Learning Discrete Structures for Graph Neural Networks

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, Xiao He

Compiled by Hongming Shan
shanh@rpi.edu

Department of Biomedical Engineering
Rensselaer Polytechnic Institute

May 8, 2019

Summary

- GNNs are a popular class of machine learning models which can incorporate a sparse and discrete dependency structure between data points.
- GNNs can only be used when such a graph-structure is available. In practice, real-world graphs are often noisy and incomplete or might not be available at all.
- We propose to jointly learn the **graph structure** and **the parameters of graph convolutional networks (GCNs)**.
- Apply GCNs not only in scenarios where the given graph is incomplete or corrupted but also in those where a graph is not available.
- Experiments show that the proposed method outperforms related methods by a significant margin.

Outline

Relational learning

Relational learning: is concerned with methods that cannot only leverage the attributes of data points but also their relationships.

Example: Diagnosing a patient not only depends on the patient's vitals and demographic information but also on the same information about their relatives, the information about the hospitals they have visited, and so on.

Relational learning does not make the assumption of independence between data points but models their dependency explicitly.

When graph is not available

Graph has to be inferred or constructed

- Create a **k-nearest neighbor (kNN) graph** based on some measure of similarity between data points (e.g. LLE and Isomap). However, the efficacy of the resulting models hinges on the choice of k and, more importantly, on the choice of a suitable similarity measure over the input features.
- Simply use a **kernel matrix** to model the similarity of examples implicitly at the cost, however, of introducing a dense dependency structure which may be problematic from a computational point of view.

This work

- Learning *discrete* and *sparse* dependencies between data points while simultaneously training the parameters of graph convolutional networks (GCN). Intuitively, GCNs learn node representations by passing and aggregating messages between neighboring nodes.
- We propose to learn a generative probabilistic model for graphs, samples from which are used both during training and at prediction time.
- Edges are modelled with random variables whose parameters are treated as hyperparameters in a bilevel learning framework. We iteratively sample the structure while minimizing an inner objective (a training error) and optimize the edge distribution parameters by minimizing an outer objective (a validation error).

Graph Theory Basics

- A graph G is a pair (V, E) with $V = \{v_1, ..., v_N\}$ the set of vertices and $E \subseteq V \times V$ the set of edges. Let N be the number of vertices and M the number of edges.
- Each graph can be represented by an adjacency matrix A of size $N \times N$, where $A_{i,j} = 1$ if there is an edge from vertex v_i to vertex v_j, and $A_{i,j} = 0$ otherwise.
- The graph Laplacian is defined by $L = D - A$ where $D_{i,i} = \sum_j A_{i,j}$ and $D_{i,j} = 0$ if $i \neq j$.
- We denote the set of all $N \times N$ adjacency matrices by $\mathcal{H}_N$.

Graph Neural Networks

- All GNNs have the same two inputs.
    - A feature matrix $X \in \mathcal{X}_N \subset \mathbb{R}^{N \times n}$. n is # node features
    - A graph $G = (V, E)$ with adjacency matrix $A \in \mathcal{H}_N$.
- Given a set of class labels $\mathcal{Y}$ and a labeling function $y : V \to \mathcal{Y}$ that maps (a subset of) the nodes to their true class label, the objective is, given a set of training nodes $V_{\texttt{Train}}$, to learn a function

$$f_w : \mathcal{X}_N \times \mathcal{H}_N \to \mathcal{Y}^N$$

by minimizing some regularized empirical loss

$$L(w, A) = \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w), \tag{1}$$

where $w \in \mathbb{R}^d$ are the parameters of f_w, $f_w(X, A)_v$ is the output of f_w for node v, $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$ is a point-wise loss function, and Ω is a regularizer.

Graph Convolutional Network

An example of the function f_w is the following two hidden layer GCN that computes the class probabilities as

$$f_w(X, A) = \text{Softmax}(\hat{A}\ \text{ReLu}(\hat{A}\ X\ W_1)\ W_2), \tag{2}$$

where $w = (W_1, W_2)$ are the parameters of the GCN and $\hat{A}$ is the normalized adjacency matrix, given by $\hat{A} = \tilde{D}^{-1/2}(A + I)\tilde{D}^{-1/2}$ with $\tilde{D}_{i,i} = \sum_{i,j}(A + I)_j$.

