arXiv:1907.00481v1 [cs.LG] 30 Jun 2019

MINCUT POOLING IN GRAPH NEURAL NETWORKS

Filippo Maria Bianchi[*] **Daniele Grattarola**[†] **Cesare Alippi**[‡]

ABSTRACT

The advance of node pooling operations in a Graph Neural Network (GNN) has lagged behind the feverish design of new graph convolution techniques, and pooling remains an important and challenging endeavor for the design of deep architectures. In this paper, we propose a pooling operation for GNNs that implements a differentiable unsupervised loss based on the `minCUT` optimization objective. First, we validate the effectiveness of the proposed loss function by clustering nodes in citation networks and through visualization examples, such as image segmentation. Then, we show how the proposed pooling layer can be used to build a deep GNN architecture for graph classification.

1 INTRODUCTION

Graph Neural Networks (GNNs) learn how to represent entities and how to combine them, according to arbitrary relationships that are given as part of the task inputs. State of the art GNNs extend the convolution operation from regular domains, such as images or time series, to data with arbitrary topologies and unordered structures described by graphs (Bronstein et al., 2017; Battaglia et al., 2018; Wu et al., 2019).

A fundamental component in deep learning is the *pooling* operation, which replaces the output of convolutions with local summaries of nearby points and is implemented by low-pass operations, such as maximum or average (Lee et al., 2016). An effective synergy is attained by alternating the convolution operation, which extrapolates local patterns irrespective of the specific location on the signal, and pooling, which lets the ensuing convolutions capture broader patterns. Pooling allows to learn more abstract representations in deeper layers of the network, by discarding information superfluous for the final inference task and keeps complexity under control avoiding an exponential growth of intermediate features. However, traditional pooling techniques, which subsample a regular grid and replace its elements by summary statistics, are not suitable for graphs.

The development of pooling strategies for GNNs has lagged behind the design of new and more efficient graph convolutions. The reason is mainly the difficulty in defining a coarsened version of the original graph that supports the pooled signal. A naïve pooling in GNNs averages all nodes features (Li et al., 2015), but has limited flexibility since no further convolutions can be applied afterwards. Since all nodes are aggregated at once, it is not possible to extract local summaries that capture heterogeneous behaviors and local structures on the graphs. A solution is to pre-compute coarsened versions of the original graph and then fit the data to these deterministic structures (Bruna et al., 2013). This aggregation accounts for the graph connectivity, but ignores the supervised task objective as well as node the features that evolve during training.

In this paper, we propose a differentiable pooling operation implemented as a neural network layer, which can be seamlessly combined with other layers that perform message-passing operations such as graph convolutions (see Fig. 1). The parameters in the pooling layer are learned by combining a supervised loss with a regularization term, which optimizes a continuous relaxation of the normalized `minCUT` objective. The proposed *minCUTpool* yields partitions that i) cluster together nodes which are similar and strongly connected on the graph, and ii) are optimal in terms of supervised

[*]NORCE Norwegian Research Center (Tromsø, NO), `fibi@norceresearch.no`
[†]Università della Svizzera italiana (Lugano, CH), `grattd@usi.ch`
[‡]Università della Svizzera italiana (Lugano, CH) and Politecnico di Milano (Milano, IT)
Work in progress.

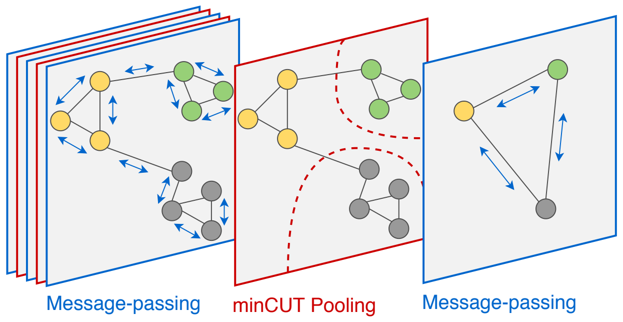


Figure 1: A deep GNN architecture where message-passing is followed by minCUT pooling.

objectives, such as the cross-entropy in graph classification. The proposed method is capable of generalization as it produces effective partitions of unseen graphs at inference time.

2 Background

2.1 Mincut and spectral clustering

Given a graph $G = \{\mathcal{V}, \mathcal{E}\}$, $|\mathcal{V}| = N$, and the associated adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, the K*-way normalized* `minCUT` (simply referred to as `minCUT`) aims at partitioning $\mathcal{V}$ in K disjoint subsets by removing the minimum volume of edges. The problem is equivalent to maximizing

$$\frac{1}{K}\sum_{k=1}^{K} \frac{\text{links}(\mathcal{V}_k)}{\text{degree}(\mathcal{V}_k)} = \frac{1}{K}\sum_{k=1}^{K} \frac{\sum_{i,j \in \mathcal{V}_k} \mathcal{E}_{i,j}}{\sum_{i \in \mathcal{V}_k, j \in \mathcal{V} \setminus \mathcal{V}_k} \mathcal{E}_{i,j}}, \tag{1}$$

