CNTree Hierarchical Clustering

Oren Livne*[1] and Achi Brandt†[2]

[1]Educational Testing Service, Attn: MS-12, T-197, 660 Rosedale Road, Princeton, NJ 08540
[2]Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, 234 Herzl Street, Rehovot 7610001 Israel

December 16, 2019

Abstract

A stepping stone towards building multilevel optimization solvers for neural network training is clustering gradients of weights or activations into groups, and for each gradient computing a set of p approximate nearest neighboring groups to interpolate from. This can be cast as the problem of clustering n points in $\mathbb{R}^m$ into $\sim n/3 - n/2$ groups of small bounded radius ("mini-clustering").

We present CNTree (Cluster and Neighbor Tree), an $O(mpn \log n)$ top-down hierarchical clustering algorithm that produces both groups and neighborhoods. The tree consists of levels of groups. Each group is split into b subgroups at the next level. At each level of the tree we keep track of the neighbors of each group, which allows for an efficient application of local and global k-means iterations to determine subgroup assignment and center locations.

In addition to inertia for measuring group quality, we define a neighborhood quality metric called ED (Excess Distance). Numerical experiments show that CNTree neighborhoods are exact in over 98% of cases. A python implementation clusters 10^6 points in 16 dimensions in 3 minutes on a single CPU.

1 The Problem; Notation

We consider the problem of clustering a set of n points $\mathcal{X} := \{x_1 \ldots, x_n\}$ in $\mathbb{R}^m$ into k small groups $\mathcal{C} := \{C_1, \ldots, C_k\}$ (i.e., a coarsening ratio of

*olivne@ets.org
†achi.brandt@weizmann.ac.il

$k/n \sim 0.3 - 0.2$). For simplicity, groups are non-overlapping. A *group* C consists of a subset of X and a *center*

$$z(C) := \frac{1}{|C|} \sum_{x \in C} x \,. \tag{1}$$

Points belonging to a group are called its *members*. Conversely, the *label*(i) of a point i is the group it belongs to. Instead of explicitly controlling the number of groups as in the standard k-means algorithm [Llo82] or the group size, we require that all groups have *radius* $r(C_j) \leq r_{\max}$, $j = 1, \ldots, k$, where

$$r(C) := \left(\sum_{x \in C} \|x - z(C)\|^q \right)^{\frac{1}{q}} \tag{2}$$

with a large enough q so that the largest distances from the center dominate the sum; we use $q = 4$. The *distance measure* is the Euclidean distance $\|x-y\|$, although our algorithm works with any inner-product-induced norm.

For each $x_i, i = 1, \ldots, n$ we would also like to produce a *neighborhood*: a subset $N_i \subseteq \mathcal{C}$ of p approximately-nearest neighboring group centers, $z_j := z(C_j)$, $j = 1, \ldots, p$.

Given a (tentative neighbor) set N, $r_p(N, x) \subseteq N$ denotes the set of p nearest neighors of x in N.

2 CNTree Algorithm

Our algorithm is called CNTree (Clusters and Neighbors Tree), a top-down hierarchical clustering. The tree consists of L increasingly-finer levels. Each level l, $l = 1, \ldots, L$, contains

- Groups $\mathcal{C}^l := \left\{C_1^l, \ldots, C_{m_l}^l\right\}$.
- Point neighborhoods $\mathcal{N}^l := \left\{N_1^l, \ldots, N_n^l\right\}$, i.e. the p nearest neighboring group centers of each point.
- Group neighborhoods $\mathcal{M}^l := \left\{M_1^l, \ldots, M_{m_l}^l\right\}$, i.e. the p nearest neighboring group center of each center (including itself: $j \in M_j^l$, $j = 1, \ldots, m_l$).

Level 1 consists of a single group ($m_l = 1$) containing all points, $M_1^1 = \{1\}$ and $N_i^1 = \{1\}$, $j = 1, \ldots, n$. It is refined in L stages until no group radius exceeds $r_{\max}$, so level L is the final clustering result.