Bilevel Programming in Machine Learning

- Bilevel programs are optimization problems where a set of variables occurring in the objective function are constrained to be an optimal solution of another optimization problem
- Given two objective functions F and L, the outer and inner objectives, and two sets of variables, $\theta \in \mathbb{R}^m$ and $w \in \mathbb{R}^d$, the outer and inner variables, a bilevel program is given by

$$\min_{\theta, w_\theta} F(w_\theta, \theta) \text{ such that } w_\theta \in \arg\min_{w} L(w, \theta). \tag{3}$$

- **Hyperparamter optimization:**
  - Inner objective can be a regularized training error
  - Outer objective can be the corresponding unregularized validation error.
  - θ would then be the coefficient (hyperparameter) of the regularizer
  - w the parameters of the model.

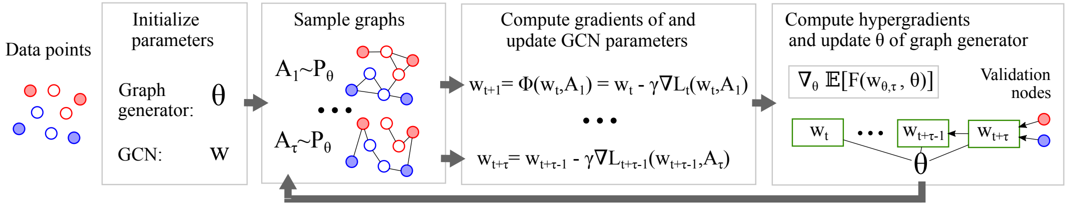
Solving Bilevel Programming

- **Challenge:** the solution sets of the inner problem are usually not available in closed-form
- **Standard approach:** replacing the minimization of L with the repeated application of an iterative optimization dynamics Φ such as (stochastic) gradient descent. Let $w_{\theta,T}$ denote the inner variables after T iterations of the dynamics Φ, that is, $w_{\theta,T} = \Phi(w_{\theta,T-1}, \theta) = \Phi(\Phi(w_{\theta,T-2}, \theta), \theta), \ldots$.
- **Hypergradient:** The gradient of the function $F(w_{\theta,T}, \theta)$ w.r.t. θ, denoted throughout as the *hypergradient* $\nabla_\theta F(w_{\theta,T}, \theta)$, is calculated as

$$\partial_w F(w_{\theta,T}, \theta) \nabla_\theta w_{\theta,T} + \partial_\theta F(w_{\theta,T}, \theta). \tag{4}$$

The first term can be computed efficiently in time $O(T(d+m))$ with reverse-mode algorithmic differentiation by unrolling the optimization dynamics, repeatedly substituting $w_{\Phi,t} = \Phi(w_{\theta,t-1}, \theta)$ and applying the chain rule.

Learning discrete graph structures for GNNs


Data points
Initialize parameters
Graph generator: θ
GCN: w
Sample graphs
$A_1 \sim P_\theta$
$A_\tau \sim P_\theta$
Compute gradients of and update GCN parameters
$w_{t+1} = \Phi(w_t, A_1) = w_t - \gamma \nabla L_t(w_t, A_1)$
$w_{t+\tau} = w_{t+\tau-1} - \gamma \nabla L_{t+\tau-1}(w_{t+\tau-1}, A_\tau)$
Compute hypergradients and update θ of graph generator
$\nabla_\theta \mathbb{E}[F(w_{\theta,\tau}, \theta)]$
Validation nodes
w_t
$w_{t+\tau-1}$
$w_{t+\tau}$
θ

Schematic representation of our approach for learning discrete graph structures for GNNs.

Jointly Learning the Structure and Parameters

- **Outer objective function**: aims to find an optimal discrete graph structure

$$F(w_A, A) = \sum_{v \in V_{\texttt{Val}}} \ell(f_{w_A}(X, A)_v, y_v) \tag{5}$$

  Here, w_A is the minimizer, assumed unique, of L for a *fixed adjacency matrix* A.

- **Inner objective function**: aims to find the optimal parameters of a GCN given a graph

