Gradient-based Hierarchical Clustering

Nicholas Monath*, **Ari Kobren***, **Akshay Krishnamurthy, Andrew McCallum**
College of Information and Computer Science
University of Massachusetts Amherst
{nmonath, akobren, akshay, mccallum}@cs.umass.edu

Abstract

We derive a continuous cost function for hierarchical clustering that is amenable to gradient-based optimization. Our continuous cost function is inspired by a recently proposed, discrete hierarchical cost function [5]. We present an accompanying algorithm that proceeds in two stages. In the first stage, the algorithm learns parametric routing functions at each node in a fixed hierarchy. In the second stage, the data points are clustered by routing each one from the root to a leaf. The routing function at each node may be arbitrarily complex as long as it is differentiable–e.g., a neural network. This facilitates discovery of arbitrary cluster boundaries. We present two algorithms for routing function optimization; the more efficient algorithm optimizes routers of disjoint subtrees in parallel, leading to enhanced scalability. In experiments with traditional clustering datasets, our method is competitive with other state of the art methods.

1 Introduction

Clustering is a fundamental unsupervised learning problem that has been widely studied in both theory and practice. Clustering algorithms can be organized into two families: hierarchical and flat. Hierarchical clustering has proven to be particularly useful for many reasons. A hierarchy simultaneously encodes flat clusterings of various granularities, facilitates data visualization, and promotes efficient search and insertion of new data. In the flat clustering setting, gradient-based approaches are remarkably effective because they are scalable and efficient; they can also be used for online optimization [13]. However, in the hierarchical clustering setting, gradient-based methods are scarce. This is largely due to the fact that most hierarchical clustering methods are defined algorithmically rather than in terms of cost minimization; in the few cases in which a cost function is well-defined [7, 5, 14], gradient-based methods are not a natural fit because of the discreteness of the problem.

In this work, we present a continuous cost function for hierarchical clustering that can be optimized using gradient-based methods. Our continuous objective is inspired by a discrete cost function that has recently received significant study [5]. Both the continuous and discrete objectives encourage similar points to be placed near one another in the hierarchy while also encouraging the hierarchy to be balanced. In more detail, for each pair of points, the cost incurred is proportional to the product of their distance to one another in the tree and their similarity. Thus, splitting similar points near the root is expensive, while splitting dissimlar points near the root is cheap.

In our continuous hierarchical clustering model, each node stores a *parametric routing function* that is used to compute the fit of any data point at that node. Collectively, the routers define how data points are placed in the hierarchical clustering subject to a top-down traversal of the tree structure. We show how our continuous cost can be written as a differentiable function in terms of the routing parameters alone. Then, the search over hierarchical clusterings can be performed efficiently via

*The first two authors contributed equally.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

gradient-based methods. This parametric tree structure bears resemblance to hierarchical structures used for extreme classification [11, 1, 4, 6]. Unlike the extreme classification setting, our setting is completely unsupervised.

Our approach is advantageous from a modeling perspective because it supports joint training of arbitrarily complex routing functions. This advantage plays a significant role in hierarchical clustering quality especially for the the first bipartitioning of the data is crucially important and notoriously difficult [6]. With complex, jointly trained routers it is possible to learn a locally suboptimal split of the data at the root for the sake of creating a globally optimal hierarchical clustering. In this work, we explore the use of 2-layer neural network routing functions.

Jointly optimizing routing parameters of all nodes in the tree can be expensive. Thus, we also present an alternative training procedure that scales logarithmically in the size of the tree. This approach jointly optimizes the routers in specific subtrees independently. In this setting, we are able to improve training efficiency by optimizing disjoint subtrees in parallel. The use of stochastic gradient descent and the continuous cost function makes our approach highly amenable to parallel computation and accelerated computing using GPUs.

We compare the clustering performance of our method to state-of-the-art hierarchical clustering algorithms on benchmark datasets. The results show that our method is competitive with existing approaches. Our experiments highlight the power of both non-linear routing functions and joint training. Our work introduces a method for gradient-based hierarchical clustering, which we believe has the potential to be highly scalable and effective in practice.

