End-to-end Differentiable Clustering with Associative Memories

Bishwajit Saha [1] **Dmitry Krotov** [2] **Mohammed J. Zaki** [1] **Parikshit Ram** [2 3]

Abstract

Clustering is a widely used unsupervised learning technique involving an intensive discrete optimization problem. Associative Memory models or AMs are differentiable neural networks defining a recursive dynamical system, which have been integrated with various deep learning architectures. We uncover a novel connection between the AM dynamics and the inherent discrete assignment necessary in clustering to propose *a novel unconstrained continuous relaxation of the discrete clustering problem*, enabling end-to-end differentiable **cl**ustering with **AM**, dubbed **`ClAM`**. Leveraging the pattern completion ability of AMs, we further develop a novel self-supervised clustering loss. Our evaluations on varied datasets demonstrate that **`ClAM`** benefits from the self-supervision, and significantly improves upon both the traditional Lloyd's k-means algorithm, and more recent continuous clustering relaxations (by upto 60% in terms of the Silhouette Coefficient).

1. Introduction

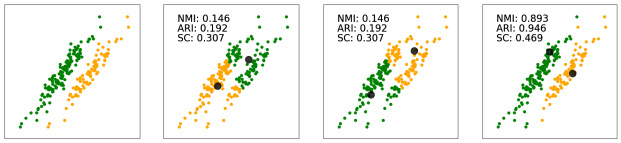
Clustering is considered one of the most fundamental **unsupervised** techniques to identify the structure of large volumes of data. Based on the similarity characteristics, clustering performs a structured division of data into groups (Xu & Wunsch, 2005). This is often considered the very first step in exploratory data analysis (Saxena et al., 2017; Bijuraj, 2013). Various formulations and algorithms have been studied to identify an efficient clustering of the data. Among them k-means (MacQueen, 1967), Spectral Clustering (Donath & Hoffman, 1973), Hierarchical Clustering (Johnson, 1967), Density-based Clustering (Ester et al., 1996), and Expectation Maximization (Dempster et al., 1977) have been widely used. However, these formulations are computationally expensive involving an intensive combinatorial task: for example, exact k-means is NP-hard (Dasgupta, 2008), though approximations can be efficient. Spectral, hierarchical and density-based clustering are computationally very expensive with a naive complexity between quadratic and cubic in the number of data points. One challenge of these formulations is the inability to leverage a differentiable algorithm – they are inherently discrete.

There has been recent interest in clustering in an end-to-end differentiable manner. **Deep clustering** techniques combine representation learning and clustering and try to learn in a differentiable manner by leveraging some **continuous relaxation** of the discrete assignment necessary in clustering (Ren et al., 2022; Zhou et al., 2022). This relaxation replaces the discrete assignment with *partial cluster assignments*, which violates a fundamental premise of clustering – each point belongs to only one cluster. With this in mind, the "sum-of-norms" form of the k-means objective allows differentiable clustering, but requires quadratic time (in the number of points) to perform the final cluster assignments (Panahi et al., 2017). Non-negative matrix factorization with structured sparsity has also been considered but requires the repeated alternating solution of two large constrained least squares problems (Kim & Park, 2008). To the best of our knowledge, *there is no differentiable clustering scheme that can seamlessly leverage the stochastic gradient descent or SGD (Nemirovski et al., 2009) based optimization frameworks (Duchi et al., 2011; Kingma & Ba, 2014) for unconstrained optimization and yet maintain the inherent discrete nature of clustering*. There is also a problem with multiple local minima, which lead to suboptimal solutions with discrete algorithms. Beyond efficiency, SGD is known to be capable of escaping local minima and can lead to models with better generalization (Hardt et al., 2016), and we believe that the problem of clustering can benefit from utilizing SGD based solutions. To that end, we look for ideas in a very distant field of associative memories.

