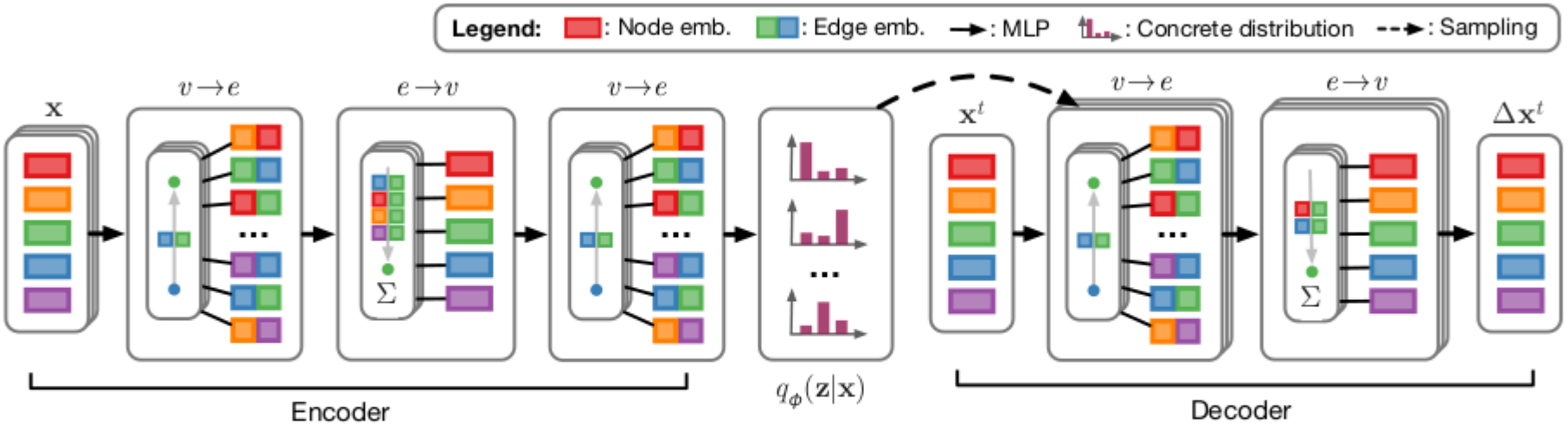
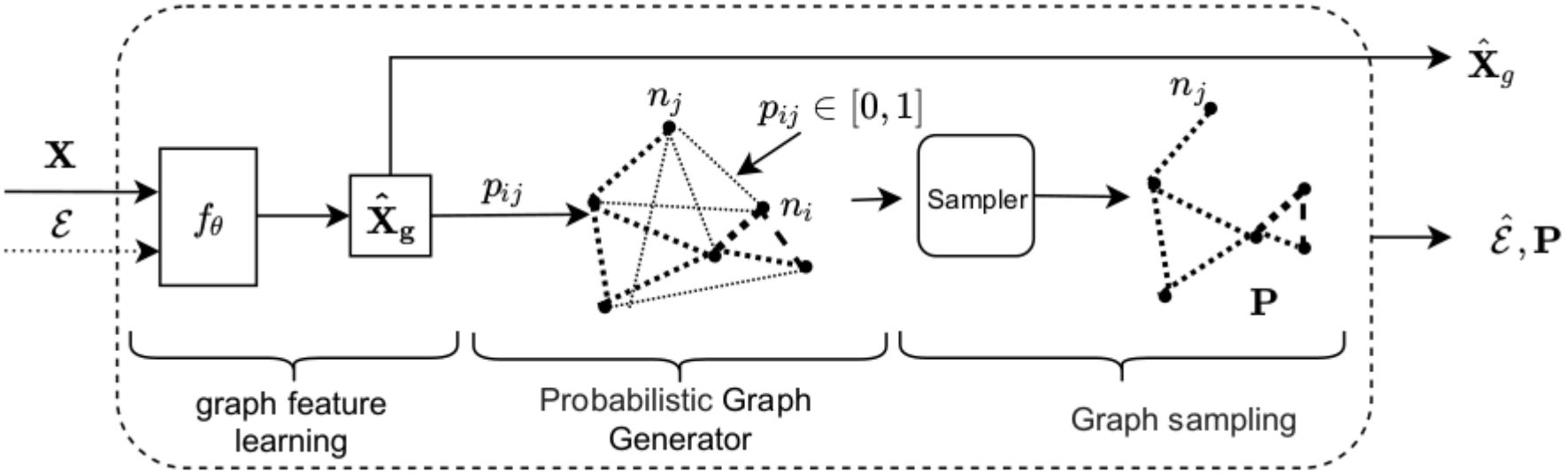