where the numerator counts the edge volume within each cluster and the denominator the edges connecting the nodes in the cluster with the rest of the graph (Shi & Malik, 2000). Let $\mathbf{C} \in \mathbb{R}^{N \times K}$ be a *cluster assignment matrix*, so that $\mathbf{C}_{i,j} = 1$ if the node i belongs to cluster j, and 0 otherwise. The `minCUT` problem can be expressed as

$$\text{maximize} \quad \frac{1}{K}\sum_{k=1}^{K} \frac{\mathbf{C}_k^T \mathbf{A} \mathbf{C}_k}{\mathbf{C}_k^T \mathbf{D} \mathbf{C}_k}, \quad \text{s.t. } \mathbf{C} \in \{0,1\}^{N \times K}, \quad \mathbf{C}\mathbf{1}_K = \mathbf{1}_N, \tag{2}$$

where $\mathbf{D} = \text{diag}(\mathbf{A}\mathbf{1}_N)$ is the degree matrix (Dhillon et al., 2004). Since problem (2) is NP-hard, it is usually recast in a relaxed formulation that can be solved in polynomial time and guarantees a near-optimal solution (Yu & Shi, 2003):

$$\underset{\mathbf{Q} \in \mathbb{R}^{N \times K}}{\arg\max} \; \frac{1}{K}\text{Tr}\left(\mathbf{Q}^T \mathbf{A} \mathbf{Q}\right), \quad \text{s.t. } \mathbf{Q} = \mathbf{C}(\mathbf{C}^T\mathbf{D}\mathbf{C})^{-\frac{1}{2}}, \; \mathbf{Q}^T\mathbf{Q} = \mathbf{I}_K. \tag{3}$$

While the optimization problem (3) is still non-convex, there exists an optimal solution $\mathbf{Q}^* = \mathbf{U}_K\mathbf{O}$, where $\mathbf{U}_K \in \mathbb{R}^{N \times K}$ contains the eigenvector of $\mathbf{A}$ corresponding the K largest eigenvalues and $\mathbf{O} \in \mathbb{R}^{K \times K}$ is an arbitrary orthogonal transformation (Ikebe et al., 1987). Since the elements in $\mathbf{Q}^*$ are real values rather then binary cluster indicators, the *spectral clustering* (SC) approach treats the rows of $\mathbf{Q}^*$ as node representations embedded in the eigenspace of the Laplacian, and clusters them with standard algorithms such as k-means (Von Luxburg, 2007). One of the main limitations in SC is due to the computation of the spectrum of $\mathbf{A}$, which requires $\mathcal{O}(N^2)$ space and $\mathcal{O}(N^3)$ computations. This prevents its applicability to large datasets.

To deal with such scalability issues, the constrained optimization in (3) can be solved by gradient descent algorithms that refine the solution by iterating operations whose individual complexity is $\mathcal{O}(N^2)$, or even $\mathcal{O}(N)$ (Han & Filippone, 2017). Those algorithms search the solution on the manifold induced by the orthogonality constraint on the columns of $\mathbf{Q}$, by performing gradient updates along the geodesics (Wen & Yin, 2013; Collins et al., 2014). Alternative approaches rely on the QR factorization to constrain the space of feasible solutions (Damle et al., 2016), and alleviate the cost $\mathcal{O}(N^3)$ of the factorization by ensuring that orthogonality holds only on one minibatch at a time (Shaham et al., 2018).

Other works based on neural networks include an autoencoder trained to map the ith row of the Laplacian in the ith components of the first K eigenvectors, to avoid the spectral decomposition (Tian et al., 2014). Yi et al. (2017) use a soft orthogonality constraint to learn spectral embeddings as a volumetric reparametrization of a precomputed Laplacian eigenbase. Shaham et al. (2018); Kampffmeyer et al. (2019) propose differentiable loss functions to generate partitions on generic data that can be applied at inference time on out-of-samples. Nazi et al. (2019) generate balanced node partitions with a GNN, but follow an optimization procedure that does not encourage cluster assignments to be orthogonal.