2 A Continuous Cost Function for Hierarchical Clustering

Hierarchical clustering is a recursive partitioning of data in a tree structure. Formally,

Definition 1 (Hierarchical Clustering [9])**.** *A binary* ***hierarchical clustering*** $\mathcal{T}$ *on a dataset* $\{x_i\}_{i=1}^N$ *is a collection of subsets such that* $C_0 \triangleq \{x_i\}_{i=1}^N \in \mathcal{T}$ *and for each* $C_i, C_j \in \mathcal{T}$ *either* $C_i \subset C_j$, $C_j \subset C_i$ *or* $C_i \cap C_j = \emptyset$*. For any* $C \in \mathcal{T}$*, if* $\exists C' \in \mathcal{T}$ *with* $C' \subset C$*, then there exists two* $C_L, C_R \in \mathcal{T}$ *that partition* C*.*

Our goal is to develop a gradient-based method for hierarchical clustering capable of discovering complex cluster boundaries while promoting efficiency and scalability. To do this, we employ a *routing-based* hierarchical clustering approach [15, 8]–that is, each point is clustered via a top-down traversal of the hierarchy. We define a continuous, differentiable cost function that can be expressed in terms of the routers in the tree and show how to optimize the routing parameters via gradient descent.

2.1 Hierarchical Clustering as Routing in a Tree

In our model, each node, v, (other than the root node) is assigned a parametric routing function, $f(\cdot;\theta^{(v)})$, where $\theta^{(v)}$ are v's parameters. The router at v scores the fit of a point in that node. Given that a point x reaches v, the probability that x will be routed to v's left child, $v.\texttt{left}$, is proportional to the score of the routing function at $v.\texttt{left}$ with input x, i.e. $\Pr[x \rightsquigarrow v.\texttt{left}|v] \propto f(x;\theta^{(v.\texttt{left})})$. The normalized probability is computed via the softmax between the scores for x at $v.\texttt{left}$ and $v.\texttt{right}$. Further, the (unconditional) probability of x reaching any node v' is simply the product of scores corresponding to the path from the root to v'. Given a tree $\mathcal{T}$ and the corresponding set of routing parameters, clustering is performed by routing each data point greedily along its route-to-leaf path. The probability of such a hierarchical clustering is computed using the path probabilities.

Related Work Tree structures are commonly used in classification to reduce the complexity of classifying a data point from linear to logarithmic in the size of the label-space. Many approaches including hierarchical softmax [11] and more recent work in extreme classification such as LOM-Trees [4] and Recall Trees [6], make use of learned routing functions at each node of the tree. The router at

each node determines the child to which a data point should descend. The primary difference among these approaches lies in the mechanisms by which the routers are trained and a point is labeled once it reaches a leaf. Our work is inspired by this hierarchical router-learning paradigm but differs from previous work significantly in that our setting is unsupervised.

Some previous work in hierarchical clustering also develops top-down, routing-based algorithms. Both PERCH and BIRCH route incoming points to the leaves of an incrementally built tree [15, 8]. In PERCH, routing in performed via $A^\star$ search, which can be slow; in BIRCH, routing a new data point is performed by choosing, at each node in the tree, the child for which the distance between the new data point and the mean of the descendants of that child is smallest. While routing in BIRCH is efficient, the algorithm produces low-quality clusterings in practice.

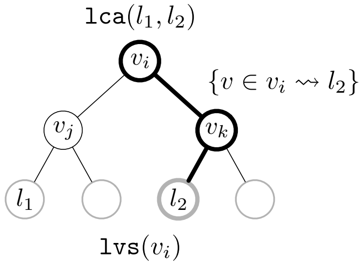

Figure 1: A tree. $v_i = \mathtt{lca}(l_1, l_2)$ and $|\mathtt{lvs}(v_i)| = 4$, shown in gray. The path $v_i \rightsquigarrow l_2$ is bolded.

2.2 Dasgupta's Hierarchical Clustering Cost Function

Our work is inspired by a recently proposed cost function for hierarchical clustering [5]. For a dataset X, the cost of a tree, $\mathcal{T}$, is defined to be:

$$J(\mathcal{T}) = \sum_{x,x' \in X^2} \mathsf{Sim}(x, x')|\mathtt{lvs}(\mathtt{lca}(x, x'))| \tag{1}$$

where, X^2 is the set of pairs of points of X, $\mathtt{lvs}(\cdot)$ returns the descendant leaves of its argument and $\mathtt{lca}(\cdot, \cdot)$ returns the least common ancestor of its arguments (see Figure 1). In words, for each pair of points in the tree, the cost of the pair is the product of their similarity and the number of descendant leaves of their least common ancestor (and the total cost is the sum of these products). Intuitively, we incur a large cost for splitting similar points near the root of the tree, since the root has the largest number of descendant leaves. The cost function has a number of nice properties like: it encourages balanced trees, the optimal tree under the cost is binary and minimizing the cost of any subtree improves the global objective.