Figure 3. The NRI model consists of two jointly trained parts: An encoder that predicts a probability distribution $q_\phi(\mathbf{z}|\mathbf{x})$ over the latent interactions given input trajectories; and a decoder that generates trajectory predictions conditioned on both the latent code of the encoder and the previous time step of the trajectory. The encoder takes the form of a GNN with multiple rounds of node-to-edge ($v \to e$) and edge-to-node ($e \to v$) message passing, whereas the decoder runs multiple GNNs in parallel, one for each edge type supplied by the latent code of the encoder $q_\phi(\mathbf{z}|\mathbf{x})$.

Introduction



Introduction

- We omit further literature review.
- The idea proposes a generalized technique for **learning the graph based on the output features of each layer and optimize these graphs along with the network parameters during the training**.
- The main obstacle for including the graph construction as a part of the deep learning pipeline that, **being a discrete structure,it is non-differentiable**.
  → Backpropagation through the graph Structure
- The main idea is to use the **continuous deterministic relaxation of neighborhood selection** rules such as kNN (k-nearest neighbors), thus allowing differentiating the output w.r.t. the edges of the graph.
- Approach: Statistical (Probabilistic) approach for edges

Method

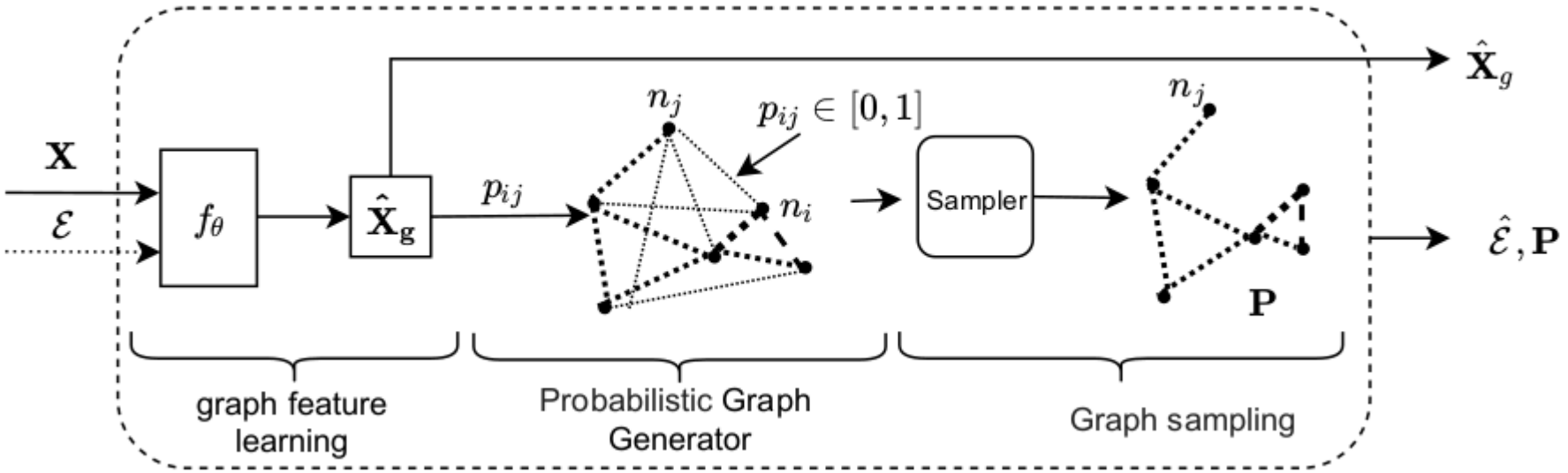
- The goal is to **discover the underlying latent graph structure** in order to enable the use of graph convolutional operators for learning classification tasks.
- Discrete structure of a **graph (G) makes it non-differentiable with respect to its edge set E**, therefore it cannot be directly optimized with gradient-descent based optimization techniques

$\mathbf{X} \in \mathbb{R}^{N \times d}$ $\mathcal{G} = (\{1, \ldots, N\}, \mathcal{E})$ where an edge $(i, j) \in \mathcal{E}$

- To overcome this limitation, we replace the edge set $\mathcal{E}$ with weighted adjacency $\mathbf{P}$, where $p_{ij} \in (0, 1]$ is interpreted as the probability of $(i, j) \in \mathcal{E}$. The probability p_{ij} is computed in a separate feature space $\hat{\mathbf{X}}_g = f_\Theta(\mathbf{X}_g)$ designated as the *graph representation*, where f_Θ is a learnable function. A graph constructed this way can than be sampled according to p_{ij} to be used in any graph convolutional layer for node representation learning.

