End-to-end learning and optimization on graphs

Bryan Wilder [1] **Eric Ewing** [2] **Bistra Dilkina** [2] **Milind Tambe** [1]

Abstract

Real-world applications often combine learning and optimization problems on graphs. For instance, our objective may be to cluster the graph in order to detect meaningful communities (or solve other common graph optimization problems such as maxcut, vertex cover, and so on). However, graphs or related attributes are often only partially observed, introducing learning problems such as link prediction which must be solved prior to optimization. We propose an approach to integrate a differentiable proxy for common graph optimization problems into training of machine learning models for tasks such as link prediction. This allows the model to focus specifically on the downstream task that its predictions will be used for. Experimental results show that our end-to-end system obtains better performance on example optimization tasks than can be obtained by combining state of the art link prediction methods with expert-designed graph optimization algorithms.

We encourage readers to view the final version of the paper at `https://arxiv.org/abs/1905.13732`, which contains significant changes.

1. Introduction

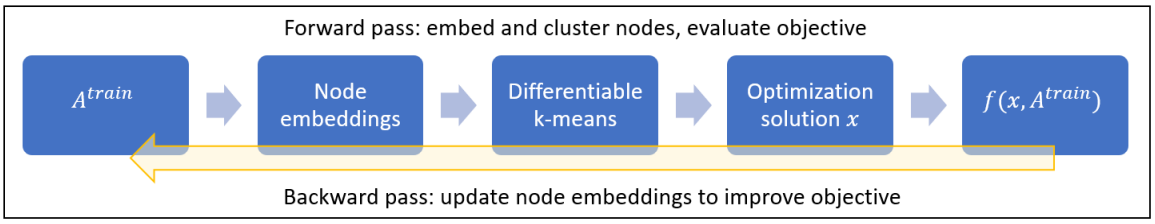
Graph optimization problems are ubiquitous across domains. For instance, many applications require clustering a graph into well-connected subgraphs; such techniques have been used to discover communities in social networks, mine knowledge graphs, and a variety of other tasks (Ahn et al., 2010). Other classic examples include maxcut (applied to computational biology (Snir & Rao, 2006) and circuit design (Barahona et al., 1988)), vertex cover, and so on. While there is a great deal of work on such tasks, complete applications must also solve difficult machine learning challenges. For instance, the input graph is usually incomplete: some edges may be unobserved, or nodes may have attributes that are only partially known. Recent work has introduced sophisticated methods for tasks such as link prediction and semi-supervised classification (Perozzi et al., 2014; Kipf & Welling, 2017; Schlichtkrull et al., 2018; Hamilton et al., 2017; Zhang & Chen, 2018), but these methods are developed in isolation of downstream optimization tasks.

Most current solutions use a two-stage approach which first trains a model using a standard loss, and then plugs the model's predictions into an optimization algorithm. However, predictions which minimize a standard loss function (e.g., cross-entropy) may be suboptimal for specific optimization tasks, especially in complex and noisy settings where even the best model is imperfect. Here, we develop methods which integrate graph learning and optimization, enabling end-to-end training. Our key technical contribution is a differentiable surrogate for common graph optimization problems that can then be combined end-to-end with differentiable learning models. This is nontrivial because the underlying problems are highly combinatorial, lacking well-defined gradients. A large body of work aims to replace combinatorial algorithms by purely using neural networks (Vinyals et al., 2015; Kool & Welling, 2018). While such systems can be trained to perform well on specific data distributions, they must discover algorithmic concepts entirely from scratch. We propose an approach that gets the best of both worlds by *incorporating algorithmic structure as a differentiable layer in the overall system*, which can then be fine-tuned by training parameters of the learned component.

We propose a specific form of algorithmic structure based on including a differentiable version of a clustering algorithm as a layer in the neural network. Clustering is motivated by the observation that state of the art deep learning models for graphs work by embedding the nodes of the graph into a continuous space. This suggests that we can approximate optimization over the *discrete* graph with an optimization problem in the *continuous* embedding space. This provides explicit algorithmic structure to the learned system: instead of having to learn the concept of grouping similar entities from scratch, the system simply must adjust the learned representations so that nodes which are clustered together via

[1]School of Engineering and Applied Sciences, Harvard University [2]Department of Computer Science, University of Southern California. Correspondence to: Bryan Wilder <bwilder@g.harvard.edu>.