Recently, traditional **associative memory** or AM models (Hopfield, 1982; 1984) have been reformulated to significantly increase their memory storage capacity and integrated with modern deep learning techniques (Krotov & Hopfield, 2016; Ramsauer et al., 2020; Krotov & Hopfield, 2021; Krotov, 2021). These novel models, called Dense Associative Memories, are fully differentiable systems capable of storing a large number of multi-dimensional vectors,

[1]CS Department, Rensselaer Polytechnic Institute, Troy, NY, USA [2]MIT-IBM Watson AI Lab, IBM Research, Cambridge, MA, USA [3]IBM Research, Yorktown Heights, NY, USA. Correspondence to: Bishwajit Saha <sahab@rpi.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.



(a) True (b) Lloyd's (c) DEC (d) **ClAM**

Figure 1: **Clustering with ClAM.** Two clusters (figure 1a), and solutions found by k-means (Lloyd, 1982) (figure 1b), DEC relaxation (Xie et al., 2016) of k-means (figure 1c), and our proposed end-to-end differentiable SGD-based **ClAM** (figure 1d). The black dots indicate the learned prototypes. ***ClAM*** *discovers the ground-truth clusters while the baselines cannot*. See experimental details in §4.

called patterns or "memories", in their synaptic weights. They can be trained in an end-to-end fully differentiable setting using the backpropagation algorithm, typically with a self-supervised loss.

We believe that this ability to learn synaptic weights of the AM in an end-to-end differentiable manner together with the discrete assignment (association) of each data point to exactly one memory makes AM uniquely suited for the task of differentiable clustering. We present a simple result in figure 1, where standard prototype-based clustering schemes (discrete and differentiable) are unable to find the right clusters, but our proposed AM based scheme is successful. Specifically, we make the following contributions:

- We develop a flexible mathematical framework for **cl**ustering with **AM** or **ClAM**, which is a *novel continuous unconstrained relaxation* of the discrete optimization problem of clustering that allows for clustering in an **end-to-end differentiable manner** while maintaining the discrete cluster assignment throughout the training, with linear time cluster assignment.
- We leverage the pattern completion capabilities of AMs to develop a **differentiable self-supervised loss** that improves the clustering quality.
- We empirically demonstrate that **ClAM** is able to consistently improve upon k-means by upto 60%, while being competitive to spectral and agglomerative clustering, and producing insightful interpretations.

2. Related work

Discrete clustering algorithms. Clustering is an inherently hard combinatorial optimization problem. Prototype based clustering formulations such as the k-means clustering is NP-hard, even for $k = 2$ (Dasgupta, 2008). However, the widely used Lloyd's algorithm or Voronoi iteration (Lloyd, 1982) can quite efficiently find locally optimal solutions, which can be improved upon via multiple restarts and careful seeding (Vassilvitskii & Arthur, 2006). It is an intuitive discrete algorithm, alternating between (i) assigning points to clusters based on the Voronoi partition (Aurenhammer, 1991) induced by the current prototypes, and (ii) updating prototypes based on the current cluster assignments. However, the discrete nature of this scheme makes it hard to escape any local minimum. *We believe that SGD-based prototype clustering will improve the quality of the clusters by being able to escape local minima, while also inheriting the computational efficiency of SGD*. The spectral clustering formulation (Donath & Hoffman, 1973; Von Luxburg, 2007) utilizes the eigen-spectrum of the pairwise similarity or affinity of the points (represented as a graph Laplacian) to partition the data into "connected components" but does not explicitly extract prototypes for each cluster. This allows spectral clustering to partition the data in ways disallowed by Voronoi partitions. However, it naively requires quadratic time (in the number of points) to generate the Laplacian, and cubic time for the spectral decomposition. Hierarchical clustering (Johnson, 1967) finds successive clusters based on previously established clusters, dividing them in a top-down fashion or combining in a bottom-up manner. These discrete hierarchical algorithms naively scale cubically in the number of points, although some forms of bottom-up agglomerative schemes can be shown to scale in quadratic (Sibson, 1973) or even sub-quadratic time (March et al., 2010) leveraging dual-tree algorithms (Ram et al., 2009; Curtin et al., 2013; 2015).