2.2 Graph Neural Networks

Many approaches have been proposed to process a graph with a neural network, including recurrent architectures (Scarselli et al., 2009; Li et al., 2015) or convolutional operations inspired by filters used in graph signal processing (Defferrard et al., 2016; Levie et al., 2017; Bianchi et al., 2019). Since our focus is on graph pooling, we base our GNN implementation on a simple message passing (MP) operation (Gilmer et al., 2017), which combines the features of each node with its 1st-order neighbors. To account for the absence of self-loops, a typical approach is to add a (scaled) identity matrix to the diagonal of $\mathbf{A}$ (Kipf & Welling, 2016). Since our pooling will also modify the structure of the adjacency matrix, we prefer a MP implementation that operates on the original $\mathbf{A}$ and accounts for the initial node features by means of skip connections (Velickovic et al., 2017)

Let $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \in \mathbb{R}^{N\times N}$ be the symmetrically normalized adjacency matrix and $\mathbf{X} \in \mathbb{R}^{N\times F}$ the matrix containing the node features. The output of the MP layer is

$$\mathbf{X}^{(t+1)} = MP(\mathbf{X}^{(t)}, \tilde{\mathbf{A}}) = \text{ReLU}(\tilde{\mathbf{A}}\mathbf{X}^{(t)}\mathbf{W}_m + \mathbf{X}^{(t)}\mathbf{W}_s), \tag{4}$$

where $\mathbf{\Theta}_{MP} = \{\mathbf{W}_m, \mathbf{W}_s\}$ are the trainable weights relative to the mixing and skip component of the layer, respectively. In a graph classification task, which is the main focus in this work, the GNN is trained to map a pair $\{\mathbf{A}, \mathbf{X}\}$ to a discrete label y, by minimizing a supervised loss function. A common choice is to optimize the categorical cross-entropy

$$\mathcal{L}_s = -\sum_{i=1}^{C} y_i \log(\hat{y}_i), \tag{5}$$

where C is the number of classes, $y_i = 1$ if i is the correct class and zero otherwise, and $\hat{y}_i$ is the GNN prediction.

3 Proposed method

The proposed minCUTpool computes a cluster assignment matrix $\mathbf{S} \in \mathbb{R}^{N\times K}$ by means of a 2-layers MLP, which maps each node feature $\mathbf{x}_i$ into the ith row of $\mathbf{S}$:

$$\mathbf{S} = softmax(\text{ReLU}(\mathbf{X}\mathbf{W}_1)\mathbf{W}_2), \tag{6}$$

where $\mathbf{\Theta}_{Pool} = \{\mathbf{W}_1 \in \mathbb{R}^{F\times H}, \mathbf{W}_2 \in \mathbb{R}^{H\times K}\}$ are trainable parameters. The *softmax* function guarantees that $s_{i,j} \in [0, 1]$ and enforces the constraints $\mathbf{S}\mathbf{1}_K = \mathbf{1}_N$ inherited from the optimization problem in (2). The parameters $\mathbf{\Theta}_{MP}$ and $\mathbf{\Theta}_{Pool}$ are jointly optimized by minimizing a supervised loss $\mathcal{L}_s$ and an unsupervised regularization $\mathcal{L}_u$, which is composed of two terms

$$\mathcal{L}_u = \mathcal{L}_c + \mathcal{L}_o = \underbrace{-\frac{Tr(\mathbf{S}^T\tilde{\mathbf{A}}\mathbf{S})}{Tr(\mathbf{S}^T\tilde{\mathbf{D}}\mathbf{S})}}_{\mathcal{L}_c} + \underbrace{\left\|\frac{\mathbf{S}^T\mathbf{S}}{\|\mathbf{S}^T\mathbf{S}\|_F} - \frac{\mathbf{I}_K}{\sqrt{K}}\right\|_F}_{\mathcal{L}_o}, \tag{7}$$

where $\|\cdot\|_F$ indicates the Frobenius norm.

The cut loss term, $\mathcal{L}_c$, evaluates the `minCUT` given by the cluster assignment $\mathbf{S}$, and is bounded by $-1 \leq \mathcal{L}_c \leq 0$. Minimizing $\mathcal{L}_c$ encourages strongly connected nodes to be clustered together, since the inner product $\langle \mathbf{s}_i, \mathbf{s}_j \rangle$ increases when $\tilde{a}_{i,j}$ is large. $\mathcal{L}_c$ has a single maximum, reached when the numerator $Tr(\mathbf{S}^T\tilde{\mathbf{A}}\mathbf{S}) = \frac{1}{K}\sum_{k=1}^{K}\mathbf{S}_k^T\tilde{\mathbf{A}}\mathbf{S}_k = 0$. This occurs if, for each pair of connected nodes (i.e., $\tilde{a}_{i,j} > 0$), the cluster assignments are orthogonal (i.e., $\langle \mathbf{s}_i, \mathbf{s}_j \rangle = 0$). $\mathcal{L}_c$ reaches its minimum, -1, when $Tr(\mathbf{S}^T\tilde{\mathbf{A}}\mathbf{S}) = Tr(\mathbf{S}^T\tilde{\mathbf{D}}\mathbf{S})$. This occurs when in a graph with K disconnected components the cluster assignments are equal for all the nodes in the same component and orthogonal to the cluster assignments of nodes in different components. However, $\mathcal{L}_c$ is a non-convex function and its minimization can lead to local minima or degenerate solutions. For example, given a connected graph, a trivial optimal solution is the one that assigns all nodes to the same cluster. As a consequence of the continuous relaxation, another degenerate minimum occurs when the cluster assignments are all uniform, that is, all nodes are equally assigned to all clusters. This problem is exacerbated by prior message-passing operations, which make the node features more uniform.