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



their embeddings also, e.g., belong to the same communities of the graph (for the community detection domain).

In short, we make three contributions. First, we introduce a general framework for integrating graph learning and optimization by differentiable optimization in the continuous space as a proxy for the discrete problem. Second, we give a means of differentiating through the clustering procedure, allowing it to be used as a building block in deep learning systems. Third, we show experimentally that our approach improves over both two-stage baselines and approaches that do not include explicit algorithmic structure on a range of example datasets and optimization problems.

2. Setting

We consider settings that combine learning and optimization. The input is a graph $G = (V, E)$, which is in some way partially observed. We will formalize our problem in terms of link prediction as an example, but our framework applies to other common graph learning problems (e.g., semi-supervised classification). In link prediction, the graph is not entirely known; instead, we observe only training edges $E^{train} \subset E$. Let A denote the adjacency matrix of the graph and A^{train} denote the adjacency matrix with only the training edges. The learning task is to predict A from A^{train}. In domains we consider, the motivation for performing link prediction, is to solve a decision problem for which the objective depends on the full graph. Specifically, we have a decision variable x, objective function $f(x, A)$, and a feasible set $\mathcal{X}$. We aim to solve the optimization problem

$$\max_{x \in \mathcal{X}} f(x, A). \tag{1}$$

However, A is unobserved. The most common approach is to train a model to reconstruct A from A^{train} using a standard loss function (e.g., cross-entropy), producing an estimate $\hat{A}$. The *two-stage* approach plugs $\hat{A}$ into an optimization algorithm for Problem 1, maximizing $f(x, \hat{A})$.

We propose end-to-end models which map from A^{train} directly to a feasible decision x. The model will be trained to maximize $f(x, A^{train})$, i.e., the quality of its decision evaluated on the training data (instead of a loss $\ell(\hat{A}, A^{train})$ that measures purely predictive accuracy). One approach is to "learn away" the problem by training a standard model (e.g., a GCN) to map directly from A^{train} to x. However, this forces the model to entirely rediscover algorithmic concepts, while two-stage methods are able to exploit highly sophisticated, hand-tuned methods for optimization. We propose an alternative that embeds algorithmic structure into the learned model, getting the best of both worlds.

3. Approach: ClusterNet

Our proposed approach merges two differentiable components into a system that is trained end-to-end. First, a *graph embedding* layer which uses A^{train} and any node features to embed the nodes of the graph into $\mathbb{R}^p$. In our experiments, we use GCNs (Kipf & Welling, 2017). Second, a layer that performs *differentiable optimization*. This layer takes the continuous-space embeddings as input and uses them to produce a solution x to the graph optimization problem. Specifically, we propose to use a layer the implements a differentiable version of K-means clustering. This layer produces a soft assignment of the nodes to clusters, along with the cluster centers in embedding space. We remark that some recent work has trained deep representations together with a set of cluster centers (Xie et al., 2016; Yang et al., 2016); however, none of this work introduces an explicit clustering layer that can be incorporated into larger systems.

The intuition is that cluster assignments can be interpreted as the solution to many common graph optimization problems. For instance, in community detection we can interpret the cluster assignments as assigning the nodes to communities. Or, in maxcut, we can use two clusters to assign nodes to either side of the cut. Another example is maximum coverage and related problems, where we attempt to select a set of K nodes which cover (are neighbors to) as many other nodes as possible. This problem can be approximated by clustering the nodes into K components and choosing nodes whose embedding is close to the center of each cluster. We do not claim that any of these problems is exactly reducible to K-means. Rather, the idea is that including K-means as a layer in the network provides a useful inductive bias. This algorithmic structure can be fine-tuned to specific problems by training the first component, which produces the embeddings, so that the learned representations induce clusterings with high objective value for the underlying task.

