GRAPH STRUCTURE LEARNING FOR GCNS

Luca Franceschi[1,3]*, **Mathias Niepert**[2], **Massimiliano Pontil**[1,3], **Xiao He**[2]

1. Istituto Italiano di Tecnologia, Genoa, Italy
{luca.franceschi,massimiliano.pontil}@iit.it
2. NEC Labs Europe, Heidelberg, Germany
{mathias.niepert,xiao.he}@neclab.eu
3. Dept of Computer Science, University College London, UK

ABSTRACT

Graph convolutional networks (GCNs) are a popular class of neural networks leveraging sparse and discrete dependency structures between data points. Applying GCNs to real-world applications, however, is challenging due to the fact that graph-structures are noisy and incomplete or may not be available at all. We propose a method for jointly learning the graph and weights of a GCN by approximately solving a bilevel program whose inner and outer objectives aim at optimizing, respectively, the parameters of a GCN and its graph structure. We show that the proposed method, named LDS, outperforms related approaches by a significant margin on datasets where the dependency structure is either incomplete or completely missing[1].

1 INTRODUCTION

Relational learning aims to leverage not only the attributes of data points but also their dependency. Diagnosing a patient, for example, 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. Graph neural networks (GNNs) (Scarselli et al., 2009) are a popular class of learning methods that incorporate sparse and discrete dependency structures between data points.

While a graph structure is available in some domains, in others it has to be inferred or constructed. A possible approach is to first create a k-nearest neighbor (kNN) graph based on some measure of similarity between data points. This is a common strategy used by several learning methods such as LLE (Roweis & Saul, 2000) and Isomap (Tenenbaum et al., 2000). A major shortcoming of this approach, however, is that 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. In any case, the graph creation and parameter learning steps are independent and require heuristics and trial and error. Alternatively, one could 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.

We propose LDS (learning descrete structures), a novel framework for learning *discrete* and *sparse* dependencies between data points while simultaneously training the parameters of graph convolutional networks (GCN), a class of GNNs. Intuitively, GCNs learn node representations by passing and aggregating messages between neighboring nodes (Kipf & Welling, 2017; Monti et al., 2017; Gilmer et al., 2017; Hamilton et al., 2017; Duran & Niepert, 2017; Velickovic et al., 2018). 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 (Franceschi et al., 2018). 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).

*Work done while visiting researcher at NEC Labs Europe.

[1]This is an abridged version of the paper *Learning Discrete Structures for Graph Neural Networks*, to appear at ICML 2019, and available as pre-print at https://arxiv.org/abs/1903.11960.

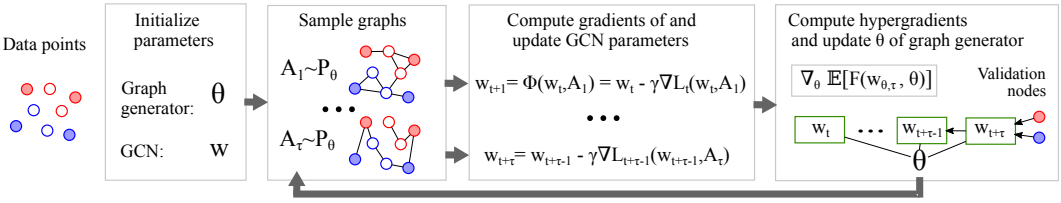

Figure 1: Schematic representation of our approach for learning discrete graph structures for GNNs.

To the best of our knowledge, LDS is the first method that simultaneously learns the graph structure and parameters of a GNN for semi-supervised classification. Moreover, and this might be of independent interest, we adapt gradient-based hyperparameter optimization to work for a class of discrete hyperparameters (unweighted edges, in this work). The proposed approach makes GCNs applicable to problems where the graph is incomplete or entirely missing. We conduct a series of experiments and show that the proposed method is competitive with and often outperforms existing approaches. We also verify that the resulting graph generative models have meaningful edge probabilities.

2 Learning Discrete Graph Structures

Given a graph G with N nodes, input features $X \in \mathcal{X} \subseteq \mathbb{R}^{N \times n}$, and an (unweighted) adjacency matrix $A \in \{0,1\}^{N \times N} := \mathcal{H}$, we are interested in GCNs of the form

$$f_w : \mathcal{X} \times \mathcal{H} \to \mathcal{Y}^N \qquad f_w(X, A) = \hat{y}$$

where $\mathcal{Y}$ is a set of class labels and $w \in \mathbb{R}^d$ are the model's weights. Given a subset of training nodes $V_{\texttt{Train}}$ and a point-wise loss function ℓ, the parameters of f can be learned by minimizing some regularized empirical loss $L(w, A) = \sum_{v \in V_{\texttt{Train}}} \ell(\hat{y}_v, y_v) + \Omega(w)$.