The orthogonality loss term, $\mathcal{L}_o$, penalizes the degenerate minima of $\mathcal{L}_c$ by encouraging the cluster assignments to be orthogonal and the clusters to be of similar size. Since the two matrices in $\mathcal{L}_o$ have unitary norm, $\mathcal{L}_o$ does not dominate over $\mathcal{L}_c$ (see Fig. 4 for an example). It is easy to see that $0 \leq \mathcal{L}_o \leq 2$, since in general, given two matrices A and B, $\|A - B\| = \|A + (-B)\| \leq \|A\| + \|-B\| = \|A\| + \|B\| = 2$. Contrarily to SC methods that search feasible solutions only within the space of orthogonal matrices, $\mathcal{L}_o$ only poses a soft constraint that can be violated during the learning procedure. Since $\mathcal{L}_c$ is non-convex, the violation compromises the theoretical guarantee of convergence to the optimum of (3). However, it must be noticed that:

1. in the GNN architecture, the `minCUT` objective is a regularization term and, therefore, a solution which is sub-optimal for (3) could be, instead, optimal for the supervised loss $\mathcal{L}_s$;
2. $\mathcal{L}_s$ contributes in steering the gradient descent astray from spurious minima.

3.1 Coarsening

The coarsened version of the adjacency matrix and the graph signal are computed as

$$\tilde{\mathbf{A}}^{pool} = \mathbf{S}^T\tilde{\mathbf{A}}\mathbf{S}; \quad \mathbf{X}^{pool} = \mathbf{S}^T\mathbf{X}, \tag{8}$$

where the entry i, j of $\mathbf{X}^{pool} \in \mathbb{R}^{K\times F}$ is the average value of the feature j among the elements in cluster i. $\tilde{\mathbf{A}}^{pool} \in \mathbb{R}^{K\times K}$ is a symmetric matrix, whose entries $\tilde{a}_{i,i}^{pool}$ are the total number of edges between the nodes of cluster i, while $\tilde{a}_{i,j}^{pool}$ is the number of edges between cluster i and j. Since $\tilde{\mathbf{A}}^{pool}$ corresponds to the numerator of $\mathcal{L}_c$ in (8), the trace maximization yields clusters with many internal connections and weakly connected with each other. Hence, $\tilde{\mathbf{A}}^{pool}$ will be a diagonal-dominant matrix, which describes a graph with self-loops much stronger than any other connection. The self-loops hamper the propagation across adjacent nodes in the convolutions ensuing the pooling operation. Therefore, we define a new adjacency matrix $\bar{\mathbf{A}}$ by zeroing its diagonal and by applying the degree normalization

$$\hat{\mathbf{A}} = \tilde{\mathbf{A}}^{pool} - \mathbf{I}_K diag(\tilde{\mathbf{A}}^{pool}); \quad \bar{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}. \tag{9}$$

where $diag(\cdot)$ returns the matrix diagonal.

3.2 Discussion and relationship with spectral clustering

There are several differences between minCUTpool and classic SC methods. SC partitions the graph based on the Laplacian, but does not account for node features. Instead, the cluster assignment $\mathbf{s}_i$ depends on $\mathbf{x}_i$, which works well if connected nodes have similar features. This is a reasonable assumption in GNNs since, even in disassortative graphs (i.e., networks where the vertices have the tendency to be connected if they are unlike (Newman, 2003)), the features become similar thanks to the MP operations performed beforehand. Another difference is that the trainable parameters in our model are of order $\mathcal{O}(K)$. For large graphs, this is much less than $\mathcal{O}(N)$, the order of parameters of $\mathbf{Q}$ in (3). SC handles a single graph and is not conceived for tasks with multiple graphs to be partitioned independently. Instead, thanks to the independence of the model parameters from N and the graph spectrum, minCUTpool can generalize to out-of-sample data without re-doing the optimization to generate partitions. This is fundamental in problems such as graph classification, where each sample is a graph with a different structure. Finally, minCUTpool directly uses the soft cluster assignments rather than performing k-means afterwards, and it can be trained on small graphs and process larger ones at inference time.

4 RELATED WORK ON POOLING IN GNNS

Differentiable pooling methods. Like our method, these approaches learn how to generate coarsened version of the graph through differentiable functions, which take as input the nodes features $\mathbf{X}$ and are parametrized by weights optimized for the task at hand.