Method

- **Graph representation feature learning.** The learnable part of our DGM conists of a parametric function $\hat{\mathbf{X}}_g = f_\theta(\mathbf{X}_g)$ transforming input features $\mathbf{X}_g$ into features $\hat{\mathbf{X}}_g$

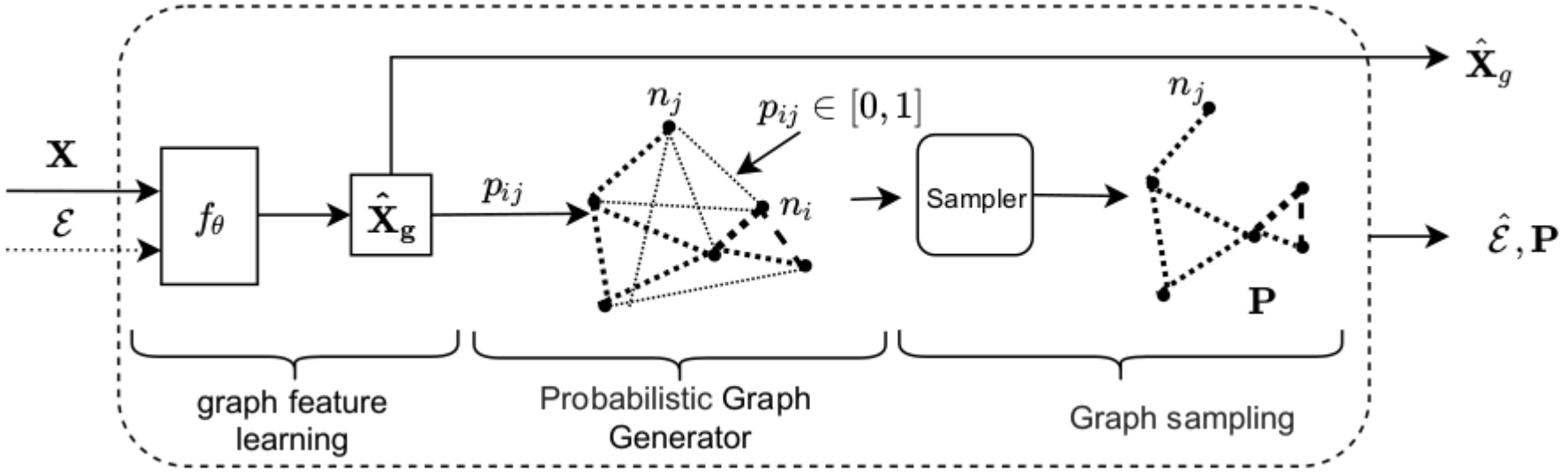


Method

- **Probabilistic graph generator.** The probabilistic graph generator part (shown in fig 1) assumes initially a fully connected graph and computes the probability

$$p_{ij} = e^{-t\|\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j\|^2} \quad (1)$$

