Combining learning and optimization on graphs

Anonymous

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 facility location, maxcut, 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. Standard approaches treat learning and optimization entirely separately, while recent machine learning work aims to predict the optimal solution directly from the inputs. Here, we propose an alternative *decision-focused learning* approach that integrates a differentiable proxy for common graph optimization problems as a layer in learned systems. The main idea is to learn a representation that maps the original optimization problem onto a simpler proxy problem that can be efficiently differentiated through. Experimental results show that our CLUSTERNET system outperforms both pure end-to-end approaches (that directly predict the optimal solution) and standard approaches that entirely separate learning and optimization.

An extended version of this paper appears at NeurIPS 2019.

Introduction

While deep learning has proven enormously successful at a range of tasks, an expanding area of interest concerns systems that can flexibly combine learning with optimization. Examples include recent attempts to solve combinatorial optimization problems using neural architectures (Vinyals, Fortunato, and Jaitly 2015; Khalil et al. 2017; Bello et al. 2016; Kool, van Hoof, and Welling 2019), as well as work which incorporates explicit optimization algorithms into larger differentiable systems (Amos and Kolter 2017; Donti, Amos, and Kolter 2017; Wilder, Dilkina, and Tambe 2019). The ability to combine learning and optimization promises improved performance for real-world problems which require decisions to be made on the basis of machine learning predictions by enabling end-to-end training which focuses the learned model on the decision problem at hand.

We focus on graph optimization problems, an expansive subclass of combinatorial optimization. While graph optimization is ubiquitous across domains, complete applications must also solve 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, Al-Rfou, and Skiena 2014; Kipf and Welling 2017; Schlichtkrull et al. 2018; Hamilton, Ying, and Leskovec 2017; Zhang and 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 (Yan and Gregory 2012; Burgess, Adar, and Cafarella 2016; Bahulkar et al. 2018; Berlusconi et al. 2016; Tan et al. 2016). However, predictions which minimize a standard loss function (e.g., cross-entropy) may be suboptimal for specific optimization tasks, especially in difficult settings where even the best model is imperfect.

A preferable approach is to incorporate the downstream optimization problem into the training of the machine learning model. Many recent works take a pure end-to-end approach where a neural network is trained to predict a solution to the optimization problem using supervised or reinforcement learning (Vinyals, Fortunato, and Jaitly 2015; Khalil et al. 2017; Bello et al. 2016; Kool, van Hoof, and Welling 2019). However, this often requires a large amount of data and results in suboptimal performance because the network needs to discover algorithmic structure entirely from scratch. Between the extremes of a two stage approach and pure end-to-end architectures, *decision-focused learning* (Donti, Amos, and Kolter 2017; Wilder, Dilkina, and Tambe 2019) embeds a solver for the optimization problem as a differentiable layer within a learned system. This allows the model to train using the downstream performance that it induces as the loss, while leveraging prior algorithmic knowledge for optimization. The downside is that this approach requires manual effort to develop a differentiable solver for each problem and often results in cumbersome systems that must, e.g, call a QP solver every forward pass.

We propose a new approach that gets the best of both worlds: incorporate a solver for a simpler optimization problem as a differentiable layer, and then learn a representation that maps the (harder) problem of interest onto an instance of the simpler problem. Compared to earlier approaches to decision-focused learning, this places more emphasis on representation learning and simplifies the optimization component. However, compared to pure end-to-end approaches, we only need to learn the reduction to the simpler problem instead of the entire algorithm.

Her, we instantiate the simpler problem as a differentiable version of k-means clustering. Clustering is motivated by the fact that graph neural networks embed nodes into a continuous space, allowing us to approximate optimization over the graph with optimization in continuous embedding space. We then interpret the cluster assignments as a solution to the discrete problem. We instantiate this approach for two classes of optimization problems: those that require *partitioning* the graph (e.g., community detection or maxcut), and those that require *selecting a subset of K nodes* (facility location, influence maximization, immunization, etc). We don't claim that clustering is the right structure for all tasks, but it is sufficient for many problems as shown in this paper.

Related work

There recent interest in training neural networks to solve combinatorial optimization problems (Vinyals, Fortunato, and Jaitly 2015; Khalil et al. 2017; Bello et al. 2016; Kool, van Hoof, and Welling 2019). While we focus mostly on combining graph learning with optimization, our model can also be trained just to solve an optimization problem given complete information about the input. The main methodological difference is that we include more structure via a differentiable k-means layer instead of using more generic tools (e.g., feed-forward or attention layers).