The work that is most related to our approach is *Diffpool* (Ying et al., 2018), which uses two MP layers in parallel, one to compute the new node features $\mathbf{X}^{(t+1)}$ (as in Eq. (4)) and another to generate the cluster assignments $\mathbf{S}$. In minCUTpool, instead, we compute $\mathbf{S}$ by means of a MLP applied on $\mathbf{X}^{(t+1)}$. However, the main difference is in the regularization loss $\mathcal{L}_u$, which in Diffpool consists of two terms. The first is the *link prediction* term $\|\mathbf{A} - \mathbf{S}\mathbf{S}^T\|_F$, which minimizes the Frobenius norm of the difference between the adjacency and the Gram matrix of the cluster assignments, and encourages nearby nodes to be clustered together. The second term $\frac{1}{N}\sum_{i=1}^{N} H(\mathbf{S}_i)$ minimizes the entropy of the cluster assignments, making them similar to one-hot vectors.

A second approach, dubbed *Top-Kpool* (Cangea et al., 2018; Hongyang Gao, 2019), learns a projection vector that is applied to each node feature to obtain a score. The nodes associated to the K-highest scores are retained, while the remaining are dropped. Since the top-K selection is not differentiable, the scores are also used as a gating for the node features, to let the gradient reach the projection vector. This method is more memory efficient as it avoids generating the cluster assignments. To prevent $\mathbf{A}$ from becoming disconnected when the nodes are removed, Top-Kpool drops the rows and the columns from $\mathbf{A}^2$ and uses it as the new adjacency matrix. At every forward pass, computing the power of $\mathbf{A}$ costs $\mathcal{O}(N^2)$ and is inefficient to implement with sparse operations.

Finally, a procedure proposed in Gama et al. (2018) diffuses a signal from designated nodes on the graph and stores the sequence of diffused components observed. The resulting stream of information is interpreted as a time signal, where standard CNN pooling is applied.

Determinisitc pooling methods. These methods generate in a pre-processing step a pyramid of coarsened graphs, by accounting only for the topology of $\mathbf{A}$. During training, the node features are pooled with standard procedures and are fit into these deterministic graph structures. These methods are less flexible, but provide a stronger bias that could prevent degenerate solutions (e.g., coarsened graphs collapsing in a single node), especially in presence of few data.

The approach proposed by Bruna et al. (2013), which has been adopted also in other GNN architectures (Defferrard et al., 2016; Fey et al., 2018), exploits *GRACLUS* (Dhillon et al., 2004), a hierarchical algorithm based on SC. At each level l, two vertices i_l and j_l are clustered together in a new vertex $k_{(l+1)}$. At inference phase, max-pooling is used to determine which node of the pair is kept. Fake vertices are added so that the number of nodes can be halved each time, but this injects noisy information in the graph.

Node decimation is a method originally proposed in graph signal processing (Shuman et al., 2016), which as been adapted also for GNNs (Bianchi et al., 2019). The nodes are partitioned in two sets, according to the signs of the eigenvector of the Laplacian associated to the largest eigenvalue. One of the two sets is dropped, reducing the number of nodes each time approximately by half. Kron reduction is used to compute a pyramid of coarsened Laplacians from the remaining nodes.

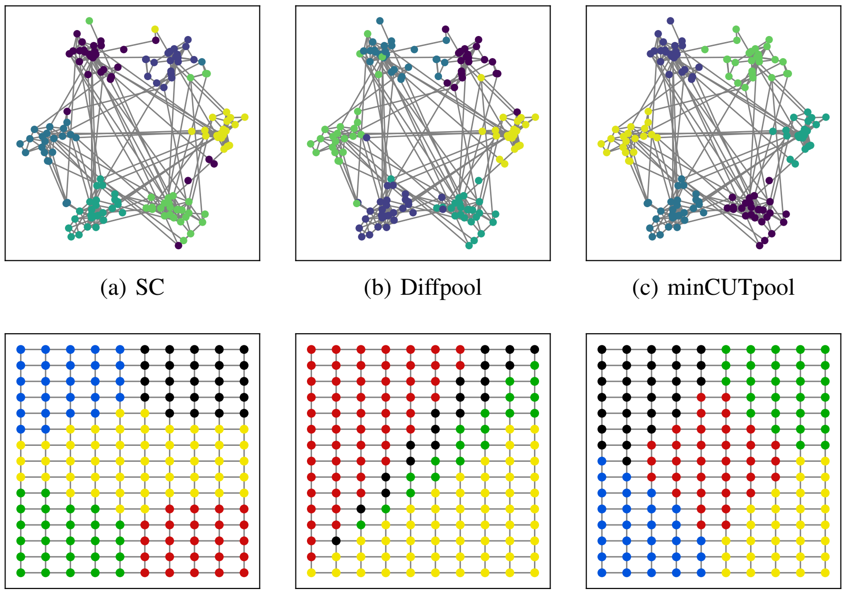
We also mention a pooling operation to coarsen binary unweighted graphs by aggregating maximal cliques (Luzhnica et al., 2019). Nodes assigned to the same clique are summarized by max or average pooling and become a new node in the coarsened graph.

5 EXPERIMENTS

5.1 EVALUATION OF THE UNSUPERVISED LOSS

To evaluate the effectiveness of the proposed loss, we perform different node clustering tasks with a simple GNN composed of a single MP layer followed by a pooling layer. The GNN is trained only by minimizing $\mathcal{L}_u$.