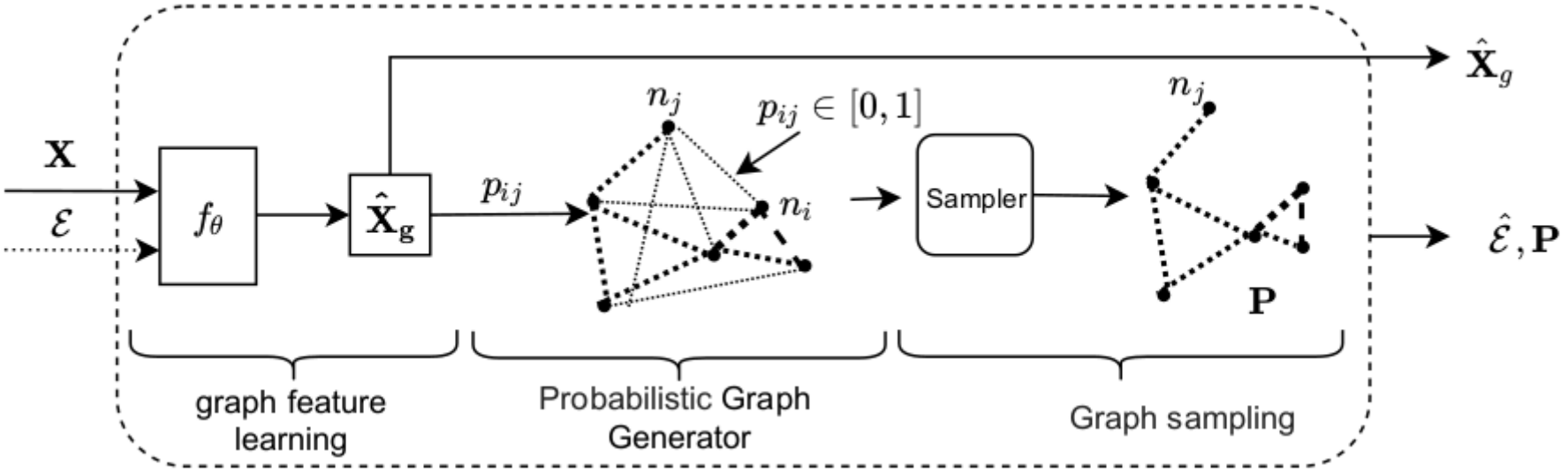
of the edge $(i, j) \in \mathcal{E}$. Here t is a optimized temperature parameter and $\hat{\mathbf{x}}_i \in \hat{\mathbf{X}}_g$ is the output of f_θ. Such a continuous modeling of $\mathcal{E}$ allows back propagation of the gradients through the neighborhood selection.



Method

- **Graph sampling** From the estimated edge probability matrix $\mathbf{P}$, we then sample a fixed k-degree graph. We make use of the *Gumbel-Top-k trick* (Kool et al., 2019) for sampling the unnormalized probability distribution defined in equation 1, thus making the sampling a stochastic relaxation of the k-NN rule.

  Let $\mathbf{p}_i = (p_{ij} : j = 1 \dots N)$ be the unnormalized probability distribution of ingoing edges of i^{th} node. We extract k edges according to the first k elements of $\text{argsort}(\log(\mathbf{p}_i) - \log(-\log(\mathbf{q}))$ where $\mathbf{q} \in \mathbb{R}^N$ is uniform i.i.d. in the interval $[0, 1]$. (Kool et al., 2019) prove that the samples extracted this way follow the categorical distribution



Method

- Forward Pass:

$$\hat{\mathbf{X}}_g^{(l+1)} = f_\theta(\mathcal{E}^{(l)}, \mathbf{X}_g^{(l)})$$
$$\mathcal{E}^{(l+1)} = \hat{\mathcal{E}}^{(l)} \sim \mathbf{P}^{(l)}(\hat{\mathbf{X}}_g^{(l+1)})$$
$$\mathbf{X}^{(l+1)} = g_\phi(\mathcal{E}^{(l+1)}, \mathbf{X}^{(l)}),$$

where ϕ and θ are learned parameters of some non-linear functions f and g. $\mathbf{X}_g^{(l)}$ are features given as input to DGM for graph representation.

In its simplest implementation $\mathbf{X}_g^{(l)} = \mathbf{X}^{(l)}$, meaning that we use same input features for both graph and node representation. We instead propose to use the concatenation of previous graph and node representation features for $l > 1$:

$$\mathbf{X}_g^{(l)} = [\hat{\mathbf{X}}_g^{(l)} | \mathbf{X}^{(l)}]. \quad (2)$$

Method

The final node features $\mathbf{X}^{(L)}$ of last layer L are then used to generate the predictions. Whether not specified we use 'EdgeConv' proposed by (Wang et al., 2019) for both f and g, since it is the natural choice for our fixed k-degree graph sampling:

$$\hat{\mathbf{x}}_i = \square_{j:(i,j)\in\hat{\mathcal{E}}} h_\psi\left(\mathbf{x}_i, \mathbf{x}_j\right) \tag{3}$$

where $\square$ is the permutation-invariant aggregation operation (chosen as $\sum$ in our paper) and h_ψ is a non-linear function with a set of ψ as the learnable parameters. In the first DGM layer, where no graph is available, we just set f as the identity function, letting the network learn only the temperature parameter.