Let $b \geq 2$ be the tree branching factor. We describe a refinement stage $l-1 \to l$. Level $l-1$ is the "parent level" and l is the "child level".

For each $j = 1, \ldots, m^{l-1}$, we split the parent group C_j^{l-1} into b_j^{l-1} child groups, where

$$b_j^{l-1} := \begin{cases} b\,, & \text{if } r(C_j) > r_{\max} \text{ and } |C_j| \geq b\,, \\ 2\,, & \text{if } r(C_j) > r_{\max} \text{ and } |C_j| < b\,, \\ 1\,, & \text{otherwise.} \end{cases} \tag{3}$$

Thus $m^l = \sum_{j=1}^{m^{l-1}} b_j^{l-1}$. We denote by S_j^{l-1} the children set of C_j^{l-1} and $parent^{l-1}(k)$ the parent of child k (i.e., $parent(k) = j \, \forall k \in S_j$).

Children centers $\left\{z_k^l\right\}_{k \in S_j}$ are initialized to b_j^{l-1} distinct random members of C_j^{l-1} (better starts are conceivable but haven't been implemented, e.g., choose centers as points that are farthest from each other along the longest-axis dimension of the group).

The centers are first improved by ν_1 relatively inexpensive local k-means iterations within each parent. That is, for each $j = 1, \ldots, m^{l-1}$ we set

$$label^l(i) \leftarrow \underset{k \in S_i^{l-1}}{\arg\min} \left\| x_i - z_k^l \right\|\,, \quad i \in C_j^{l-1}\,, \tag{4}$$

followed by updating child centers by

$$z_k^l \leftarrow \frac{1}{|C_k^l| + 1} \left(\sum_{x \in C_k^l} x + z_k^l \right)\,, \quad i \in C_j^{l-1}\,. \tag{5}$$

Eq. (5) was suggested by [Pak09] to avoid empty groups, which are likely at the finest tree levels. This update also makes the k-means process more stable when a point is at nearly the same distance from multiple centers.

Subsequently, we perform ν_2 iterations to globally improve the centers. In each iteration, we define the *tentative neighborhood* of child k as

$$\tilde{M}_k^l := \bigcup_{j' \in N_{parent(k)}} S_{j'}\,, \tag{6}$$

which the set of k's siblings and cousins,

$$M_k^l := r_p\left(z_k^l, \tilde{M}_k^l\right)\,. \tag{7}$$

Similarly, we define point neighboorhoods by

$$N_i^l := r_p\left(x_i, \tilde{M}_{label^l(i)}^l\right)\,, \quad i = 1, \ldots, n\,, \tag{8}$$

$label^l(i)$ is i's label at level l, i.e., the index k of the child group C_k^l it belongs to.

Next, we perform a global k-means iteration based on the constructed neighborhoods. Namely, each point is assigned to the nearest child center in its neighorhood:

$$label^l(i) \leftarrow \underset{k \in N^l_{label^l(i)}}{\arg\min} \left\| x_i - z_k^l \right\| , \quad i = 1, \ldots, n \,, \tag{9}$$

followed by updating child centers by

$$z_k^l \leftarrow \frac{1}{|C_k^l| + 1} \left(\sum_{x \in C_k^l} x + z_k^l \right) , \quad i = 1, \ldots, m^l \,. \tag{10}$$

After ν_2 global iterations, we recalculate point and child neighborhoods based on the new centers. This was shown to important to improve neighborhood quality for small ν_2. We use $\nu_1 = \nu_2 = 1$, since neighborhood quality seemed to only marginally improve by additional iterations.