$$L(w, A) = \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w), \tag{6}$$

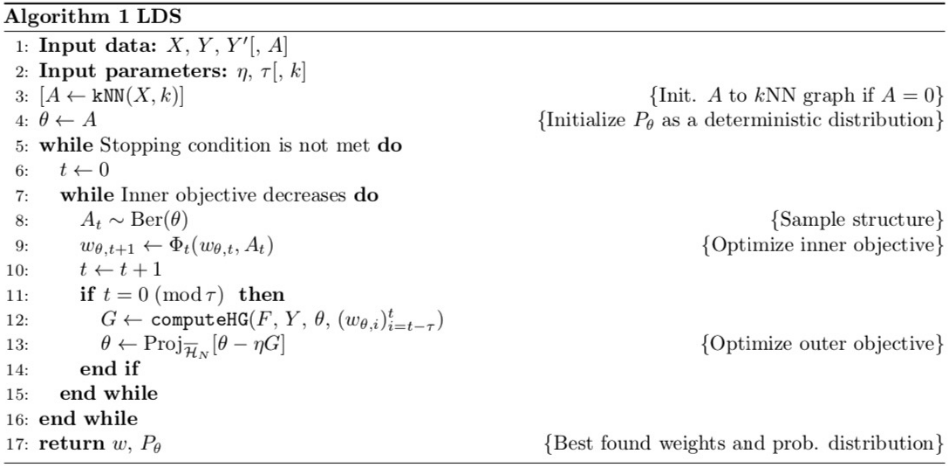
Probability distribution over graph

- The resulting bilevel problem is intractable due to both continuous and discrete-valued variables.
- Maintaining a generative model for the graph structure and reformulate the bilevel program in terms of the (continuous) parameters of the resulting distribution over discrete graphs.
- Model each edge with a Bernoulli random variable.
  - Let $\overline{\mathcal{H}}_N = \mathrm{Conv}(\mathcal{H}_N)$ be the convex hull of the set of all adjacency matrices for N nodes.
  - By modeling all possible edges as a set of mutually independent Bernoulli random variables with parameter matrix $\theta \in \overline{\mathcal{H}}_N$ we can sample graphs as $\mathcal{H}_N \ni A \sim \mathrm{Ber}(\theta)$.
  - Inner and outer objective functions can then be replaced, by using the expectation over graph structures. The resulting bilevel problem can be written as

$$\min_{\theta \in \overline{\mathcal{H}}_N} \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[F(w_\theta, A) \right] \tag{7}$$

$$\text{such that } \; w_\theta = \arg\min_w \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[L(w, A) \right]. \tag{8}$$

Algorithm



Algorithm 1 LDS

```
 1: Input data: $X$, $Y$, $Y'$[, $A$]
 2: Input parameters: $\eta$, $\tau$[, $k$]
 3: [$A \leftarrow \texttt{kNN}(X, k)$]                    {Init. $A$ to $k$NN graph if $A = 0$}
 4: $\theta \leftarrow A$                                   {Initialize $P_\theta$ as a deterministic distribution}
 5: while Stopping condition is not met do
 6:   $t \leftarrow 0$
 7:   while Inner objective decreases do
 8:     $A_t \sim \mathrm{Ber}(\theta)$                      {Sample structure}
 9:     $w_{\theta,t+1} \leftarrow \Phi_t(w_{\theta,t}, A_t)$   {Optimize inner objective}
10:     $t \leftarrow t + 1$
11:     if $t = 0 \pmod{\tau}$ then
12:       $G \leftarrow \texttt{computeHG}(F, Y, \theta, (w_{\theta,i})_{i=t-\tau}^{t})$
13:       $\theta \leftarrow \mathrm{Proj}_{\overline{\mathcal{H}}_N}[\theta - \eta G]$   {Optimize outer objective}
14:     end if
15:   end while
16: end while
17: return $w$, $P_\theta$                               {Best found weights and prob. distribution}
```

Three main objectives in experiments

- Evaluated LDS on node classification problems where a graph structure is **available** but where a certain fraction of edges is missing. Here, we compared LDS with graph-based learning algorithms including vanilla GCNs.
- Validate our hypothesis that LDS can achieve competitive results on semi-supervised classification problems for which a graph is ***not* available**. To this end, we compared LDS with a number of existing semi-supervised classification approaches. We also compared LDS with algorithms that first create k-NN affinity graphs on the data set.
- Analyzed the **learned graph generative model** to understand to what extent LDS is able to learn meaningful edge probability distributions even when a large fraction of edges is missing.

Objective 1

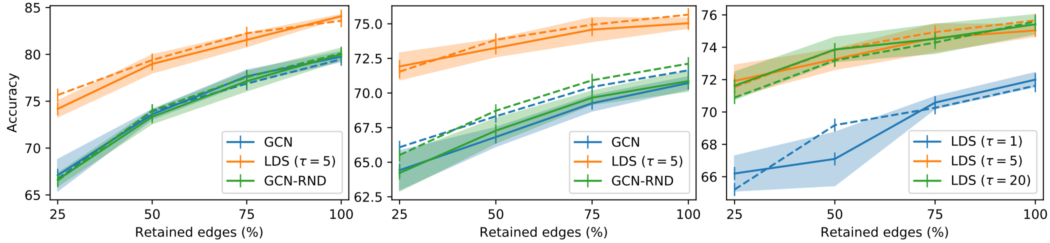


Figure: Mean accuracy ± standard deviation on validation (early stopping; dashed lines) and test (solid lines) sets for edge deletion scenarios on Cora (left) and Citeseer (center). (Right) Validation of the number of steps τ used to compute the hypergradient (Citeseer); $\tau = 1$ corresponds to alternating minimization. All results are obtained from five runs with different random seeds.