3.1. Forward pass

Let x_j denote the embedding of node j and μ_k denote the center of cluster k. r_{jk} denotes the degree to which node j is assigned to cluster k. In traditional K-means, this is a binary quantity, but we will relax it to a fractional value such that $\sum_k r_{jk} = 1$ for all j. Specifically, we take $r_{jk} = \frac{\exp(-\beta||x_j - \mu_k||)}{\sum_\ell \exp(-\beta||x_j - \mu_\ell||)}$, which is a soft-min assignment of each point to the cluster centers based on distance. Here, β is a temperature hyperparameter; taking $\beta \to \infty$ recovers the standard k-means assignment. We can optimize the cluster centers via an iterative process analogous to the typical k-means updates by alternately setting

$$\mu_k = \frac{\sum_j r_{jk} x_j}{\sum_j r_{jk}} \ \forall k = 1...K \tag{2}$$
$$r_{jk} = \frac{\exp(-\beta||x_j - \mu_k||)}{\sum_\ell \exp(-\beta||x_j - \mu_\ell||)} \ \forall k = 1...K, j = 1...n.$$

These iterates converge to a fixed point where μ remains the same between successive updates (MacKay, 2003).

3.2. Backward pass

We will use the implicit function theorem to analytically differentiate through the fixed point that the forward pass iterates converge to, obtaining expressions for $\frac{\partial \mu}{\partial x}$ and $\frac{\partial r}{\partial x}$. This in turn allows us to backpropgate gradients from the loss function to the component that produced the embeddings x. Define a function $f : \mathbb{R}^{Kp} \to \mathbb{R}$ as

$$f_{i,\ell}(\mu, x) = \mu_i^\ell - \frac{\sum_j r_{jk} x_j}{\sum_j r_{jk}} \tag{3}$$

Now, (μ, x) are a fixed point of the iterates if $f(\mu, x) = \mathbf{0}$. Applying the implicit function theorem yields that

$$\frac{\partial \mu}{\partial x} = -\left[\frac{\partial f(\mu, x)}{\partial \mu}\right]^{-1} \frac{\partial f(\mu, x)}{\partial x}. \tag{4}$$

Note that after obtaining $\frac{\partial \mu}{\partial x}$, we can calculate $\frac{\partial r}{\partial x}$ as

$$\frac{\partial r(\mu, x)}{\partial x} = \frac{\partial r(\mu, x)}{\partial \mu}\frac{\partial \mu}{\partial x} + \frac{\partial r(\mu, x)}{\partial x}.$$

3.2.1. Exact backward pass

We now turn to calculating $\frac{\partial \mu}{\partial x}$. Define $R_i = \sum_{j=1}^n r_{ji}$ and $C_i = \sum_{j=1}^n r_{ji} x_j$. We will work with $C_i \in \mathbb{R}^{p \times 1}$ as a column vector. For a fixed i, j, we have

$$\frac{\partial f_{i,\cdot}}{\partial x_j} = -\frac{R_i x_j \left[\frac{\partial r_{ji}}{\partial x_j}\right]^\top - C_i \left[\frac{\partial r_{ji}}{\partial x_j}\right]^\top}{R_i^2} - \frac{r_{ji}}{R_i} I$$

where I denotes the p-dimensional identity matrix. Similarly, fixing i, k gives

$$\frac{\partial f_{i,\cdot}}{\partial \mu_k} = \delta_{ik} I - \frac{R_i \sum_{j=1}^n x_j \left[\frac{\partial r_{ji}}{\partial \mu_k}\right]^\top - C_i \left[\sum_{j=1}^n \frac{\partial r_{ji}}{\partial \mu_k}\right]^\top}{R_i^2}$$

where δ_{ik} is the indicator function. Now, one option to obtain $\frac{\partial \mu}{\partial x}$ is to explicitly calculate the above expressions (across all i, j, k) and then evaluate Equation 4. This can be done either explicitly, or by using iterative methods to compute the backward pass vector without explicitly inverting $\frac{\partial f}{\partial \mu}$. In either case, though, the procedure is potentially costly in computational terms: $\frac{\partial f_i}{\partial \mu_k}$ takes $O(np)$ time to compute, for $O(K^2 np)$ time overall to obtain $\frac{\partial f}{\partial \mu}$. Then, inverting or factoring $\frac{\partial f}{\partial \mu}$ will likely require $O(K^3 p^3)$ time, since it is a matrix of size $(Kp) \times (Kp)$. While the exact backward pass may be feasible for some problems, it quickly becomes burdensome for large instances. We now propose a fast approximation to the exact backward pass.