Other work uses deep architectures as a part of a clustering algorithm (Tian et al. 2014; Law, Urtasun, and Zemel 2017; Guo et al. 2017; Shaham et al. 2018; Nazi et al. 2019), or includes a clustering step as a component of a deep network (Greff et al. 2016; Greff, van Steenkiste, and Schmidhuber 2017; Ying et al. 2018). While some techniques are similar, the overall task we address and framework we propose are entirely distinct. Our aim is not to cluster a Euclidean dataset (as in (Tian et al. 2014; Law, Urtasun, and Zemel 2017; Guo et al. 2017; Shaham et al. 2018)), or to solve perceptual grouping problems (as in (Greff et al. 2016; Greff, van Steenkiste, and Schmidhuber 2017)). Rather, we propose an approach for graph optimization problems. There is also some work which uses deep networks for graph clustering (Xie, Girshick, and Farhadi 2016; Yang et al. 2016). However, none of this work consider an explicit clustering algorithm in the network, or our goal of integrating graph learning and optimization.

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 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 rediscover algorithmic concepts, while two-stage methods exploit sophisticated optimization methods. We propose an alternative that embeds algorithmic structure into the learned model, getting the best of both worlds.

Approach: ClusterNet

Our proposed ClusterNet system 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 and 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 that 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.

The intuition is that cluster assignments can be interpreted as the solution to common graph optimization problems. For instance, in community detection we can interpret the cluster assignments as assigning the nodes to communities. 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. The first component, which produces the embeddings, is trained so that the learned representations induce clusterings with high objective value for the downstream optimization task.

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. While our architecture can be used with any norm $|| \cdot ||$, we use the negative cosine similarity due to its strong empirical performance. β is an inverse-temperature hyperparameter. We can optimize the cluster centers via an iterative process analogous to the typical k-means updates by alternately setting

$$\mu_k = \frac{\sum_j r_{jk} x_j}{\sum_j r_{jk}} \quad r_{jk} = \frac{\exp(-\beta||x_j - \mu_k||)}{\sum_\ell \exp(-\beta||x_j - \mu_\ell||)} \tag{2}$$

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

Backward pass

We will use the implicit function theorem to analytically differentiate through the fixed point that the forward pass k-means iterates converge to, obtaining expressions for $\frac{\partial \mu}{\partial x}$ and $\frac{\partial r}{\partial x}$. Previous work (Donti, Amos, and Kolter 2017; Wilder, Dilkina, and Tambe 2019) has used the implicit function theorem to differentiate through the KKT conditions of optimization problems; here we take a more direct approach that characterizes the update process itself. Doing so allows us to backpropagate gradients from the decision loss to the component that produced the embeddings x. Define a function $f : \mathbb{R}^{Kp} \to \mathbb{R}$ as

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

Now, (μ, x) are a fixed point of the iterates if $f(\mu, x) = \mathbf{0}$. Applying the implicit function theorem yields that $\frac{\partial \mu}{\partial x} = -\left[\frac{\partial f(\mu,x)}{\partial \mu}\right]^{-1} \frac{\partial f(\mu,x)}{\partial x}$, from which $\frac{\partial r}{\partial x}$ can be easily obtained via the chain rule.

Exact backward pass: We now examine the process of calculating $\frac{\partial \mu}{\partial x}$. Both $\frac{\partial f(\mu,x)}{\partial x}$ and $\frac{\partial f(\mu,x)}{\partial \mu}$ can be easily calculated in closed form (see appendix). Computing the former requires time $O(nKp^2)$. Computing the latter requires $O(npK^2)$ time, after which it must be inverted (or else iterative methods must be used to compute the product with its inverse). This requires time $O(K^3p^3)$ 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.

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. We find to be typical 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 can formally justify this approximation when the clusters are relatively balanced and well-separated. More precisely, define $c(j) = \arg\max_i r_{ji}$ to be the closest cluster to point j. Proposition 1 (proved in the appendix) shows that the quality of the diagonal approximation improves exponentially quickly in the product of two terms: β, the hardness of the cluster assignments, and δ, which measures how well separated the clusters are. α (defined below) measures the balance of the cluster sizes. We assume for convenience that the input is scaled so $||x_j||_1 \leq 1 \forall j$.

Proposition 1. *Suppose that for all points j, $||x_j - \mu_i|| - ||x_j - \mu_{c(j)}|| \geq \delta$ for all $i \neq c(j)$ and that for all clusters i, $\sum_{j=1}^n r_{ji} \geq \alpha n$. Moreover, suppose that $\beta\delta > \log \frac{2\beta K^2}{\alpha}$. Then, $\left|\left|\frac{\partial f}{\partial \mu} - I\right|\right|_1 \leq \exp(-\delta\beta) \left(\frac{K^2\beta}{\frac{1}{2}\alpha - K^2\beta\exp(-\delta\beta)}\right)$ where $|| \cdot ||_1$ is the operator 1-norm.*

We now show that the approximate gradient obtained by taking $\frac{\partial f}{\partial \mu} = I$ can be calculated by unrolling a single iteration of the forward-pass updates from Equation 2 at convergence. Examining Equation 3, we see that the first term (μ_i^ℓ) is constant with respect to x, since here μ is a fixed value. 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 by applying standard 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 $\frac{\partial f}{\partial \mu}$. The final iteration (the one which is differentiated through) requires time $O(npK)$, *linear* in the size of the data.