Method

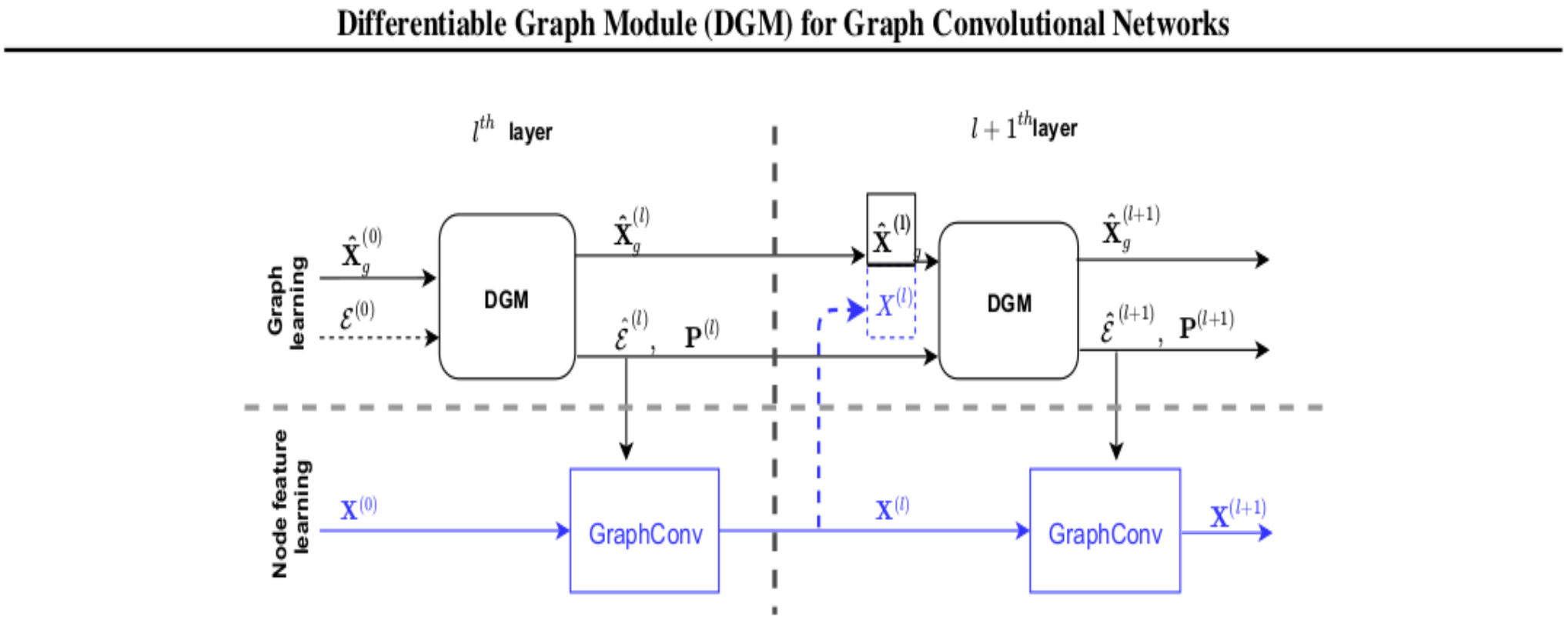
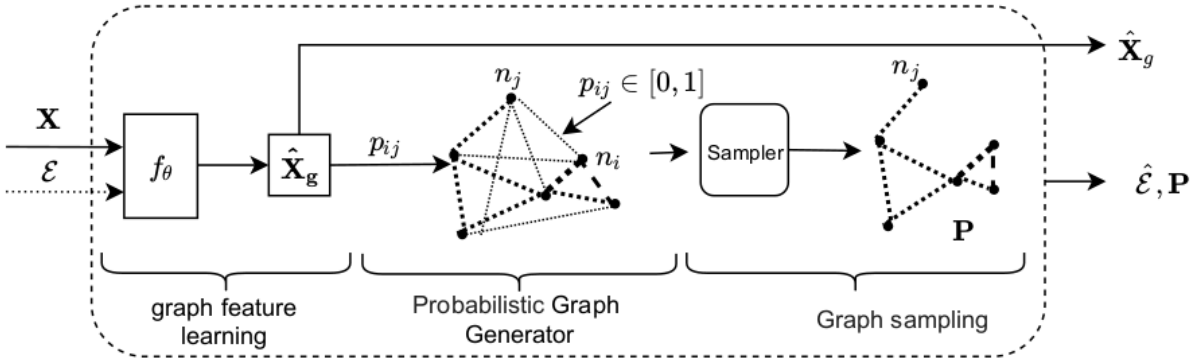


Figure 2. Forward propagation rule using the proposed DGM. We show the model architecture of two consecutive layers and the flow from input features to the predicted edges.