2.1 Optimized Distance Computation

A standard trick is to express

$$\|x - y\|^2 = \langle x, x \rangle - 2\langle x, y \rangle + \langle y, y \rangle \tag{11}$$

so that (9) is equivalent to

$$label^l(i) \leftarrow \underset{k \in N^l_{label^l(i)}}{\arg\min} \left\{ -2\langle x_i, z_k^l \rangle + \langle z_k^l, z_k^l \rangle \right\} , , \quad i = 1, \ldots, n \,, \tag{12}$$

(12) is typically 2-3 times faster to evaluate than (9), since the Python Numpy library inner product is highly optimized via vectorization.

2.2 Stopping Criterion

It turns out that insisting on all groups being smaller than the maximum radius is too restrictive. A more robust stopping criterion is

$$\max \left\{ p_{90} \left(\left\{ C_j^l \right\}_j \right) , \max_j \left(r(C_j^l) \right) / 1.5 \right\} < r_{\max} \,, \tag{13}$$

where p_{90} is the $90th$ percentile of the radius distribution. In other words, once 90% of the distribution is within $r_{\max}$, we allow the max radius to be up to $1.5 r_{\max}$. Since there's typically a small number of larger radius compared with the population, this allows us to reduce L a little bit, and therefore the complexity.

2.3 Small Groups

If there are empty groups arise, they are removed at the end of the refinement stage, and deleted from all neighborhoods. This means that a small fraction of neighorhoods are potentially smaller, but since p is increased by some number to start with (cf. Sec. 2.5.1 below), we still get full neighborhoods for all points.

Furthermore, at the end of level processing, groups of size 1 may have a non-zero radius because the center are not exactly aligned with the sole member due to the update (10). It is important to fix those centers to be the member coordinate so these groups are not considered for further refinement, as refinement is based on a radius threshold.

2.4 Pseudo-Code

In the pseudo code below, the data is expressed as an $n \times m$ matrix whose rows are point coordinates. The groups in C^l are maintained both as sets C^l_j and via $\{label^l(i)\}_i$ (the reverse mapping of point to group index).

2.5 Complexity and Parameter Optimization

2.5.1 Neighorhood Size

We work with neighboorhood size $p + 2$ at all levels where p is the target neighboorhood size in the final clustering. This is motivated by two reasons:

(a) Neighorhood accuracy boosting: it is possible to miss a nearest neighbor of child k at level l if k's parent has many proximal neighbors. If the neighbor happens to be a child of a close neighbor of the parent that's however not among the parent's p nearest neighbors, it won't be included in $\tilde{M}^l_k$. Since neighborhood accuracy (measured by mean and max ED, cf. Sec. 3) seems to increase exponentially with p, increasing p by a small number increases accuracy without incurring a lot more work.

(b) Empty group mitigation: after removing empty groups (cf. Sec. 2.3), some neighorhood sizes become smaller than p, so increasing p initially restores them to be large enough.

2.5.2 Branching Factor

i.e., the p nearest neighboring groups of each group (including itselfThe worst-case complexity occurs when the points are distributed approximately

Algorithm 1 CNTree

Input: points $X_{n\times m}$; maximum group radius $r_{\max}$; branching factor b; neighorhood size p