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. 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.

Obtaining solutions to the optimization problem

Having obtained the cluster assignments r, along with the centers μ, in a differentiable manner, we need a way to **(1)** differentiably interpret the clustering as a soft solution to the optimization problem, **(2)** differentiate a relaxation of the objective value of the graph optimization problem in terms of that solution, and then **(3)** round to a discrete solution at test time. We give a generic means of accomplishing these three steps for two broad classes of problems: those that involve *partitioning the graph into K disjoint components*, and those that that involve *selecting a subset of K nodes*.

Partitioning: **(1)** We can naturally interpret the cluster assignments r as a soft partitioning of the graph. **(2)** One generic continuous objective function (defined on soft partitions) follows from the random process of assigning each node j to a partition with probabilities given by r_j, repeating this process independently across all nodes. This gives the expected training decision loss $\ell = \mathbb{E}_{r^{hard} \sim r}[f(r^{hard}, A^{train})]$, where $r^{hard} \sim r$ denotes this random assignment. ℓ is differentiable wrt r, and can be computed in closed form via standard autodifferentiation tools for many problems of interest (see below). **(3)** At test time, we simply apply a hard maximum to r.

Subset selection: **(1)** Here, it is less obvious how to obtain a subset of K nodes from the cluster assignments. Our continuous solution will be a vector x, $0 \leq x \leq 1$, where $||x||_1 = K$. Intuitively, x_j is the probability of including x_j in the solution. Our approach obtains x_j by placing greater probability mass on nodes that are near the cluster centers. Specifically, each center μ_i is endowed with one unit of probability mass, which it allocates to the points x as $a_{ij} = \text{softmin}(\eta||x - \mu_i||)_j$. The total probability allocated to node j is $b_j = \sum_{i=1}^{K} a_{ij}$. Since we may have $b_j > 1$, we pass b through a sigmoid function to cap the entries at 1; specifically, we take $x = 2 * \sigma(\gamma b) - 0.5$ where γ is a tunable parameter. If the resulting x exceeds the budget constraint ($||x||_1 > K$), we output $\frac{Kx}{||x||_1}$.

(2) We interpret this solution in terms of the objective similarly as above. Specifically, we consider drawing a discrete solution $x^{hard} \sim x$ where every node j is included (i.e., set to 1) independently with probability x_j from the end of step **(1)**. The training objective is then $\mathbb{E}_{x^{hard} \sim x}[f(x^{hard}, A^{train})]$. Often, this can again be computed and differentiated through in closed form (see below).

(3) At test time, we need a feasible discrete vector x; note that independently rounding the individual entries may produce a vector with more than K ones. Here, we apply a generic approach based on pipage rounding (Ageev and Sviridenko 2004), a randomized rounding scheme which has been applied to many problems. Pipage rounding can be implemented to produce a random feasible solution in time $O(n)$ (Karimi et al. 2017); in practice we round several times and take the solution with the best decision loss on the observed edges. While pipage rounding has theoretical guarantees only for specific classes of functions, we find it more broadly successful (e.g., facility location). However, more domain-specific rounding methods can also be applied.

Experimental results

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

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 60% of the edges in the graph, with 40% observed during training. We used a graph dataset which is not included in our results to set our method's hyperparameters, which were kept constant across datasets (see appendix for details). The learning task is to use the training edges to predict whether the remaining 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 optimization tasks, one from each of the broad classes introduced above. 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 $\frac{1}{2m}\sum_{u,v \in V}\sum_{k=1}^{K}\left[A_{uv} - \frac{d_u d_v}{2m}\right] r_{uk} r_{vk}$. Here, 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 expected number if edges were placed randomly. Our clustering module has one cluster for each of the K communities. Defining B to be the modularity matrix with entries $B_{uv} = A_{uv} - \frac{d_u d_v}{2m}$, our training objective (the expected value of a partition sampled according to r) is $\frac{1}{2m}\text{Tr}\left[r^\top B^{train} r\right]$.