Now, let us suppose that the true adjacency matrix A is missing or incomplete. Since, ultimately, we are interested in finding a model that minimizes the generalization error, we assume the existence of a second subset of instances with known target, $V_{\texttt{Val}}$ (the validation set), from which we can estimate the generalization error. Hence, we propose to find $A \in \mathcal{H}$ that minimizes the function

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

where w_A is the minimizer, assumed unique, of the training loss L for a *fixed adjacency matrix* A. L and F can be considered as the inner and outer objectives of a mixed-integer bilevel programming problem where the outer objective aims to find an optimal discrete graph structure and the inner objective the optimal parameters of the GCN given a graph. Unfortunately, the resulting bilevel problem is intractable. Instead of solving it directly, we propose to learn the parameters of a graph-generative model and to solve the bilevel programming problem in expectation. We maintain the generative model for graph structures and reformulate the bilevel program in terms of the (continuous) parameters of its distribution. Specifically, we propose to model each edge with a Bernoulli random variable. Let $\overline{\mathcal{H}} = \mathrm{Conv}(\mathcal{H})$ be the convex hull of the set of adjacency matrices for N nodes. By modeling all possible edges as a set of mutually independent Bernoulli random variables with parameter matrix $\theta \in \overline{\mathcal{H}}$ we can sample graphs $\mathcal{H} \ni A \sim \mathrm{Ber}(\theta)$, and restate the bilevel problem by taking the expectation over graphs as

$$\min_{\theta \in \overline{\mathcal{H}}} \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[F(w_\theta, A) \right] \qquad w_\theta = \arg\min_w \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[L(w, A) \right]; \tag{1}$$

both the inner and the outer objectives are now continuous (and possibly smooth) functions of the Bernoulli parameters. The bilevel problem (1) is still challenging to solve efficiently since the solution of the inner problem is not available in closed form for GCNs and the expectations are intractable to compute exactly. An efficient algorithm, therefore, will only be able to find approximate stochastic solutions, that is, $\theta_{i,j} \in (0, 1)$. Before describing a method to solve the optimization problem approximately with hypergradient descent, we turn to the question of obtaining a final GCN model that we can use for prediction. For a given distribution P_θ, we propose to compute an

Algorithm 1 LDS

Input data: $X, Y, Y'[, A]$
Input parameters: η, $\tau[, k]$
$[A \leftarrow \texttt{kNN}(X, k)]$ {kNN graph if $A = 0$}
$\theta \leftarrow A$ {Init. P_θ as a deterministic distr.}
while Stopping condition is not met **do**
  $t \leftarrow 0$
  while Inner objective decreases **do**
    $A_t \sim \mathrm{Ber}(\theta)$ {Sample structure}
    $w_{\theta,t+1} \leftarrow \Phi_t(w_{\theta,t}, A_t)$ {Optimize inner objective}
    $t \leftarrow t + 1$
    if $t = 0 \pmod \tau$ **then**
      $G \leftarrow \texttt{RevHG}(F, Y, \theta, (w_{\theta,i})_{i=t-\tau}^{t})$
      $\theta \leftarrow \mathrm{Proj}_{\overline{\mathcal{H}}}[\theta - \eta G]$ {Optimize outer objective}
    end if
  end while
end while
return w, P_θ {Best found weights and probability distribution}

empirical estimate of the output of f_w as $\hat{f}_w(X) = \frac{1}{S}\sum_{i=1}^{S} f_w(X, A_i)$, where $S > 0$ is the number of graphs we wish to draw; $\hat{f}_w$ is an unbiased estimator of the expected output $\mathbb{E}_A[f_w(X, A)]$. Hence, to use a GCN f_w learned with the bilevel formulation for prediction, we sample S graphs from the distribution P_θ and compute the prediction as the empirical mean of the values of f_w. We argue that, for sparse graphs and small S, this is more efficient than evaluating a "dense GCN." Indeed, computing $\hat{f}_w$ has a cost of $O(SCd)$ compared to $O(N^2 d)$ for a fully connected graph, where $C = \sum_{ij} \theta_{ij}$ is the expected number of edges, and d is the dimension of the weights. Another advantage of using a graph-generative model is that we can interpret it probabilistically.