Unsurprisingly, finding the minimum cost tree, $\mathcal{T}^\star = \operatorname{argmin}_{\mathcal{T}} J(\mathcal{T})$, is NP-hard. Moreover, the algorithm presented for constructing the tree relies on recursively solving sparsest-cut problems. This strategy is somewhat undesirable in that it may make mistakes by virtue of local optimization of each cut rather than global optimization of a collection of cuts. Recent work develops global inference algorithms based on rounding relaxed integer and semi-definite programs [12, 3]. These approaches also achieve provably better approximation guarantees. To our knowledge, no gradient-based algorithms for optimizing the cost, or related cost functions, are known.

2.3 A Continuous Version of the Cost Function

We derive a continuous objective for hierarchical clustering inspired by the cost function described above (Section 2.2). In the continuous objective, for each pair of data points, the cost incurred is computed with respect to the probability distribution over the pair's least common ancestor in the tree:

$$\begin{aligned}
J'(\mathcal{T}; \Theta) &= \sum_{x,x' \in X^2} \mathsf{Sim}(x, x')\Pr[|\mathtt{lvs}(\mathtt{lca}(x, x'))|] \\
&= \sum_{x,x' \in X^2} \mathsf{Sim}(x, x') \sum_{v \in \mathcal{T}} \Pr[v = \mathtt{lca}(x, x')]\Pr[|\mathtt{lvs}(v)|] \\
&= \sum_{x,x' \in X^2} \mathsf{Sim}(x, x') \sum_{v \in \mathcal{T}} \Pr[v = \mathtt{lca}(x, x')] \sum_{x'' \in X} \Pr[x'' \rightsquigarrow v] \\
&= \sum_{x,x' \in X^2} \sum_{x'' \in X} \sum_{v \in \mathcal{T}} \mathsf{Sim}(x, x')\Pr[v = \mathtt{lca}(x, x')]\Pr[x'' \rightsquigarrow v].
\end{aligned}$$

The cost function can be rewritten by expressing both $\Pr[x'' \rightsquigarrow v]$ and $\Pr[v = \texttt{lca}(x, x')]$ in terms of the routing parameters . The term $\Pr[x'' \rightsquigarrow v]$ is simply a product of conditional probabilities along the path from the root to v with respect to the data point x'' (Section 2.1). The term $\Pr[v = \texttt{lca}(x, x')]$ can be computed as follows:

$$\Pr[v = \texttt{lca}(x, x')] = \Pr[x \rightsquigarrow v.\texttt{right}]\Pr[x' \rightsquigarrow v.\texttt{left}] + \Pr[x \rightsquigarrow v.\texttt{left}]\Pr[x' \rightsquigarrow v.\texttt{right}]$$

where $v.\texttt{left}$ is v's left child and $v.\texttt{right}$ is v's right child. In words, the probability that v is the least common ancestor of x and x' is the probability that both x and x' arrive at v but then one is routed to the left and the other is routed to the right. After rewriting the cost in terms of the routing parameters the cost can be optimized via gradient-based methods–as long as the routing functions are differentiable.

3 Router Optimization

In this section, we describe two algorithms for optimizing the routing parameters. The first procedure jointly optimizes all routers in the tree and second optimizes subtrees independently for improved efficiency.

For both methods, the tree structure, $\mathcal{T}$, is fixed a priori; specifically, $\mathcal{T}$ is a complete binary tree with L leaves (where L is a hyper-parameter of the algorithm). Note that this restricts the space of trees to those that have binary branching factor and do not exceed a depth of $\log_2 L$. However, the shape of the resulting hierarchical clustering need not be balanced. For example, if L is large, some nodes may be unused (i.e., no data points are routed to those nodes). Unused nodes may be pruned in post-processing.

Full Tree Optimization As written, the continuous cost function has an inner sum over all data points, another inner sum over all nodes in the tree, and an outer sum over all pairs of data points (Section 2.3). Computing the cost exactly for a large dataset is not scalable. One mitigation strategy is to estimate various elements of the continuous cost function using samples.