Output: groups $\mathcal{C}^L$; neighborhoods $\mathcal{N}^L$

$p \leftarrow p+2$
$l \leftarrow 1, \mathcal{C}^l \leftarrow \{\{1,\dots,n\}\}, M^l = \{\{1\}\}, N_i^l = \{1\},\ j = 1,\dots,n$
while (13) doesn't hold **do**
  $l \leftarrow l+1$
  for all $j = 1,\dots,m^{l-1}$ **do**
    Calculate b_j^{l-1} by (3)
    Initialize child centers of C_j^{l-1} to b_j^{l-1} distinct random members
  end for
  for all $t = 1,\dots,\nu_1$ **do**
    Update labels within the parent (local k-means) by (4)
    Update child centers by (5)
  end for
  for all $t = 1,\dots,\nu_2+1$ **do**
    for all $j = 1,\dots,m^l$ **do**
      Calculate tentative group neighborhood $\tilde{M}_k^l \leftarrow \bigcup_{j'\in N_{parent(k)}} S_{j'}$
    end for
    if $t \le \nu_2$ **then**
      for all $j = 1,\dots,m^l$ **do**
        Calculate group neighborhood $M_k^l \leftarrow r_p\left(z_k^l, \tilde{M}_k^l\right)$
      end for
      for all $i = 1,\dots,n$ **do**
        Calculate point neighborhood $N_i^l \leftarrow r_p\left(x_i, \tilde{M}_{label^l(i)}^l\right)$
      end for
    end if
    Update point assignment (global k-means) by (9)
    Update child centers by (10)
  end for
  Remove empty groups from $\mathcal{C}^L, \mathcal{M}^L, \mathcal{N}^L$.
  Set level l centers of groups of size 1 to their point coordinates.
end while

uniformly (otherwise larger admissible groups can be formed in locally dense areas, hence the tree would have less levels and/or most groups in higher tree levels would be considered small and require no further splitting, reducing run-time). Here $L \approx \log_b n$, and level l contains about $m_l = b^l$ groups. Level l neighorhood construction requires $O(pb)$ distance computations per group and $O(pb)$ per point. Labels update requires $O(p)$ distance computations per point, and centers update requires $O(1)$ operations per point (amortized). Thus the dominating step is point neighorhood construction, and the total complexity is $O(\nu_2 bmpn \log_b n) = O(\nu_2 (b/\log_2 b) mpn \log_2 n)$. The term $b/\log_2 b$ is minimized for $b = e$, so the optimal b is expected to be small (say, $2 \le b \le 4$). We didn't attempt to optimize b and for simplicity (and to avoid the corner case of a larger number of empty groups, which is a bit problematic in our implementation) use $b = 2$.

3 Quality Metrics

There exist plenty of clustering quality metrics [SL18, Sec. 2.3.9], however we need to properly interpret their value, as our data set does not have a small number of well-separated groups; we just wish to have reasonably separated groups. A simple metric is the *silhouette coefficient*. For each point we calculate the mean distance a between the point and all other points in its group, and the mean distance b between the point and all other points in the next nearest group. The silhouette coefficient is the mean of $(b-a)/\max\{a,b\}$ over all points. The score is in $[-1,1]$ and is higher when groups are dense and well separated. It does not require knowing the true labels, which are anyway fuzzy in this case. A silhouette coefficient of $0.3-0.5$ is good enough for our purposes.

To measure neighboorhood quality, we calculate the exact p nearest neighboring groups $z^*_{i,1}, \ldots, z^*_{i,p}$ of each point i by brute force. Let $N_i = \{z_{i,1}, \ldots, z_{i,p}\}$ be the neighorhood of point i (sorted by ascending distance from x_i). We measure

$$\mathrm{ED}_{t,i} := \frac{\|z_{i,t} - x_i\| - \|z^*_{i,t} - x_i\|}{\|z^*_{i,t} - x_i\|} \qquad i = 1, \ldots, n\,, t = 1, \ldots, p\,. \tag{14}$$

$ED_{t,i}$ is the Excess Distance (MED) of the tth approximate nearest neighbor relative to the true tth nearest neighbor for point i. $ED_{t,i} = 0$ means point i's neighborhood includes the tth the nearest neighbor; $ED_{t,i} = 1$ means the approximate tth neighbor is twice as far as the true one; and so on. Useful statistics are the mean, median and max ED over all points.