Second, minmax *facility location*, where the decision problem is to select a subset of K nodes from the graph, minimizing the maximum distance from any node to a facility (selected node). Letting $d(v, S)$ be the shortest path length from a vertex v to a set of vertices S, the objective is $f(S) = \min_{|S| \leq k} \max_{v \in V} d(v, S)$. To obtain the training loss, we take two steps. First, we replace $d(v, S)$ by $\mathbb{E}_{S \sim x}[d(v, S)]$, where $S \sim x$ denotes drawing a set from the product distribution with marginals x. This can easily be calculated in closed form (Karimi et al. 2017). Second, we replace the $\min$ with a softmin.

Baseline learning methods: We instantiate CLUSTERNET using a 2-layer GCN for node embeddings, followed by a clustering layer. We compare to three families of 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, outlined below. *Second*, "train", which runs each optimization algorithm only on the observed training subgraph (without attempting any link prediction). *Third*, GCN-e2e, an end-to-end approach which does not include explicit algorithm structure. We train a GCN-based network to directly predict the final decision variable (r or x) using the same training objectives as our own model. Empirically, we observed best performance with a 2-layer GCN. This baseline allows us to isolate the benefits of including algorithmic structure.

Table 1: Performance on the community detection task

	Learning + optimization					Optimization				
	cora	cite.	prot.	adol	fb	cora	cite.	prot.	adol	fb
ClusterNet	**0.54**	**0.55**	**0.29**	**0.49**	**0.30**	**0.72**	**0.73**	**0.52**	**0.58**	0.76
GCN-e2e	0.16	0.02	0.13	0.12	0.13	0.19	0.03	0.16	0.20	0.23
Train-CNM	0.20	0.42	0.09	0.01	0.14	0.08	0.34	0.05	0.57	**0.77**
Train-Newman	0.09	0.15	0.15	0.15	0.08	0.20	0.23	0.29	0.30	0.55
Train-SC	0.03	0.02	0.03	0.23	0.19	0.09	0.05	0.06	0.49	0.61
GCN-2stage-CNM	0.17	0.21	0.18	0.28	0.13	-	-	-	-	-
GCN-2stage-Newman	0.00	0.00	0.00	0.14	0.02	-	-	-	-	-
GCN-2stage-SC	0.14	0.16	0.04	0.31	0.25	-	-	-	-	-

Table 2: Performance on the facility location task.

	Learning + optimization					Optimization				
	cora	cite.	prot.	adol	fb	cora	cite.	prot.	adol	fb
ClusterNet	**10**	**14**	**6**	**6**	**4**	**9**	**14**	**6**	**5**	**3**
GCN-e2e	12	15	8	**6**	5	11	**14**	7	6	5
Train-greedy	14	16	8	8	6	**9**	**14**	7	6	5
Train-gonzalez	12	17	8	**6**	6	10	15	7	7	**3**
GCN-2Stage-greedy	14	17	8	7	6	-	-	-	-	-
GCN-2Stage-gonzalez	13	17	8	**6**	6	-	-	-	-	-

Baseline optimization approaches: In each domain, we compare to expert-designed optimization algorithms found in the literature. In community detection, we compare to "CNM" (Clauset, Newman, and Moore 2004), an agglomerative approach, "Newman", an approach that recursively partitions the graph (Newman 2006a), and "SC", which performs spectral clustering (Von Luxburg 2007) on the modularity matrix. In facility location, we compare to "greedy", the common heuristic of iteratively selecting the point with greatest marginal improvement in objective value, and "gonzalez" (Gonzalez 1985), an algorithm which iteratively selects the node furthest from the current set. "gonzalez" attains the optimal 2-approximation for this problem.

Datasets: We use several standard graph datasets: cora (Sen et al. 2008) (a citation network with 2,708 nodes), citeseer (Sen et al. 2008) (a citation network with 3,327 nodes), protein (Collection 2017c) (a protein interaction network with 3,133 nodes), adol (Collection 2017a) (an adolescent social network with 2,539 vertices), and fb (Collection 2017b; Leskovec and Mcauley 2012) (an online social network with 2,888 nodes). For facility location, we use the largest connected component of the graph (since otherwise distances may be infinite). Cora and citeseer have node features (based on a bag-of-words representation of the document), which were given to all GCN-based methods. For the other datasets, we generated unsupervised node2vec features (Grover and Leskovec 2016) using the training edges.

Results: We start out with results for the combined link prediction and optimization problem. Table 1 shows the objective value obtained by each approach on the full graph for community detection, with Table 2 showing facility location. We focus first on the "Learning + Optimization" column which shows the combined link prediction/optimization task. We use $K = 5$ clusters; $K = 10$ is very similar and may be found in the appendix. CLUSTERNET outperforms the baselines in nearly all cases, often substantially. GCN-e2e learns to produce nontrivial solutions, often rivaling the other baseline methods. However, the explicit structure used by our approach CLUSTERNET results in much higher performance.