Table: Test accuracy (± standard deviation) in percentage on various classification datasets. The best results and the statistical competitive ones (by paired t-test with $\alpha = 0.05$) are in bold. All experiments have been repeated with 5 different random seeds. We compare kNN-LDS to several supervised baselines and semi-supervised learning methods. No graph is provided as input. kNN-LDS achieves high accuracy results on most of the datasets and yields the highest gains on datasets with underlying graphs (Citeseer, Cora). kNN-LDS (d) is the dense version of LDS.

	Wine	Cancer	Digits	Citeseer	Cora	20news	FMA
LogReg	92.1 (1.3)	**93.3 (0.5)**	85.5 (1.5)	62.2 (0.0)	60.8 (0.0)	42.7 (1.7)	37.3 (0.7)
Linear SVM	93.9 (1.6)	**90.6 (4.5)**	87.1 (1.8)	58.3 (0.0)	58.9 (0.0)	40.3 (1.4)	35.7 (1.5)
RBF SVM	**94.1 (2.9)**	**91.7 (3.1)**	86.9 (3.2)	60.2 (0.0)	59.7 (0.0)	41.0 (1.1)	**38.3 (1.0)**
RF	93.7 (1.6)	**92.1 (1.7)**	83.1 (2.6)	60.7 (0.7)	58.7 (0.4)	40.0 (1.1)	**37.9 (0.6)**
FFNN	89.7 (1.9)	**92.9 (1.2)**	36.3 (10.3)	56.7 (1.7)	56.1 (1.6)	38.6 (1.4)	33.2 (1.3)
LP	89.8 (3.7)	76.6 (0.5)	**91.9 (3.1)**	23.2 (6.7)	37.8 (0.2)	35.3 (0.9)	14.1 (2.1)
ManiReg	90.5 (0.1)	81.8 (0.1)	83.9 (0.1)	67.7 (1.6)	62.3 (0.9)	**46.6 (1.5)**	34.2 (1.1)
SemiEmb	91.9 (0.1)	89.7 (0.1)	**90.9 (0.1)**	68.1 (0.1)	63.1 (0.1)	**46.9 (0.1)**	34.1 (1.9)
Sparse-GCN	63.5 (6.6)	72.5 (2.9)	13.4 (1.5)	33.1 (0.9)	30.6 (2.1)	24.7 (1.2)	23.4 (1.4)
Dense-GCN	90.6 (2.8)	90.5 (2.7)	35.6 (21.8)	58.4 (1.1)	59.1 (0.6)	40.1 (1.5)	34.5 (0.9)
RBF-GCN	90.6 (2.3)	**92.6 (2.2)**	70.8 (5.5)	58.1 (1.2)	57.1 (1.9)	39.3 (1.4)	33.7 (1.4)
kNN-GCN	93.2 (3.1)	**93.8 (1.4)**	**91.3 (0.5)**	68.3 (1.3)	66.5 (0.4)	41.3 (0.6)	**37.8 (0.9)**
kNN-LDS (d)	**97.5 (1.2)**	**94.9 (0.5)**	**92.1 (0.7)**	**70.9 (1.3)**	**70.9 (1.1)**	**45.6 (2.2)**	**38.6 (0.6)**
kNN-LDS	**97.3 (0.4)**	**94.4 (1.9)**	**92.5 (0.7)**	**71.5 (1.1)**	**71.5 (0.8)**	**46.4 (1.6)**	**39.7 (1.4)**

Table: Initial number of edges and expected number of sampled edges of learned graph by LDS.

% Edges	25%	50%	75%	100%
Cora Initial	1357	2714	4071	5429
Cora Learned	3635.6	4513.9	5476.9	6276.4
Citeseer Initial	1183	2366	3549	4732
Citeseer Learned	3457.4	4474.2	7842.5	6745.2