2.1 Structure Learning via Hypergradient Descent

We now discuss a practical algorithm to approach the bilevel problem[2] defined by Eq. (1). Regarding the inner problem, we note that the expectation $\mathbb{E}_{A\sim\mathrm{Ber}(\theta)}\left[L(w, A)\right] = \sum_i P_\theta(A_i) L(w, A_i)$ is composed of a sum of $2^{N\cdot N}$ terms, intractable even for relatively small graphs. We can, however, choose a tractable stochastic learning dynamics Φ such as stochastic gradient descent (SGD),

$$w_{\theta,t+1} = \Phi(w_{\theta,t}, A_t) = w_{\theta,t} - \gamma_t \nabla L(w_{\theta,t}, A_t) \tag{2}$$

where γ_t is a learning rate and $A_t \sim \mathrm{Ber}(\theta)$ is drawn at each iteration. For $t \to \infty$, $w_{\theta,t}$ converges to a weight vector w_θ that depends on the edges' probability distribution (Bottou, 2010).

Let $w_{\theta,T}$ be an approximate minimizer of $\mathbb{E}L$ (where T may depend on θ). We now need to compute an estimator for the hypergradient $\nabla_\theta \mathbb{E}_{A\sim\mathrm{Ber}(\theta)} F$.We have

$$\nabla_\theta \mathbb{E} F(w_{\theta,T}, A) = \mathbb{E}\nabla_\theta F(w_{\theta,T}, A) = \mathbb{E}\left[\partial_w F(w_{\theta,T}, A)\nabla_\theta w_{\theta,T} + \partial_A F(w_{\theta,T}, A)\nabla_\theta A\right] \tag{3}$$

where we can swap the gradient and expectation operators since the expectation is over a finite random variable, assuming that the loss function F bounded. We use the so-called straight-through estimator (Bengio et al., 2013) and set $\nabla_\theta A := I$ (which would be 0 a.e. otherwise; see also Appendix B); $\nabla_\theta A$ appears both explicitly in Eq. (3) and in the computation of $\nabla_\theta w_{\theta,T}$, through $\partial_\theta \Phi(w_{\theta,t}, A_t)$ (see Franceschi et al., 2017, for details). Finally, we take the single sample Monte Carlo estimator of Eq. (3) to update the parameters θ with projected gradient descent on the unit hypercube.

Computing the hypergradient by fully unrolling the dynamics may be too expensive both in time and memory We propose to truncate the computation and estimate the hypergradient every τ iterations, where τ is a parameter of the algorithm. This is essentially an adaptation of truncated back-propagation through time (Werbos, 1990; Williams & Peng, 1990). A sketch of the method is presented in Algorithm 1. Inputs and operations in squared brackets are optional. The algorithm contains stopping conditions at the outer and at the inner level. While it is natural to implement the latter with a decrease condition on the inner objective, we find it useful to implement the first with a simple early stopping criterion. A fraction of the examples in the validation set is held-out to compute after each outer iteration the accuracy using the predictions of the empirically expected model, and we let the optimization procedure terminate if there is no improvement for some consecutive outer loops. This helps avoiding overfitting the outer objective of Eq. (1), which may be a concern in this context given the quantity of (hyper)parameters being optimized. The hypergradients estimated

[2]We refer to the appendix for a short introduction to bilevel programming for machine learning.

Table 1: Test accuracy ($\pm$ standard deviation, 5 random seeds) 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. 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).

	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)
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)**
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	88.2 (3.3)	86.0 (4.0)	82.8 (3.3)	67.7 (1.6)	62.3 (0.9)	**46.6 (1.5)**	34.2 (1.1)
SemiEmb	88.9 (3.7)	86.4 (5.4)	**90.7 (2.7)**	68.1 (0.7)	62.9 (1.1)	**46.3 (1.6)**	35.5 (1.9)
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	93.1 (2.7)	**93.4 (2.8)**	**91.3 (2.0)**	69.5 (1.1)	66.5 (0.4)	43.3 (0.4)	**37.9 (0.5)**
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)**