Method



Differentiable Graph Module (DGM) for Graph Convolutional Networks

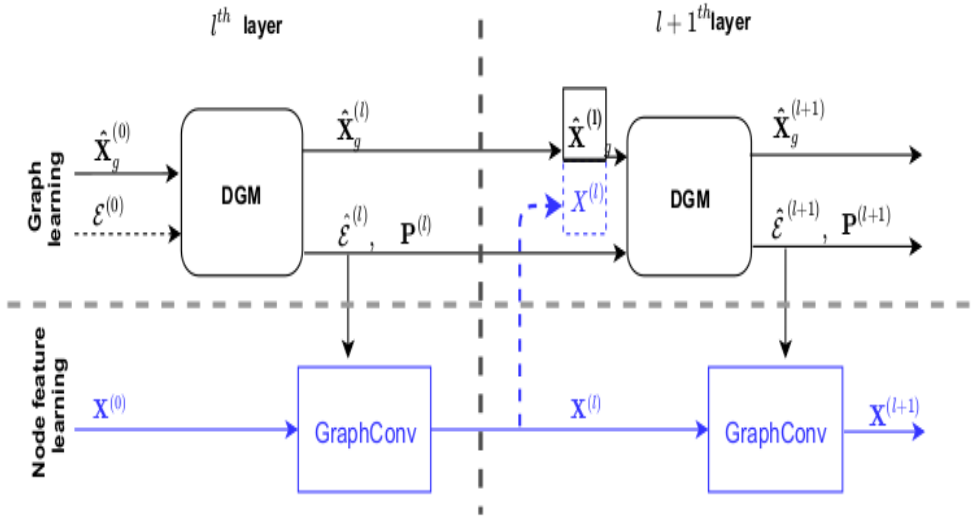


Figure 2. Forward propagation rule using the proposed DGM. We show the model architecture of two consecutive layers and the flow from input features to the predicted edges.

Method

- Loss function and and optimization: we imagine two losses, one for graph prediction and one for classification
- The sampling scheme we adopt does not allow the gradient of any classification loss function involving just graph features X to flow through the graph prediction branch of our network. To allow its optimization we exploit tools from reinforcement learning
- Suppose that, after a forward step, the network outputs the classification y_i for the input features $\mathbf{x_i}$ with sampled edges $\hat{\mathcal{E}}^{(l)}$ at layer l. We define the following graph loss function:

$$L_{graph} = \sum_i \delta(y_i, \tilde{y}_i) \prod_{l=1}^{L} \prod_{j:(i,j)\in\hat{\mathcal{E}}^{(l)}} p_{ij}^{(l)} \qquad (4)$$

, where $\delta(y_i, \tilde{y}_i)$ is a function taking value -1 if $y_i = \tilde{y}_i$ and 1 otherwise, and $\tilde{y}_i$ is the ground truth label.

Method

- The previous definition intrinsically weights unevenly positive and negative samples, especially in the early stages of the training where the classification accuracy is low. This drives the network to favor a uniform low probability estimation for all the edges.

- $$L_{graph} = \sum_{\alpha}^{Classes} \sum_{i \in \alpha} \delta_{\alpha}(y_i, \tilde{y}_i) \prod_{l=1}^{L} \prod_{j:(i,j) \in \hat{\mathcal{E}}^{(l)}} p_{ij}^{(l)}, \quad (5)$$

$$\delta_{\alpha}(y_i, \tilde{y}_i) = \begin{cases} \text{acc}_{\alpha} - 1 & \text{if } y_i = \tilde{y}_i \\ \text{acc}_{\alpha} & \text{otherwise} \end{cases} \quad (6)$$

with acc_{α} being the class accuracy computed on predictions y_i. Using a per-class accuracy rather than a global accuracy helps in dealing with uneven distribution of samples among different classes in the dataset.