3.2.2. Approximate backward pass

We start from the observation that $\frac{\partial f}{\partial \mu}$ will often be dominated by its diagonal terms (the identity matrix). The off-diagonal entries capture the extent to which updates to one entry of μ indirectly impact other entries via changes to the cluster assignments r. However, when the cluster assignments are relatively firm, r will not be highly sensitive to small changes to the cluster centers. Substantial off-diagonal entries of $\frac{\partial f}{\partial \mu}$ reflect cases where r is sufficiently sensitive that changes to one cluster center cause large enough changes to the overall cluster assignments that the centers of other clusters themselves move. We find this case to be relatively rare empirically, especially since the optimal choice of the parameter β (which controls the hardness of the cluster assignments) is typically fairly high. Under these conditions, we can approximate $\frac{\partial f}{\partial \mu}$ by its diagonal, $\frac{\partial f}{\partial \mu} \approx I$. This in turn gives $\frac{\partial \mu}{\partial x} \approx -\frac{\partial f}{\partial x}$.

We now show that this approximate gradient can be calculated by unrolling a single iteration of the forward-pass updates from Equation 2 at convergence. Examining the definition of $f(\mu, x)$ in Equation 3, we see that the first term (μ_i^ℓ) is in fact constant with respect to x, since here μ a fixed value input to f. Hence,

$$-\frac{\partial f_k}{\partial x} = \frac{\partial}{\partial x}\frac{\sum_j r_{jk} x_j}{\sum_j r_{jk}}$$

which is just the update equation for μ_k. Since the forward-pass updates are written entirely in terms of differentiable functions, we can automatically compute the approximate backward pass with respect to x (i.e., compute products with our approximations to $\frac{\partial \mu}{\partial x}$ and $\frac{\partial r}{\partial x}$) by applying standard

Table 1. Objective value of each method on the full graph.

	Community detection			Maxcut		
	cora	citeseer	pubmed	cora	citeseer	pubmed
ClusterNet	**0.574**	**0.617**	**0.629**	**6631**	**6188**	54181
GCN-2stage	0.268	0.324	-	5218	3462	-
GCN-e2e	0.090	0.221	0.011	2812	273	1538

autodifferentiation tools to the final update of the forward pass. Compared to computing the exact analytical gradients, this avoids the need to explicitly reason about or invert $\frac{\partial f}{\partial \mu}$.

Compared to differentiating by unrolling the entire sequence of updates in the computational graph (as has been suggested for other problems (Domke, 2012; Andrychowicz et al., 2016; Zheng et al., 2015)), our approach has two key advantages. First, it avoids storing the entire history of updates and backpropagating through all of them. The runtime for our approximation is independent of the number of updates needed to reach convergence. Second, we can in fact use entirely non-differentiable operations to arrive at the fixed point, e.g., heuristics for the K-means problem, stochastic methods which only examine subsets of the data, etc. This allows the forward pass to scale to larger datasets since we can use the best algorithmic tools available, not just those that can be explicitly encoded in the autodifferentiation tool's computational graph.

4. Experimental results

We now show experiments on domains that combine link prediction with two example optimization problems.

Learning problem: In link prediction, we observe a partial graph and aim to infer which unobserved edges are present. In each of the experiments, we hold out 50% of the edges in the graph (uniformly at random) as the unobserved test edges. The remaining edges are split into 40% training and 10% validation. The learning task is to use the training edges to predict whether the test edges are present, after which we will solve an optimization problem on the predicted graph. The objective is to find a solution with high objective value measured on the *entire* graph, not just the training edges.