We next examine an optimization-only task where the entire graph is available as input (the "Optimization" column of Tables 1 and Table 2). This tests CLUSTERNET's ability to learn to solve combinatorial optimization problems compared to expert-designed algorithms, even when there is no partial information or learning problem in play. We find that CLUSTERNET is highly competitive, meeting and frequently exceeding the baselines. It is particularly effective for community detection, where we observe large ($>$ 3x) improvements compared to the best baseline on some datasets. At facility location, our method always at least ties the baselines, and frequently improves on them. These experiments provide evidence that our approach, which is automatically specialized during training to optimize on a given graph, can rival and exceed hand-designed algorithms from the literature. The alternate learning approach, GCN-e2e, which is an end-to-end approach that tries to learn to predicts optimization solutions directly from the node features, at best ties the baselines and typically underperforms. This underscores the benefit of including algorithmic structure as a part of the end-to-end architecture.

References

Ageev, A. A., and Sviridenko, M. I. 2004. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization* 8(3):307–328.

Ahmed, N. K.; Rossi, R.; Lee, J. B.; Willke, T. L.; Zhou, R.; Kong, X.; and Eldardiry, H. 2018. Learning role-based graph embeddings. *arXiv preprint arXiv:1802.02896*.

Amos, B., and Kolter, J. Z. 2017. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*.

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

Bahulkar, A.; Szymanski, B. K.; Baycik, N. O.; and Sharkey, T. C. 2018. Community detection with edge augmentation in criminal networks. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Berlusconi, G.; Calderoni, F.; Parolini, N.; Verani, M.; and Piccardi, C. 2016. Link prediction in criminal networks: A tool for criminal intelligence analysis. *PloS one* 11(4):e0154244.

Burgess, M.; Adar, E.; and Cafarella, M. 2016. Link-prediction enhanced consensus clustering for complex networks. *PloS one* 11(5):e0153384.

Clauset, A.; Newman, M. E.; and Moore, C. 2004. Finding community structure in very large networks. *Physical review E* 70(6):066111.

Collection, K. N. 2017a. Adolescent health. http://konect.uni-koblenz.de/networks/moreno_health.

Collection, K. N. 2017b. Facebook (nips). http://konect.uni-koblenz.de/networks/ego-facebook.

Collection, K. N. 2017c. Human protein (vidal). http://konect.uni-koblenz.de/networks/maayan-vidal.

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

Donti, P.; Amos, B.; and Kolter, J. Z. 2017. Task-based end-to-end model learning in stochastic optimization. In *Advances in Neural Information Processing Systems*, 5484–5494.

Gonzalez, T. F. 1985. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science* 38:293–306.

Greff, K.; Rasmus, A.; Berglund, M.; Hao, T.; Valpola, H.; and Schmidhuber, J. 2016. Tagger: Deep unsupervised perceptual grouping. In *NeurIPS*.

Greff, K.; van Steenkiste, S.; and Schmidhuber, J. 2017. Neural expectation maximization. In *NeurIPS*.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 855–864. ACM.

Guo, X.; Gao, L.; Liu, X.; and Yin, J. 2017. Improved deep embedded clustering with local structure preservation. In *IJCAI*.

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

Karimi, M.; Lucic, M.; Hassani, H.; and Krause, A. 2017. Stochastic submodular maximization: The case of coverage functions. In *Advances in Neural Information Processing Systems*.

Khalil, E.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. In *NIPS*.

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

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, learn to solve routing problems! In *ICLR*.

Law, M. T.; Urtasun, R.; and Zemel, R. S. 2017. Deep spectral clustering learning. In *ICML*.

Leskovec, J., and Mcauley, J. J. 2012. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, 539–547.

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

Nazi, A.; Hang, W.; Goldie, A.; Ravi, S.; and Mirhoseini, A. 2019. Gap: Generalizable approximate graph partitioning framework. *arXiv preprint arXiv:1903.00614*.

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

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

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

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

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine* 29(3):93–93.

Shaham, U.; Stanton, K.; Li, H.; Nadler, B.; Basri, R.; and Kluger, Y. 2018. Spectralnet: Spectral clustering using deep neural networks. In *ICLR*.

Tan, S.-Y.; Wu, J.; Lü, L.; Li, M.-J.; and Lu, X. 2016. Efficient network disintegration under incomplete information: the comic effect of link prediction. *Scientific reports* 6:22916.

Tian, F.; Gao, B.; Cui, Q.; Chen, E.; and Liu, T.-Y. 2014. Learning deep representations for graph clustering. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Titsias, M. 2016. One-vs-each approximation to softmax for scalable estimation of probabilities. In *NeurIPS*.