$m \times n$	ρ	Stat	Nbhr 1	Nbhr 2	Nbhr 3	Nbhr 4	Nbhr 5
1000×2	0.25	mean	0.31	1.32	1.65	2.18	2.96
		median	0	0	0	0	0
		max	217	147	111	89	62
10000×2	0.24	mean	0.17	1.79	2.28	3.18	4.20
		median	0	0	0	0	0
		max	214	243	163	109	830
20000×2	0.24	mean	0.22	2.27	2.48	3.32	4.71
		median	0	0	0	0	0
		max	484	1137	194	131	130
1000×8	0.25	mean	0.78	8.13	10.4	12.1	13.4
		median	0	0	6.9	9.8	11.5
		max	101	92	70	65	60
10000×8	0.24	mean	1.32	13.9	15.6	17.6	20
		median	0	6	11.9	15.1	18
		max	115	147	115	109	98
20000×8	0.25	mean	1.63	14.6	16.5	18.5	20.9
		median	0	7	13	16	19
		max	145	189	132	130	141

Table 1: Excess distance (%) mean and max between points and neighboring clusters vs. number of points for the uniformly random experiment.

4 Results

CNTree was implemented in Python, and the low-level routines were optimized with Cython. All reported experiments were run on a single Intel Xeon CPU, 3.2GHz, 64-bit Ubuntu machine with $p = 5$, since that's a typical neighrhood size for interpolation construction.

4.1 Random Points

We generated n random points in $[0, 1]^m$ and measured the ED statistics as a function of n. The maximum group radius was $r_{\max} = 1.5n^{-1/m}$, which is equivalent to a roughly constant coarsening ratio $\rho := |\mathcal{C}^L|/n$, because the expected number of points within a ball of radius r in $\mathbb{R}^m$ is $nV_m r^m$, where V_m is the volume of the n-dimensional unit ball.

The median ED was 0 for all neighbors t and all n for $m = 2$, and is still very small ($\leq 20\%$) for $m = 8$. This means that for the vast majority of points our neighborhoods are almost exact and the bad cases are very

unlikely. Fig.. 4.1 shows the ED distribution for the $n = 50000$. Only 2% have non-zero ED, 1% have $ED > 1$, and 0.1% have $ED > 2$. Even the mean ED stays fairly stable with n.

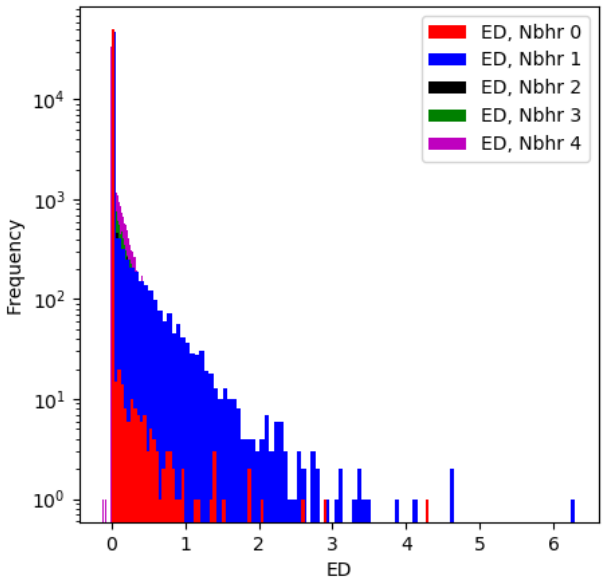

Figure 1: CNTree clustering for $n = 50000, m = 2$. Top: uniformly random points in. Bottom: Gaussian mixture.

4.2 Gaussian Mixture

In this experiment we generated 1200 points by mixing 12 multivariate Gaussian distributions with standard deviation $\sigma = 1$, whose means were on a 3×3 grid with spacing $d = 10\sigma$ (easy, non-overlapping case) and $d = 2\sigma$ (overlapping clusters). 100 points were sampled from each distribution.

In this case we expect clusters to be accurately reproduced for large d. All clusters were correctly separated for $d = 10\sigma$. However, due to the hierarhical, progressive nature of the algorithm, clusters can be split between intermediate level groups, leading to several clusters that were split into $2-3$ groups each in the final result; see Fig. 4.2. For $d = 2\sigma$ a lot more clusters (88) were identified, which is to be expected since the original 9 clusters are

not clearly separated.