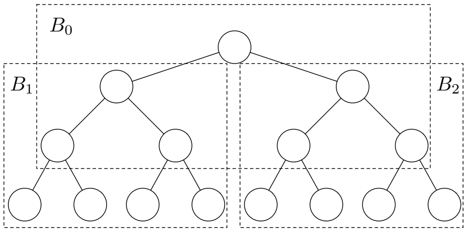

Figure 2: BW Training. B_0 is trained first; B_1 and B_2 are trained in parallel. Data point used in training B_1 and B_2 are determined by routing through B_0.

In each iteration of full tree (FT) optimization, we approximate the cost of a tree $\mathcal{T}$ using a collection of sampled pairs, $S \subset X^2$. For each pair, we approximate the number of data points under each node v using a second sample, $S'^{(i)}$: $\sum_{x \in X} \Pr[x \rightsquigarrow v] \approx \sum_{s \in S'} \Pr[s \rightsquigarrow v] \cdot \frac{|X|}{|S|}$ where i indicates the i^{th} sample (i.e., sampled pair) in S. We compute this approximation for each $v \in \mathcal{T}$ (hence, *full tree* optimization). Each training mini-batch of size b contains b pairs, i.e., $|S| = b$. The cost is minimized via stochastic gradient descent.

Block-wise Optimization Block-wise (BW) optimization minimizes a further approximation of the cost function. In BW optimization we create a set of blocks, $\mathcal{B}$, where each block is a subtree of $\mathcal{T}$. We perform FT optimization within each block; blocks are optimized in a top-down order. The training pairs for a block B are sampled from the data points that would reach the root of that block subject to a top-down traversal of $\mathcal{T}$ (see Figure 2).

BW optimization is efficient because it only updates the parameters for a small number of nodes (equal to the size of the corresponding block). Additionally, a group of disjoint blocks can be trained in parallel. Another advantage of BW optimization is that it can be focused on particular regions of the tree. However, due to the extent of its approximation, BW optimization only trains a handful of routing functions jointly and thus is at risk of inducing additional errors.

Initialization and Regularization We initialize the routing parameters in a greedy level-wise approach. For each split ordered in a top-down manner, we select a point at random, p_1 from the dataset and assign it to the left child, v.`left` and find the point that is least similar to the selected point, p_2 and assign it to the right child v.`right`. We train the corresponding routing decision to route points more similar to p_1 to v.`left` and points more similar to p_2 to v.`right`.

For each pair, we penalize the entropy of the distribution over least common ancestors induced by the routing parameters. This discourages the routers from allowing data points to appear equally likely with respect to multiple paths through the tree.

4 Experiments

Evaluation Metric We evaluate our approach experimentally by comparing its hierarchical clustering performance to state-of-the-art approaches. Following previous work we evaluate the quality of the discovered hierarchical clustering using dendrogram purity [7, 2, 8]. Dendrogram purity is defined as:

$$\mathtt{DP}(\mathcal{T}) = \frac{1}{|\mathcal{P}^\star|} \sum_{k=1}^{K} \sum_{x_i, x_j \in \mathcal{C}_k^\star} \mathtt{pur}(\mathtt{lvs}(\mathtt{lca}(x_i, x_j)), \mathcal{C}_k^\star)$$

where $\mathcal{C}^\star$ is a ground-truth flat clustering, $\mathcal{P}^\star$ is the set of data point pairs in the same cluster and $\mathtt{pur}(\cdot, \cdot)$ measures the fraction of data points under its first argument (i.e., a node in the tree) that are members of the cluster defined by its second argument (i.e., a ground-truth cluster).

Methods We compare four variants of our gradient-based hierarchical clustering approach. We use two types of routing functions: linear (GHC-L) and feed-forward neural networks with a single hidden layer (GHC-N). We use both FT training (GHC-FT) and BW training (GHC-BW). We evaluate against the following hierarchical clustering algorithms: PERCH [8], BIRCH [15], a top-down partitioning via recursive sparseset cut [5], hierarchical K-Means, and hierarchical agglomerative clustering with complete link (HAC-C). We perform a grid search over hyper-parameters for our algorithm, which include: learning rate, number of samples, and hidden layer size (in GHC-N). We report the dendrogram purity average and discrete hierarchical clustering cost [5] over five random shufflings of the data.