Method

- **Multi-modal** datasets consist of two (or more) sets of features coming from different modalities. The graph can be learned from the one of the modalities and node representation from the other modality.
- → **M-GDM:** The only difference w.r.t. the forward propagation: we train the graph learning part on separate set of features dedicated for the graph learning purpose.
- **Out-of-sample extension:** From equation 3 we focus on learning the function f_θ and g_φ , hence the learn-able parameters θ and φ during the training. Since the graph in our case is dynamic and generated at each layer, it is easy to generate the new graph with a dynamic number of nodes as well. In the inductive setting,the parameters θ and φ are used to learn the representationsbased on the previously trained filters.

Application to disease prediction

- Alzheimer's disease prediction given imaging features (MRI, fMRI, PET) and non-imaging (demographics and genotypes) per patient. We pose this problem as a classifcation of each patient either of the 3 classes viz. Normal, Alzheimer's and Mild Cognitive Impairment (MCI).
- The graph is constructed on the entire population where each patient is considered as a node and the connectivity between the patient is based on the similarity in their respective non-imaging features.
- Imaging features are assigned to each node. Finally, the features for each node are learned from this setting and used for the classification task.
- For this experiment we use Tadpole (Marinescuet al., 2018) dataset which is a subset of the Alzheimer's Dis-GCNs are being leveraged to utilize such rich multi-modal disease Neuroimaging Initiative (adni.loni.usc.edu), consisting of 557 patients with 354 multi-modal features per patient.
- Imaging features are constituted of Magnetic Resonance Imaging, Positron Emission Tomography, cognitive tests, and CSF whereas non-imaging features are constituted of demographics (age, gender), genotype and average FDG PET value data.

Application to disease prediction

- Linear classifier represents a non-graph based method where results are obtained by ridge classifier. Multi-GCN (Kazi et al., 2019a), Spectral-GCN (Parisot et al., 2017) and InceptionGCN (Kazi et al., 2019b) are spectral approaches targeting the classification task. These three methods require a pre-defined graph obtained from non-imaging modality. We also add DGCNN as a baseline, as it dynamically builds the graph, the approach is similar to our method.

Method	Accuracy
Linear classifier	70.22± 06.32
Multi-GCN (Kazi et al., 2019a)	76.06 ± 00.72
Spectral-GCN (Parisot et al., 2017)	81.00 ± 06.40
InceptionGCN (Kazi et al., 2019b)	84.11 ± 04.50
DGCNN	84.59 ± 04.33
M-DGM	**90.05 ± 03.70**
DGM	**91.05 ± 05.93**

Table 1. The accuracy of classification on the Tadpole dataset. We compare the proposed method with respect to the state of the art. The table proves that DGCNN is a strong baseline to compare with in the further experiments.

Application to disease prediction

- we vary the graph features. In this setting, we keep the node feature constant to check the sensitivity of the model towards the graph-features and compare the performance of the classification task to DGCNN.

-

node-features	graph-features	DGCNN	M-DGM	DGM
M1	M1	82.98±03.35	**92.56±02.57**	89.88±03.95
M1	M2	85.65±05.90	90.05±03.70	**91.50±05.93**
M1	M1+M2	84.22±05.82	**90.96±03.59**	90.59±02.40
M1+M2	M1+M2	84.59±04.33	86.89±04.91	**90.42±03.87**
Mean		84.36±04.85	90.12±03.69	**90.60±04.04**

Table 2. The table represents the average accuracy of classification for the 10 fold cross validation in the transductive setting, for tadpole dataset. The first two columns show the feature type used for graph learning chosen between modality 1 and modality 2 corresponding to M1 and M2 respectively.

Application to disease prediction

- We also show the results in the inductive setting in table 3. For, this setting we keep 10% of the data completely unseen and train our model with remaining data in the regular fashion. During the inductive setting, we use the pretrained model for the filters, while the graph in each layer is constructed over the whole population including the 10% out of sample set.

node-features	graph-features	DGCNN	M-DGM	DGM
M 1	M 1	82.99±04.91	87.94±03.02	**88.12±03.65**
M 1	M 2	81.06±04.80	87.59±03.05	**88.48±04.58**
M 1	M 1 +M 2	81.95±06.17	86.70±04.43	**89.54±05.69**
M 1 +M 2	M 1 +M 2	84.39±04.57	**88.64±03.63**	87.23±03.53
Mean		82.60±05.11	87.72±03.53	**88.34±04.36**

Table 3. The table represents the average accuracy of classification for the 10 fold cross validation in the inductive setting, for tadpole dataset. 10 % of the data is kept completely unseen.