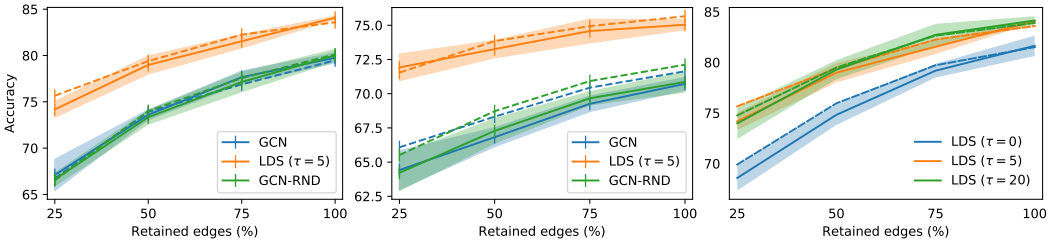

Figure 2: Mean accuracy $\pm$ standard deviation on validation (dashed lines) and test (solid lines) sets for incomplete graph scenario on Cora (Left) and Citeseer (Center). (Right) Validation of the number of steps τ used to compute the hypergradient (Cora); $\tau = 0$ is alternating minimization.

with Algorithm 1 at each outer iteration are biased. The bias stems from both the straight-trough estimator and from the truncation procedure introduced in lines 11-13 (Tallec & Ollivier, 2017). Nevertheless, we find empirically that the algorithm is able to make reasonable progresses, finding configurations in the distribution space that are beneficial for the tasks at hand.

3 Experiments

We conducted a series of experiments with the aim of evaluating LDS on node (semi-supervised) classification problems where a graph structure is available but incomplete, or not available. In the first case we show that the proposed algorithm is able to recover a meaningful edge distribution, in the latter case we show that LDS is competitive with respect to various baselines and variations, as well as other popular semi-supervised learning methods such as such as label propagation (LP) (Zhu et al., 2003), manifold regularization (ManiReg) (Belkin et al., 2006), and semi-supervised embedding (SemiEmb) (Weston et al., 2012). We use two popular benchmark datasets, Cora and Citeseer (Sen et al., 2008), used to evaluate relational learners in general and GCNs in particular, as well as datasets available in scikit-learn (Pedregosa et al., 2011), and a recent genre classification dataset (Defferrard et al., 2017) named FMA. To evaluate the efficacy of LDS in the "incomplete graph scenario", we randomly sample 25%, 50%, 75%, and 100% of the edges of Cora and Citeseer.

Table 1 lists the results for the semi-supervised classification problems. The supervised learning baselines (LogReg, Linear SVM, RBF SVM, FFNN) work well on some datasets such as Wine and Cancer but fail to provide competitive results on others; semi-supervised learning baselines can only improve upon the supervised learning baselines on few datasets.

The results on the incomplete graphs are shown in Figure 2. For each percentage of retained edges the accuracy on the validation (used for early stopping) and the

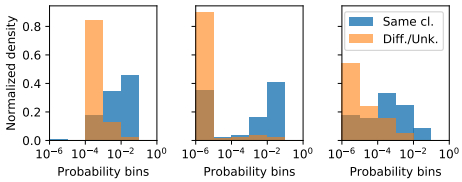

Figure 3: Histograms for three Cora nodes belonging either to the train (left), validation (center) and test set (right).

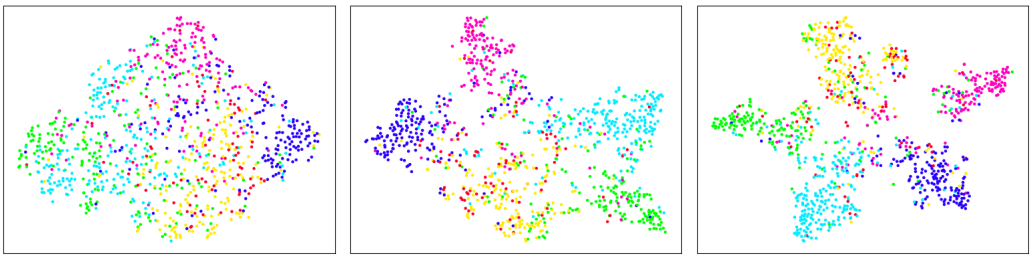

Figure 4: T-SNE visualization of the output activations (before the classification layer) on the Citeseer dataset. Left: Dense-GCN, Center: kNN-GCN, Right kNN-LDS

test sets are plotted. LDS achieves accuracy gains of up to 7 percentage points. Notably, the proposed method improves the generalization accuracy of GCN models also when the given graph is that of the respective dataset (100% of edges retained). We achieve new SotA results on Citeseer and Cora with a wide margin. Conversely, adding random edges during training (GCN-RND) does not help decreasing the generalization error. In Figure 3 we report the normalized histograms of the optimized edges probabilities for three nodes from Cora (25% edges retained), sorted into six bins in $\log_{10}$-scale. Edges are divided in two groups: edges between nodes of the same class (blue) and between nodes of unknown or different classes (orange). LDS is able to learn highly non-uniform edge probabilities that reflect the class membership of the nodes.