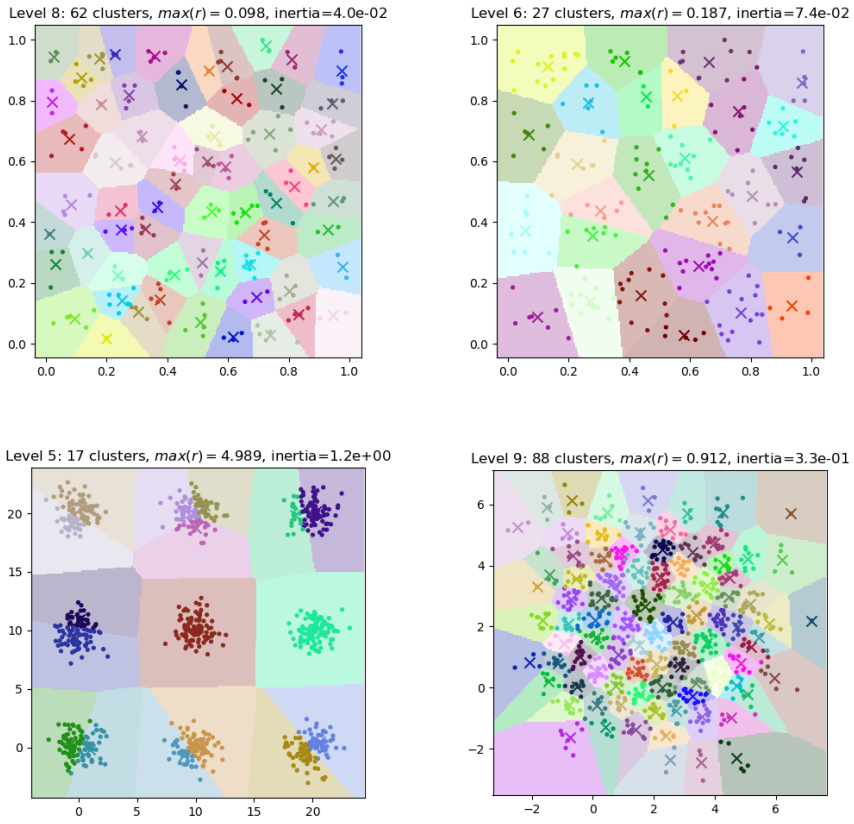

Figure 2: CNTree clustering. Top: uniformly random points with $r_{\max} = 0.1$ (left), $r_{\max} = 0.2$ (right). Bottom: Gaussian mixture with $d = 10\sigma$ (left) and $d = 2\sigma$ (right).

4.3 Runtime Scaling

We tested n uniformly distributed points in $\mathbb{R}^m$ and measured the run-time. The runtime scales roughly linearly with n and m. The time fluctuations can be partially explained by the fact that the coarsening ratio wasn't exactly constant, so the number of levels may have been relatively larger in some experiments than for others.

n	$m = 2$	$m = 4$	$m = 8$	$m = 16$	$m = 32$
10^3	0.121	0.085	0.095	0.229	0.38
10^4	1.069	0.990	1.070	1.861	3.025
10^5	8.76	8.240	12.5	21.9	38.2
10^6	122	116	121	206	-

Table 2: Runtimes [sec] vs. number of points n and dimensions m.

4.4 Conclusion

We presented a linear-scaling clustering scheme, CNTree, which also provides accurate nearest-neighborhoods. While further optimization can be implemented, the algorithm can already cluster 10^6 points in 16 dimensions in 3 minutes, which should be good enough for our neural network training application.

References

[1] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[2] Malay Kumar Pakhira. A modified k-means algorithm to avoid empty clusters. volume 1, 2009.

[3] Scitkit-Learn. Clustering performance evaluation, 2018. [Online; accessed 12-June-2018].