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

Von Luxburg, U. 2007. A tutorial on spectral clustering. *Statistics and computing* 17(4):395–416.

Wilder, B.; Dilkina, B.; and Tambe, M. 2019. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *AAAI*.

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

Yan, B., and Gregory, S. 2012. Detecting community structure in networks using edge prediction methods. *Journal of Statistical Mechanics: Theory and Experiment* 2012(09):P09008.

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

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, 4800–4810.

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

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

Supplemental material: proofs

Exact expression for gradients

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

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

Guarantee for approximate gradients

Proposition 2. *Suppose that for all points j, $||x_j - \mu_i|| - ||x_j - \mu_{c(j)}|| \geq \delta$ for all $i \neq c(j)$ and that for all clusters i, $\sum_{j=1}^n r_{ji} \geq \alpha n$. Moreover, suppose that $\beta\delta > \log \frac{2\beta K^2}{\alpha}$. Then, $\left|\left|\frac{\partial f}{\partial \mu} - I\right|\right|_1 \leq \exp(-\delta\beta)\left(\frac{K^2\beta}{\frac{1}{2}\alpha - K^2\beta\exp(-\delta\beta)}\right)$ where $||\cdot||_1$ is the operator 1-norm.*

We focus on the off-diagonal component of $\frac{\partial f_{im}}{\partial \mu_{k\ell}}$, given by

$$A_{(i,m),(k,\ell)} = -\frac{R_i \sum_{j=1}^n x_j^m \left[\frac{\partial r_{ji}}{\partial \mu_k^\ell}\right] - C_i^m \left[\sum_{j=1}^n \frac{\partial r_{ji}}{\partial \mu_k^\ell}\right]}{R_i^2}$$

The key term here is $\frac{\partial r_{ji}}{\partial \mu_k^\ell}$. Let $s_{ji} = -\beta||x_j - \mu_i||$ Since r is defined via the softmax function, we have

$$\frac{\partial r_{ji}}{\partial \mu_k^\ell} = \frac{\partial r_{ji}}{\partial s_{jk}} \frac{\partial s_{jk}}{\partial \mu_k^\ell}$$

where

$$\frac{\partial r_{ji}}{\partial s_{jk}} = \begin{cases} r_{ji}(1 - r_{ji}) & \text{if i = k} \\ -r_{ji} r_{jk} & \text{otherwise.} \end{cases}$$

Note now via Lemma 1, in both cases we have that

$$\left|\frac{\partial r_{ji}}{\partial s_{jk}}\right| \leq K \exp(-\beta\delta)$$

Define $\epsilon = K\exp(-\beta\delta)$ and note that we have that $\left|\frac{\partial s_{jk}}{\partial \mu_k^\ell}\right| \leq \beta$, since we defined s in terms of cosine similarity and have assumed that the input is normalized. Putting this together, we have

$$\begin{aligned} \left|A_{(i,m),(k,\ell)}\right| &\leq \frac{\sum_{j=1}^n x_j^m \epsilon\beta}{R_i} + \frac{C_i^m n\epsilon\beta}{R_i^2} \\ &\leq \frac{\epsilon\beta\sum_{j=1}^n x_j^m}{\alpha n} + \frac{\mu_i^m n\epsilon\beta}{R_i} \\ &\leq \frac{\epsilon\beta\sum_{j=1}^n x_j^m}{\alpha n} + \frac{\mu_i^m \epsilon\beta}{\alpha} \end{aligned}$$

and so

$$\begin{aligned} ||A||_1 &= \max_{(k,\ell)} \sum_{(i,m)} A_{(i,m),(k,\ell)} \\ &\leq \max_{(k,\ell)} \sum_{(i,m)} \frac{\epsilon\beta\sum_{j=1}^n x_j^m}{\alpha n} + \frac{\mu_i^m \epsilon\beta}{\alpha} \\ &\leq \max_{(k,\ell)} \sum_i \frac{\epsilon\beta n}{\alpha n} + \frac{\epsilon\beta}{\alpha} \quad (\text{since } ||x_j||_1, ||\mu_i||_1 \leq 1) \\ &\leq \frac{2K\epsilon\beta}{\alpha} \\ &= \frac{2K^2\beta\exp(-\beta\delta)}{\alpha}. \end{aligned}$$

Since by assumption $\beta\delta > \log\frac{2\beta K^2}{\alpha}$, we know that $||A||_1 < 1$ and applying Lemma 2 competes the proof.