We further visualize the embeddings learned by GCN and LDS using T-SNE (Maaten & Hinton, 2008). Figure 4 depicts the T-SNE visualizations of the embeddings learned on Citeseer with Dense-GCN (left), kNN-GCN (center), and kNN-LDS (right). As can be seen, the embeddings learned by kNN-LDS provides the best separation among different classes.

4 Conclusion

We propose LDS, a framework that simultaneously learns the graph structure and the parameters of a GNN. While we have used a specific GCN variant (Kipf & Welling, 2017) in the experiments, the method is more generally applicable to other GNNs. The strengths of LDS are its high accuracy gains on typical semi-supervised classification datasets at a reasonable computational cost. Moreover, due to the graph generative model LDS learns, the edge parameters have a probabilistic interpretation.

The method has its limitations. While relatively efficient, it cannot currently scale to large datasets: this would require an implementation that works with mini-batches of nodes. 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. This is something we anticipate to improve the results, but this would require a more sophisticated sampling strategy. All of these shortcomings motivate future work. Alongside these considerations, we also intend to experiment with different strategies to compute the hypergradient in Eq. (3), for instance by making use of recurrent back-propagation (Liao et al., 2018).

In addition, we hope that suitable variants of LDS algorithm will also be applied to other problems such as neural architecture search or to tune other discrete hyperparameters.

References

Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Kristin P Bennett, Jing Hu, Xiaoyun Ji, Gautam Kunapuli, and Jong-Shi Pang. Model selection via bilevel optimization. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pp. 1922–1929. IEEE, 2006.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.

Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. Fma: A dataset for music analysis. In *18th International Society for Music Information Retrieval Conference*, 2017. URL `https://arxiv.org/abs/1612.01840`.

Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pp. 318–326, 2012.

Alberto Garcia Duran and Mathias Niepert. Learning graph representations with embedding propagation. In *Advances in Neural Information Processing Systems*, pp. 5119–5130, 2017.

Matthias Feurer and Frank Hutter. Hyperparameter optimization. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (eds.), *Automatic Machine Learning: Methods, Systems, Challenges*, pp. 3–38. Springer, 2018. In press, available at http://automl.org/book.

Rémi Flamary, Alain Rakotomamonjy, and Gilles Gasso. Learning constrained task similarities in graph-regularized multi-task learning. *Regularization, Optimization, Kernels, and Support Vector Machines*, pp. 103, 2014.

Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. *ICML*, 2017.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. *ICML*, 2018.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *ICLR*, 2018.

Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam, 2008.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *ICLR*, 2017.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.

Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, KiJung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pp. 3088–3097, 2018.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pp. 2113–2122, 2015.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *ICLR*, 2017.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, volume 1, pp. 3, 2017.

Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 27–38. ACM, 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pp. 3528–3536, 2015.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, pp. 2627–2636, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR*, 2018.

Paul J Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pp. 639–655. Springer, 2012.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pp. 912–919, 2003.

A 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 (see Colson et al., 2007, for an overwiew). Formally 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

$$\min_{\theta, w_\theta} F(w_\theta, \theta) \;\text{ such that }\; w_\theta \in \arg\min_{w} L(w, \theta). \tag{4}$$

Bilevel programs arise in numerous situations such as hyperparmeter optimization, adversarial, multi-task, and meta-learning (Bennett et al., 2006; Flamary et al., 2014; Muñoz-González et al., 2017; Franceschi et al., 2018).

Solving problem 4 is challenging since the solution sets of the inner problem are usually not available in closed-form. A standard approach involves replacing the minimization of L with the repeated application of an iterative optimization dynamics Φ such as (stochastic) gradient descent (Domke, 2012; Maclaurin et al., 2015; Franceschi et al., 2017). Let $w_{\theta,T}$ denote the inner variables after T iterations of the dynamics Φ, that is, $w_{\theta,T} = \Phi(w_{\theta,T-1}, \theta) = \Phi(\Phi(w_{\theta,T-2}, \theta), \theta)$, and so on. Now, if θ and w are real-valued and the objectives and dynamics smooth, we can compute the gradient of the function $F(w_{\theta,T}, \theta)$ w.r.t. θ, denoted throughout as the *hypergradient* $\nabla_\theta F(w_{\theta,T}, \theta)$, as