Clustering on synthetic networks We consider two simple graphs: the first is a network with 6 communities and the second is a regular grid. The adjacency matrix $\mathbf{A}$ is binary and the features $\mathbf{X}$ are the node coordinates. Fig. 2 depicts the nodes partitions generated by SC (a,d), Diffpool (b,e), and minCUTpool (c,f). Cluster indexes for Diffpool and minCUTpool are obtained by taking the argmax of $\mathbf{S}$ row-wise. Compared to SC, Diffpool and minCUTpool leverage the information contained in $\mathbf{X}$. minCUTpool generates very accurate and balanced partitions, demonstrating that the cluster assignment matrix $\mathbf{S}$ is well formed. On the other hand, Diffpool assigns some nodes to the wrong community in the first example, and produces an imbalanced partition of the grid.


(a) SC (b) Diffpool (c) minCUTpool

(d) SC (e) Diffpool (f) minCUTpool

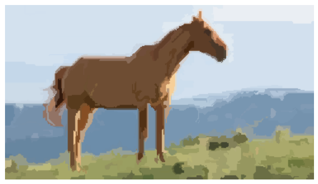
Figure 2: Node clustering on a community network (K=6) and on a grid graph (K=5).

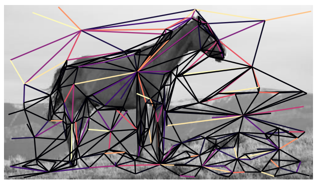
Image segmentation Given an image, we build a Region Adjacency Graph (Trémeau & Colantoni, 2000) using as nodes the regions generated by an oversegmentation procedure (Felzenszwalb & Huttenlocher, 2004). The SC technique used in this example is the recursive normalized cut (Shi & Malik, 2000), which recursively clusters the nodes until convergence. For Diffpool and minCUTpool, the node features consist of the average and total color in each oversegmented region. We set the number of desired clusters to $K = 4$. The results in Fig. 3 show that minCUTpool yields a more precise segmentation. On the other hand, Diffpool aggregates wrong regions and, in addition, SC finds too many segments.

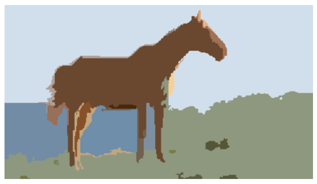
Clustering on citation networks We cluster the nodes of three popular citation networks: Cora, Citeseer, and Pubmed. The nodes are documents represented by sparse bag-of-words feature vectors stored in $\mathbf{X}$ and the binary undirected edges in $\mathbf{A}$ indicate citation links between documents. Each node i is labeled with the document class y_i. To test the quality of the partitions generated by each method we check the agreement between the cluster assignments and the original classes. Tab. 1 reports the Completeness Score $\text{CS}(\tilde{\mathbf{y}}, \mathbf{y}) = 1 - \frac{H(\tilde{\mathbf{y}}|\mathbf{y})}{H(\tilde{\mathbf{y}})}$ and Normalized Mutual Information $\text{NMI}(\tilde{\mathbf{y}}, \mathbf{y}) = \frac{H(\tilde{\mathbf{y}}) - H(\tilde{y}|\mathbf{y})}{\sqrt{H(\tilde{\mathbf{y}}) - H(\mathbf{y})}}$, where $H(\cdot)$ is the entropy.

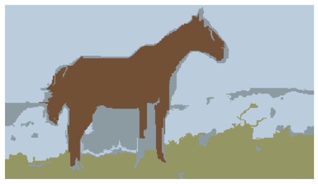
The GNN architecture configured with minCUTpool results in a higher NMI score than SC, which does not account for the node features $\mathbf{X}$ when generating the partitions. The proposed pooling operation also yields better results when compared to Diffpool, indicating that the unsupervised loss in Diffpool is unable to converge to an optimal solution, possibly due to its highly non-convex nature. This can be seen from Fig. 4, which depicts the evolution of the unsupervised losses and NMI scores of Diffpool and minCUTpool during training.

(a) Original image


(b) Oversegmentation


(c) Region Adjacency Graph


(d) SC


(e) Diffpool ($K = 4$)

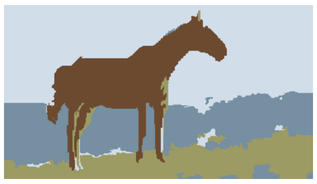
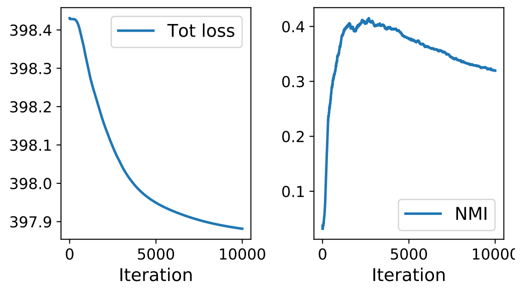
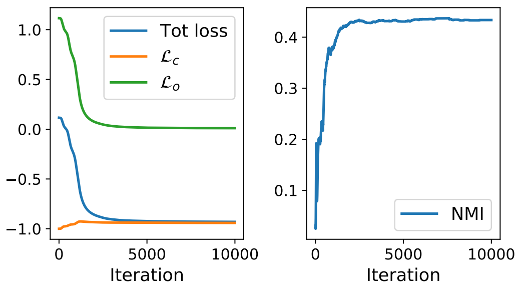

(f) minCUTpool ($K = 4$)

Figure 3: Image segmentation by clustering the nodes of the Region Adjacency Graph.