Optimization problems: We consider two example optimization tasks. First, *community detection* aims to partition the nodes of the graph into K distinct subgroups which are dense internally, but with few edges across groups. Formally, the objective is to find a partition maximizing the modularity (Newman, 2006b), defined as

$$\frac{1}{2m} \sum_{u,v \in V} \sum_{k=1}^{K} \left[A_{uv} - \frac{d_u d_v}{2m} \right] r_{uk} r_{vk} \tag{5}$$

where d_v is the degree of node v, and r_{vk} is 1 if node v is assigned to community k and zero otherwise. This measures the number of edges within communities compared to the number which would be expected if edges where placed randomly. To model this problem in our framework, our clustering module contains a cluster for each of the K communities, and we can directly interpret the r that it outputs as a soft assignment of nodes to communities. Defining B to be the modularity matrix with entries $B_{uv} = A_{uv} - \frac{d_u d_v}{2m}$, our training objective is $\frac{1}{2m} \mathrm{Tr}\left[r^\top B^{train} r\right]$, which is the expected number of training edges that fall within communities when each node j is assigned independently to a community with probabilities given by r_j.

Second, *maxcut* is a classic graph optimization problem which attempts to find a cut (a partition of the nodes into two groups) which maximizes the number of edges crossing between sides of the cut. Here, our clustering module has two clusters (representing the sides of the cut) and we interpret r as soft assignments to the two sides. The training objective is $\left|\left|(1 - rr^\top) \odot A^{train}\right|\right|_1$, where $||\cdot||_1$ denotes summing the elements of the matrix. This is the expected number of edges cut when each node j is assigned independently to one side with probabilities given by r_j.

Algorithms: We instantiate our approach using a 2-layer GCN to produce the node embeddings, followed by a clustering layer. We compare to two baselines. *First*, GCN-2stage, the two stage approach which first trains a model for link prediction, and then inputs the predicted graph into an optimization algorithm. For link prediction, we use the GCN-based system of (Schlichtkrull et al., 2018) (we also adopt their training procedure, including negative sampling and edge dropout). For the optimization algorithms, we use standard approaches for each domain. Specifically, for community detection we use the spectral approach of (Newman, 2006a), while for maxcut we use local search (a well-known approximation algorithm (Kleinberg & Tardos, 2006)). *Second*, GCN-e2e, an end-to-end approach which does not include explicit algorithm structure. We train a GCN-based network to directly predict the r_j (i.e., the nodes' community or cut assignments), using the same training objectives as our own model. The number of layers in the GCN was chosen using the validation set; empirically, we observed the best performance with 4 layers. This baseline allows us to isolate the benefits of including algorithmic structure.

Results: We show results on three standard graph datasets: cora, citeseer and pubmed. Table 1 shows the objective value

that each method achieves for each of the optimization tasks on each graph. We note that the two-stage baseline could not be run on the pubmed dataset since it required predicting a dense $n \times n$ matrix which did not fit in memory. We find that ClusterNet substantially improves on both baselines across all settings, demonstrating the advantage to of end-to-end training which incorporates algorithmic structure.

References

Ahn, Y., Bagrow, J., and Lehmann, S. Link communities reveal multiscale complexity in networks. *Nature*, 466 (7307):761, 2010.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.

Barahona, F., Grötschel, M., Jünger, M., and Reinelt, G. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988.

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

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

Kipf, T. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Kleinberg, J. and Tardos, E. *Algorithm design*. Pearson Education India, 2006.

Kool, W. and Welling, M. Attention solves your TSP. *arXiv preprint arXiv:1803.08475*, 2018.

MacKay, D. J. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Newman, M. E. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74 (3):036104, 2006a.

Newman, M. E. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006b.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

Schlichtkrull, M., Kipf, T., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, 2018.

Snir, S. and Rao, S. Using max cut to enhance rooted trees consistency. *IEEE/ACM transactions on computational biology and bioinformatics*, 3(4):323–333, 2006.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *NIPS*, 2015.

Xie, J., Girshick, R., and Farhadi, A. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pp. 478–487, 2016.

Yang, L., Cao, X., He, D., Wang, C., Wang, X., and Zhang, W. Modularity based community detection with deep learning. In *IJCAI*, volume 16, pp. 2252–2258, 2016.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. In *NIPS*, 2018.

Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., and Torr, P. H. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 1529–1537, 2015.