$$\partial_w F(w_{\theta,T}, \theta) \frac{\mathrm{d}w_{\theta,T}}{\mathrm{d}\theta} + \nabla_\theta F(w_{\theta,T}, \theta), \tag{5}$$

where the symbol ∂ denotes the the Jacobian[3], ∇ the gradient (of a scalar functions) and $\frac{\mathrm{d}}{\mathrm{d}\theta}$ is the total derivative. The first term can be computed efficiently in time $O(T(d+m))$ with reverse-mode algorithmic differentiation (Griewank & Walther, 2008) by unrolling the optimization dynamics, repeatedly substituting $w_{\theta,t} = \Phi(w_{\theta,t-1}, \theta)$ and applying the chain rule. This technique allows to optimize a number of hyperparameters several orders of magnitude greater than classic methods for hyperparameter optimization (Feurer & Hutter, 2018).

B GRADIENT ESTIMATION FOR DISCRETE RANDOM VARIABLES

Due to the intractable nature of the two bilevel objectives, LDS needs to estimate the hypergradients through a stochastic computational graph (Schulman et al., 2015). Using the score function estimator, also known as REINFORCE (Williams, 1992), would treat the outer objective as a black-box function and would not exploit F being differentiable w.r.t. the sampled adjacency matrices and the inner optimization dynamics. Conversely, the path-wise estimator is not readily applicable, since the random variables are discrete.

LDS borrows from an heuristic solution proposed before (Bengio et al., 2013), at the cost of having biased estimates. Given a function $h(z)$, where z is a discrete random variable (usually Bernoulli distributed) whose distribution depends on parameters θ, the technique consists in computing an estimator of the gradient of $\ell(\theta) = \mathbb{E}_z h(z)$ as

$$\hat{g}(z) = \frac{\partial h(z)}{\partial z}. \tag{6}$$

Note that, while $\frac{\partial z}{\partial \theta}$ is 0 almost everywhere, and not defined on the "threshold" points (being z discrete), h may still be *a smooth function of* z. In this case Eq. 6 is well defined and it yields, in general, non-zero quantities. This can be viewed as "setting" $\frac{\partial z}{\partial \theta}$ to the identity, or, from another point of view, as "ignoring" the hard thresholds on the computational graphs. $\hat{g}$ is a random variable that depends, again, from θ, and the true gradient $\nabla \ell(\theta)$ can be estimated by drawing (one sample) from $\hat{g}$.

As an illustrative example, consider the very simple case where $h(z) = (as - b)^2/2$ for scalars a and b, with $z \sim \mathrm{Ber}(\theta)$, $\theta \in [0, 1]$. The gradient (derivative) of $\mathbb{E}\, h$ w.r.t. θ can be easily computed as

$$\frac{\partial}{\partial \theta} \mathbb{E}_{z \sim \mathrm{Ber}(\theta)} h(z) = \frac{\partial}{\partial \theta} \left[\theta \frac{(a-b)^2}{2} + (1-\theta) \frac{(-b)^2}{2} \right] = \frac{a^2}{2} - ab, \tag{7}$$

[3]Which is equal to the transpose of the gradient for (real-valued) scalar functions.

whereas the corresponding straight-through estimator, which is a random variable, is given by

$$\hat{g}(z) = \frac{\partial h(z)}{\partial z} = (az - b)a \qquad z \sim \mathrm{Ber}(\theta).$$

One has, however, that

$$\mathbb{E}_{z\sim\mathrm{Ber}(\theta)}\hat{g}(z) = \theta(a-b)a + (1-\theta)(-ab) = \theta a^2 - ab,$$

resulting in $\hat{g}$ to be biased for $\theta \neq \frac{1}{2}$.

There are other approaches to the problem of estimating gradients for discrete random variables. Recently, Jang et al. (2017); Maddison et al. (2017) presented a method based on continuous relaxations to reduce variance, which Tucker et al. (2017) combined with REINFORCE to obtain an unbiased estimator. Grathwohl et al. (2018) further introduced surrogate models to construct control variates for black-box functions. Unfortunately, these latter methods require to compute the function in the interior of the hypercube, possibly in multiple points (Tucker et al., 2017), facts that would introduce additional computational overhead[4].

[4]Recall that F can be computed only after (approximately) solving the inner optimization problem.