(a) Diffpool



(b) minCUTpool

Figure 4: Unsupervised losses and NMI of Diffpool and minCUTpool on Cora dataset

Table 1: NMI and CS obtained by clustering the nodes on citation networks over 10 different runs. The number of clusters K is equal to the number of node classes.

Dataset	K	**Spectral clustering**		**Diffpool**		**minCUTpool**	
		NMI	CS	NMI	CS	NMI	CS
Cora	7	0.025 ± 0.014	0.126 ± 0.042	0.315 ± 0.005	0.309 ± 0.005	**0.404** ± 0.018	**0.392** ± 0.018
Citeseer	6	0.014 ± 0.003	0.033 ± 0.000	0.139 ± 0.016	0.153 ± 0.020	**0.287** ± 0.047	**0.283** ± 0.046
Pubmed	3	0.182 ± 0.000	**0.261** ± 0.000	0.079 ± 0.001	0.085 ± 0.001	**0.200** ± 0.020	0.197 ± 0.019

5.2 SUPERVISED GRAPH CLASSIFICATION

In this task, the i-th datum is a graph with N_i nodes represented by a pair $\{\mathbf{A}_i, \mathbf{X}_i\}$ and must be associated to the correct label $\mathbf{y}_i$. We test the models on different graph classification datasets[1,2]. If the graphs are featureless, we use the node degree information and the clustering coefficient as surrogate node features. We evaluate model performance with a 10-fold train/test split, using 10% of the training set in each fold as validation for early stopping. We adopt a fixed network architecture, MP(32)-pool-MP(32)-pool-MP(32)-globalPool-softmax, where MP is the message-passing operation in (4) with 32 hidden units and the pooling module is implemented by Graclus, Decimation pooling, Top-K pooling, Diffpool and the proposed minCUTpool. We use Adam optimizer, L_2

[1]https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets
[2]https://github.com/FilippoMB/Benchmark_dataset_for_graph_classification

penalty loss with weight 1e-4,and 16 hidden units (H) in the MLP. As baselines, we consider the popular Weisfeiler-Lehman (WL) graph kernel (Shervashidze et al., 2011), a network with only MP layers (*Flat*) or fully connected layers (*Dense*).

Table 2: Graph classification accuracy. Significantly better results ($p < 0.05$) are in bold.

Dataset	**WL**	**Dense**	**Flat**	**Graclus**	**Decim.**	**Diffpool**	**TopK**	**minCUT**
Bench-easy	92.6	29.3±0.3	98.5±0.3	97.5±0.5	97.9±0.5	98.6±0.4	82.4±8.9	**99.0±0.0**
Bench-hard	60.0	29.4±0.3	67.6±2.8	69.0±1.5	**72.6±0.9**	69.9±1.9	42.7±15.2	**73.8±1.9**
Mutagenicity	**84.4**	68.4±0.3	78.0±1.3	74.4±1.8	77.8±2.3	77.6±2.7	71.9±3.7	79.9±2.1
Proteins	71.2	68.7±3.3	72.6±4.8	68.6±4.6	70.4±3.4	72.7±3.8	69.6±3.5	**76.5±2.6**
DD	78.6	70.6±5.2	76.8±1.5	70.5±4.8	70.1±3.0	**79.3±2.4**	69.4±7.8	**80.3±2.0**
COLLAB	74.8	79.3±1.6	**82.1±1.8**	77.1±2.1	75.8±2.2	81.8±1.4	79.3±1.8	**83.4±1.7**
Reddit-Binary	68.2	48.5±2.6	80.3±2.6	79.2±0.4	84.3±2.4	86.8±2.1	74.7±4.5	**91.4±1.5**

Tab. 2 reports the classification results, highlighting those that are significantly better (p-value < 0.05 from the method with highest mean accuracy). The comparison with *Flat* helps to understand if a pooling operation is useful or not. The results of *Dense* instead, help to quantify how much additional information is brought by the graph structure, with respect to the node features alone. It can be seen that minCUTpool obtains always equal or better results with respect to every other architecture. On the other hand, the other pooling procedures do not always improve the performance compared to the *Flat* baseline, questioning their usefulness in some cases. Interestingly, in some dataset such as Proteins and COLLAB it is possible to obtain fairly good classification accuracy with the *Dense* architecture, meaning that the connectivity structure adds limited additional information.