Lemma 1. *Consider a point j and let $i = \arg\max_k r_{jk}$. Then, $r_{ji} \geq \frac{1}{1+K\exp(-\beta\delta)}$, and correspondingly, $\sum_{k\neq i} r_{jk} \leq \frac{K\exp(-\beta\delta)}{K\exp(-\beta\delta)+1} \leq K\exp(-\beta\delta)$.*

Proof. Equation 4 of (Titsias 2016) gives that

$$r_{ij} \geq \prod_{k \neq i} \frac{1}{1 + \exp(-(s_i - s_k))}.$$

Since by assumption we have $-||x_j - \mu_i|| \geq \delta||x_j - \mu_k||$, we obtain

$$\begin{aligned} r_{ij} &\geq \prod_{k \neq i} \frac{1}{1 + \exp(-\delta\beta)} \\ &\geq \frac{1}{1 + K\exp(-\delta\beta)} \quad \text{(using that } \exp(-\delta\beta) \leq 1). \end{aligned}$$

which proves the lemma. □

Lemma 2. *Suppose that for a matrix A, $||A - I|| \leq \delta$ for some $\delta < 1$ and an operator norm $||\cdot||$. Then, $||A^{-1} - I|| \leq \frac{\delta}{1-\delta}$.*

Proof. Let $B = I - A$. We have

$$\begin{aligned} A^{-1} &= (I - B)^{-1} \\ &= \sum_{i=0}^{\infty} B^i \quad \text{(using the Neumann series representation)} \\ &= I + \sum_{i=1}^{\infty} B^i \end{aligned}$$

and so $||A^{-1} - I||_\infty = \left|\left|\sum_{i=1}^{\infty} B^i\right|\right|_\infty$. We have

$$\begin{aligned} \left|\left|\sum_{i=1}^{\infty} B^i\right|\right|_\infty &\leq \sum_{i=1}^{\infty} ||B^i||_\infty \\ &\leq \sum_{i=1}^{\infty} ||B||_\infty^i \\ &\quad \text{(since operator norms are submultiplicative)} \\ &= \frac{\delta}{1-\delta} \quad \text{(geometric series).} \end{aligned}$$

□

Experimental setup details

Hyperparameters

All methods were trained with the Adam optimizer. For the single-graph experiments, we tested the following settings on the pubmed graph (which was not used in our single-graph experiments):

- β = 1, 10, 30, 50
- learning rate = 0.01, 0.001
- training iterations = 100, 200, ..., 1000
- Number of forward pass k-means updates: 1, 3
- Whether to increase the number of k-means updates to 5 after 500 training iterations.
- GCN hidden layer size: 20, 50, 100
- Embedding dimension: 20, 50, 100

For all single-graph experiments we used $\beta = 30$ for the facility location objective and $\beta = 50$ for community detection, $\gamma = 100$, GCN hidden layer = embedding dimension = 50, 1 k-means update in the forward pass, learning rate = 0.01, and 1000 training iterations, with the number of k-means updates increasing to 5 after 500 iterations.

We tested the following set of hyperparameters on the validation set for each graph distribution

- β = 30, 50, 70, 100
- learning rate = 0.01, 0.001
- dropout = 0.5, 0.2
- training iterations = 10, 20...300
- Number of forward pass k-means updates: 1, 5, 10, 15
- Hidden layer size: 20, 50, 100
- Embedding dimension: 20, 50, 100

We selected $\beta = 70$, learning rate = 0.001, dropout = 0.2, and hidden layer = embedding dimension = 50 for all experiments. On the synthetic graphs we used 70 training iterations and 10 forward-pass k-means updates. For pubmed, we used 220 and 1, respectively.

Synthetic graph generation

Each node has a set of attributes y_i (in this case, demographic features simulated from real population data); node i forms a connection to node j with probability proportional to $e^{-\frac{1}{\rho}||y_i - y_j||}d(j)$ where $d(j)$ is the degree of node j. This models both the homophily and heavy-tailed degree distribution seen in real world networks. We took $\rho = 0.025$ to obtain a high degree of homophily, so that there is meaningful community structure. In order to make the problem more difficult, our method does not observe the features y; instead, we generate unsupervised features from the graph structure alone using role2vec (Ahmed et al. 2018) (which generates inductive representations based on motif counts that are meaningful across graphs). Each graph has 500 nodes.

Code

Code and data for all of the experiments in the paper may be found on github. We do not include the link here for anonymity, but will include it in the final version of the paper.

Hardware

All methods were run on a machine with 14 i9 3.1 GHz cores and 128 GB of RAM. For fair runtime comparisons with the baselines, all methods were run on CPU.

Additional experimental results

We run experiments on Intel i9 7940X @ 3.1 GHz with 128 GB of RAM. We report runtime in seconds. For algorithms with learned models, we report both the training time and the time to complete a single forward pass.

Table 3: Results for community detection. "-" for GCN-2Stage-Newman in the Learning + optimization section denotes that the method could not be run due to numerical issues.