Datasets We evaluate on three datasets from the UCI machine learning repository [10] identified as clustering benchmarks in previous work [8, 7]. The datasets are: **glass**, which has 214 points and 6 clusters, **spambase**, which as 7000 points and 2 clusters, and **digits**, which as 200 points and 10 clusters. Each dataset has ground-truth classification labels that are used for evaluation.

Results Table 3a presents the dendrogram purity results for each algorithm. We find that our approach is competitive with the best state-of-the-art approaches. More complex routing functions (i.e., neural networks) improve over linear routing models on all datasets. We find that joint training–as opposed to optimizing a single bipartition of the data in each iteration–also improves both the cost and dendrogram purity in two out of three datasets (Figure 3).

Alg	Glass	Spambase	Digits
Sparsest Cut	0.46 ± 0.01	-	0.49 ± 0.01
GHC-N-FT	0.47 ± 0.02	0.62 ± 0.02	0.51 ± 0.03
PERCH	0.47 ± 0.02	0.61 ± 0.01	**0.61 ± 0.03**
BIRCH	0.43 ± 0.01	0.60 ± 0.01	0.54 ± 0.05
HKMeans	**0.51 ± 0.01**	**0.63 ± 0.00**	0.59 ± 0.03
HAC-C	0.47	**0.63**	0.59

(a) Dendrogram purity for clustering benchmarks.

Alg	Glass	Spambase	Digits
Sparest Cut	6.97e5	-	2.66e3
GHC-L-BW-2	7.54e5	1.42e8	2.72e3
GHC-N-BW-2	7.53e5	**8.66e7**	2.71e3
GHC-L-FT	6.85e5	1.80e8	2.66e3
GHC-N-FT	**6.68e5**	9.92e7	**2.65e3**

(b) Discrete cost [5].

Figure 3: Clustering quality. Figure 3a shows that GHC-N-FT is competitive with state-of-the-art algorithms in practice. Figure 3b shows that neural routers outperform linear routers and joint training yields lower cost clusterings than training routers independently.

5 Conclusion and Discussion

In this paper, we derive a continuous cost function for hierarchical clustering. We show how to express the cost function in terms of routing parameters at each node in the hierarchy and give two algorithms for optimizing the cost via gradient descent. We compare our approach to other hierarchical clustering algorithms empirically and demonstrate that it achieves high quality clustering performance.

By virtue of leveraging parallel training and computation of gradients, our method has potential to be highly efficient and scalable. However, we notice that at times, in both FT and BW training, gradient descent requires many iterations to converge. In future work, we are interested scaling our implementation to larger datasets. This will require a deeper understanding of the cost function in order to design initialization and regularization techniques–as well as modifications to the cost function–that facilitate faster convergence to local minima. We are also interested in rigorously analyzing the regret associated with our training procedures.

References

[1] Alina Beygelzimer, John Langford, and Pradeep Ravikumar. Error-correcting tournaments. *ALT*, 2009.

[2] Charles Blundell, Yee Whye Teh, and Katherine A Heller. Bayesian rose trees. *UAI*, 2012.

[3] Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *SODA*. Society for Industrial and Applied Mathematics, 2017.

[4] Anna E Choromanska and John Langford. Logarithmic time online multiclass prediction. *NIPS*, 2015.

[5] Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. *STOC*, 2016.

[6] Hal Daume III, Nikos Karampatziakis, John Langford, and Paul Mineiro. Logarithmic time one-against-some. *ICML*, 2017.

[7] Katherine A Heller and Zoubin Ghahramani. Bayesian hierarchical clustering. *ICML*.

[8] Ari Kobren, Nicholas Monath, Akshay Krishnamurthy, and Andrew McCallum. A hierarchical algorithm for extreme clustering. *KDD*, 2017.

[9] Akshay Krishnamurthy, Sivaraman Balakrishnan, Min Xu, and Aarti Singh. Efficient active algorithms for hierarchical clustering. *ICML*, 2012.

[10] M. Lichman. UCI machine learning repository, 2013.

[11] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. *AISTATS*, 2005.

[12] Aurko Roy and Sebastian Pokutta. Hierarchical clustering via spreading metrics. In *NIPS*, 2016.

[13] D. Sculley. Web-scale k-means clustering. *WWW*, 2010.

[14] Michael Wick, Sameer Singh, and Andrew McCallum. A discriminative hierarchical model for fast coreference at large scale. In *ACL*. Association for Computational Linguistics, 2012.

[15] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *ACM Sigmod Record*, 1996.