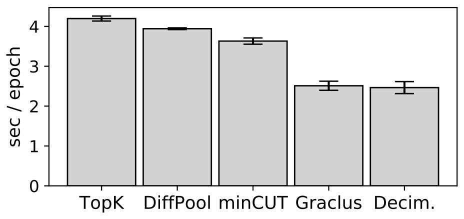


Figure 5: Average duration of one epoch using the same GNN with different pooling operations. Times were computed with an Nvidia GeForce GTX 1050, on the DD dataset with batch size of 1.

Fig. 5 reports a comparison of the execution time per training epoch for each pooling algorithm. Graclus and Decimation are understandably the fastest methods, since the coarsened graphs are precomputed. Among the differentiable pooling methods, minCUTpool is faster than Diffpool, which uses a slower MP layer rather than a MLP to compute cluster assignments, and than Top-K, which computes the square of $\mathbf{A}$ at every forward pass.

6 Conclusions

We proposed a pooling layer for GNNs that coarsens a graph by accounting for the connectivity structure and the features on the nodes. The layer exploits a regularization term that optimizes the `minCUT` objective and is minimized in conjunction with a supervised loss function to produce optimal node partitions.

To test the effectiveness of our pooling layer, we built a simple GNN architecture to solve unsupervised node clustering tasks, where we optimized the GNN parameters only by minimizing the proposed loss function. To evaluate the pooling layer in a supervised setting, we considered several graph classification datasets. Results demonstrated that our method is stable and outperforms the existing pooling strategies for GNNs.

References

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Filippo Maria Bianchi, Daniele Grattarola, L Livi, and C Alippi. Graph neural networks with convolutional arma filters. *arXiv preprint arXiv:1901.01343*, 2019.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

Maxwell D Collins, Ji Liu, Jia Xu, Lopamudra Mukherjee, and Vikas Singh. Spectral clustering with a convex regularizer on millions of images. In *European Conference on Computer Vision*, pp. 282–298. Springer, 2014.

Anil Damle, Victor Minden, and Lexing Ying. Robust and efficient multi-way spectral clustering. *arXiv preprint arXiv:1609.08251*, 2016.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pp. 3844–3852, 2016.

Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 551–556. ACM, 2004.

Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 869–877, 2018.

Fernando Gama, Antonio G Marques, Geert Leus, and Alejandro Ribeiro. Convolutional neural networks architectures for signals supported on graphs. *arXiv preprint arXiv:1805.00165*, 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.

Yufei Han and Maurizio Filippone. Mini-batch spectral clustering. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 3888–3895. IEEE, 2017.

Shuiwang Ji Hongyang Gao. Graph u-net. *Open reviews*, 2019.

Yasuhiko Ikebe, Toshiyuki Inagaki, and Sadaaki Miyamoto. The monotonicity theorem, cauchy's interlace theorem, and the courant-fischer theorem. *The American Mathematical Monthly*, 94(4): 352–354, 1987.

Michael Kampffmeyer, Sigurd Løkse, Filippo M. Bianchi, Lorenzo Livi, Arnt-Børre Salberg, and Robert Jenssen. Deep divergence-based approach to clustering. *Neural Networks*, 113:91 – 101, 2019. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2019.01.015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Chen-Yu Lee, Patrick W Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial Intelligence and Statistics*, pp. 464–472, 2016.

Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *arXiv preprint arXiv:1705.07664*, 2017.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Enxhell Luzhnica, Ben Day, and Pietro Lio. Clique pooling for graph classification. *arXiv preprint arXiv:1904.00374*, 2019.

Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, and Azalia Mirhoseini. Gap: Generalizable approximate graph partitioning framework. *arXiv preprint arXiv:1903.00614*, 2019.

Mark EJ Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003.

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.

Uri Shaham, Kelly Stanton, Henry Li, Boaz Nadler, Ronen Basri, and Yuval Kluger. Spectralnet: Spectral clustering using deep neural networks. *arXiv preprint arXiv:1801.01587*, 2018.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Departmental Papers (CIS)*, pp. 107, 2000.

David I Shuman, Mohammad Javad Faraji, and Pierre Vandergheynst. A multiscale pyramid transform for graph signals. *IEEE Transactions on Signal Processing*, 64(8):2119–2134, 2016.

Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *AAAI*, pp. 1293–1299, 2014.

Alain Trémeau and Philippe Colantoni. Regions adjacency graph applied to color image segmentation. *IEEE Transactions on image processing*, 9(4):735–744, 2000.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

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

Zaiwen Wen and Wotao Yin. A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 142(1-2):397–434, 2013.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2282–2290, 2017.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pp. 4800–4810, 2018.

Yu and Shi. Multiclass spectral clustering. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pp. 313–319 vol.1, Oct 2003. doi: 10.1109/ICCV.2003.1238361.