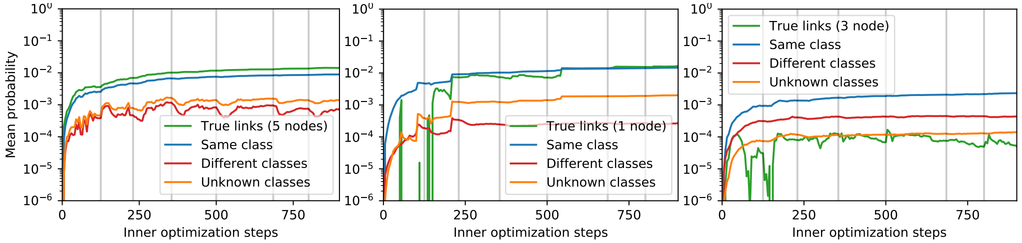


Figure: Mean edge probabilities to nodes aggregated wrt four groups during LDS optimization, in log_{10} scale for three example nodes. For each example node, all other nodes are grouped by the following criteria: (a) adjacent in the ground truth graph; (b) same class membership; (c) different class membership; and (d) unknown class membership. Probabilities are computed with LDS ($\tau = 5$) on Cora with 25% retained edges. From left to right, the example nodes belong to the training, validation, and test set, respectively. The vertical gray lines indicate when the inner optimization dynamics restarts, that is, when the weights of the GCN are reinitialized.

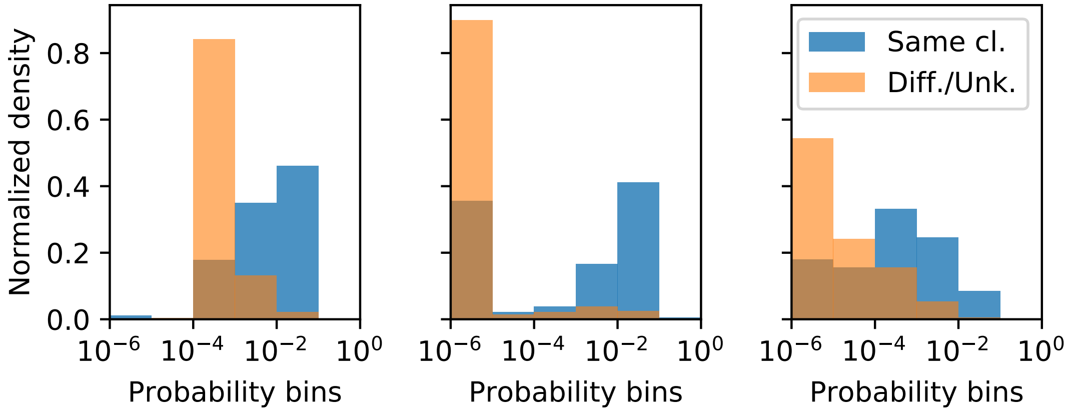


Figure: Normalized histograms of edges' probabilities for the same nodes.

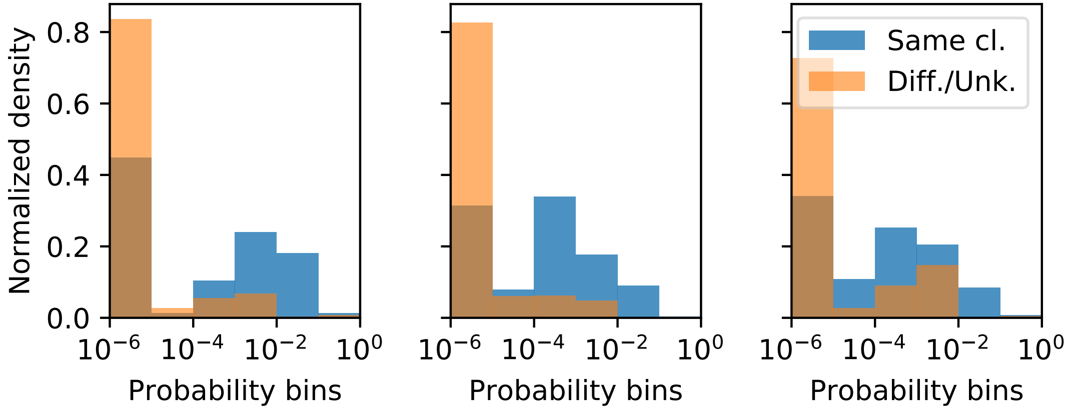


Figure: Histograms for three Citeseer test nodes, missclassified by kNN-GCN and rightly classified by kNN-LDS.

Conclusion

- Propose a framework that simultaneously learns the graph structure and the parameters of a GNN.
- The strengths of LDS are its high accuracy gains on typical semi-supervised classification datasets at a reasonable computational cost.

Limitations

- While relatively efficient, it cannot currently scale to large datasets
- We evaluated LDS only in the transductive setting, when all data points (nodes) are available during training. Adding additional nodes after training (the inductive setting) would currently require retraining the entire model from scratch
- When sampling graphs, we do not currently enforce the graphs to be connected

Thanks!