	Learning + optimization					Optimization				
	cora	cite.	prot.	adol	fb	cora	cite.	prot.	adol	fb
ClusterNet	**0.56**	**0.53**	**0.28**	**0.47**	**0.28**	**0.71**	**0.76**	**0.52**	0.55	**0.80**
GCN-e2e	0.01	0.01	0.06	0.08	0.00	0.07	0.08	0.14	0.15	0.15
Train-CNM	0.20	0.44	0.09	0.01	0.17	0.08	0.34	0.05	**0.60**	**0.80**
Train-Newman	0.08	0.15	0.15	0.14	0.07	0.20	0.22	0.29	0.30	0.47
Train-SC	0.06	0.04	0.05	0.22	0.21	0.15	0.08	0.07	0.46	0.79
GCN-2stage-CNM	0.20	0.23	0.18	0.32	0.08	-	-	-	-	-
GCN-2stage-Newman	0.01	0.00	0.00	-	0.00	-	-	-	-	-
GCN-2stage-SC	0.13	0.18	0.10	0.29	0.18	-	-	-	-	-

Table 4: Results for facility location

	Learning + optimization					Optimization				
	cora	cite.	prot.	adol	fb	cora	cite.	prot.	adol	fb
ClusterNet	**9**	**14**	**7**	**5**	**2**	**8**	**13**	**6**	**5**	**2**
GCN-e2e	12	15	8	6	4	10	14	7	**5**	4
Train-greedy	14	16	8	8	6	9	14	7	6	5
Train-gonzalez	11	15	8	7	6	9	**13**	7	6	**2**
GCN-2Stage-greedy	14	16	8	7	6	-	-	-	-	-
GCN-2Stage-gonzalez	12	16	8	6	5	-	-	-	-	-

Table 5: Timing results for the community detection task (s)

	cora	cite.	prot.	adol	fb
ClusterNet - Training Time	59.48	149.73	129.63	56.68	54.33
ClusterNet - Forward Pass	0.04	0.12	0.11	0.04	0.05
GCN-e2e - Training Time	36.83	54.99	34.60	29.04	28.17
GCN-e2e - Forward Pass	0.002	0.005	0.002	0.003	0.001
Train-CNM	1.31	1.28	1.02	1.03	2.94
Train-Newman	9.99	15.89	15.19	11.45	7.25
Train-SC	0.41	0.62	0.55	0.38	0.48
GCN-2Stage - Training Time	68.79	72.20	75.69	103.56	57.62
GCN-2Stage-CNM	119.34	178.39	159.64	101.64	142.02
GCN-2Stage-New.	37.96	58.26	51.70	33.14	43.88
GCN-2Stage-SC	0.40	0.61	0.50	0.33	0.36

Table 6: Timing results for the kcenter task (s)

	cora	cite.	prot.	adol	fb
ClusterNet - Training Time	264.14	555.84	488.37	244.74	246.57
ClusterNet - Forward Pass	0.10	0.23	0.20	0.09	0.11
GCN-e2e - Training Time	237.68	511.23	446.76	229.49	221.28
GCN-e2e - Forward Pass	0.003	0.006	.005	0.004	.003
Train-Greedy	1029.18	2387	1966	619.06	1244.09
Train-Gonzalez	0.082	0.14	0.12	0.07	.066
GCN-2Stage - Training Time	73.82	70.21	103.98	75.48	104.66
GCN-2Stage-Greedy	1189.15	2367	2017	621.59	1237.871
GCN-2Stage-Gonzalez	0.18	0.28	0.25	0.13	0.13

Table 7: Timing results in the inductive setting for community detection task (s)

	synthetic	pubmed
ClusterNet - Training time	6.57	13.74
ClusterNet - Forward Pass	0.003	0.008
GCN-e2e - Training time	11.40	15.86
GCN-e2e - Forward Pass	0.04	0.03
Train-CNM	0.08	0.17
Train-Newman	0.65	1.83
Train-SC	0.03	0.04
2Stage - Train	10.98	15.86
2Stage-CNM	3.23	13.73
2Stage-New.	1.12	4.29
2Stage-SC	0.04	0.10

Table 8: Timing results in the inductive setting for the kcenter task (s)

	synthetic	pubmed
ClusterNet - Training Time	14.36	43.06
ClusterNet - Forward Pass	0.005	0.02
GCN-e2e - Training Time	9.49	33.73
GCN-e2e - Forward Pass	0.01	0.02
Train-Gonzalez	0.07	0.49
Train-Greedy	4.99	32.7
2Stage - Train	11.00	15.78
2Stage-Gonzalez	0.07	0.07
2Stage-Greedy	5.31	16.16