Density-based clustering. Schemes, such as the popular DBSCAN (Ester et al., 1996) and SNN (Aguilar et al., 2001), do not view clustering as a discrete optimization problem but as the problem of finding the **modes** in the data distribution, allowing for arbitrary shaped clusters with robustness to noise and outliers (Sander et al., 1998). However, multi-dimensional density estimation is a challenging task, especially with nonparametric methods (Silverman, 1986; Ram & Gray, 2011), leading to the use of parametric models such as Gaussian Mixture Models (Reynolds, 2009) (which can be estimated from the data with Expectation-Maximization (Dempster et al., 1977)).

Differentiable & deep clustering. Differentiability is of critical interest in deep clustering where we wish to simultaneously learn a latent representation of the points and perform clustering in that latent space (Ren et al., 2022; Zhou et al., 2022). Existing differentiable clustering formulations such as the sum-of-norms (Panahi et al., 2017) or the non-negative matrix factorization (Kim & Park, 2008) based ones are essentially solving constrained optimization and do not directly fit into an end-to-end differentiable deep learning pipeline. Hence, various schemes utilize some form of soft clustering formulation such as fuzzy k-means (Bezdek et al., 1984). Xie et al. (2016) proposed a novel probabilistic k-means formulation inspired by t-SNE (Van der Maaten & Hinton, 2008) to perform differentiable clustering (DEC) on representations from a pretrained autoencoder (AE), while

Guo et al. (2017a) (IDEC) showed gains from jointly learning representations and clusters. For representation learning in deep clustering of images, Song et al. (2013), Xie et al. (2016), and Yang et al. (2017) used stacked autoencoders, and Guo et al. (2017b) (DCEC) showed further improvements with convolutional autoencoders (CAE). Chazan et al. (2019) utilized multiple AEs, one per cluster, but the (soft) cluster assignment was performed with fuzzy k-means. Cai et al. (2022) added a "focal loss" to the DEC relaxation to penalize non-discrete assignments, while also enhancing the representation learning. Deep subspace clustering (Peng et al., 2016; Ji et al., 2017; Zhang et al., 2019) relaxes the combinatorial spectral clustering problem (again with partial cluster assignments), and utilizes "self-expressive" layers to simultaneously learn the necessary Laplacian (affinity matrix) and the clusters in a differentiable manner. Thus, deep clustering generally uses some probabilistic partial assignment of points to multiple clusters, deviating from the discrete nature of original k-means. Furthermore, these schemes often leverage pretrained AEs and/or k-means via Lloyd (1982) to initialize the network parameters. *Our proposed* ***ClAM*** *maintains the discrete assignment nature of clustering, and works well without any special seeding*.

Associative Memory (AM). This is a neural network that can store a set of multidimensional vectors – **memories** – as fixed point **attractor states** of a recurrent dynamical system. It is designed to **associate** the initial state (presented to the system) to the final state at a fixed point (a memory), thereby defining disjoint **basins of attractions** or partitions of the data domain, and thus, can represent a clustering. This network was mathematically formalized as the classical Hopfield Network (Hopfield, 1982), but is known to have limited memory capacity, being able to only store $\approx 0.14d$ random memories in a d dimensional data domain (Amit et al., 1985; McEliece et al., 1987). For correlated data, the capacity is even smaller, and often results in one fixed point attracting the entire dataset into a single basin. This behavior is problematic for clustering, since the number of clusters – the number of stable fixed points – should be decoupled from the data dimensionality. Krotov & Hopfield (2016) proposed Modern Hopfield Network or Dense AM by introducing rapidly growing non-linearities – activation functions – into the dynamical system, allowing for a denser arrangement of memories, and super-linear (in d) memory capacity. For certain activation functions, Dense AMs have power law or even exponential capacity (Krotov & Hopfield, 2016; Demircigil et al., 2017; Ramsauer et al., 2020; Lucibello & Mézard, 2023). Ramsauer et al. (2020) demonstrated that the attention mechanism in transformers (Vaswani et al., 2017) is a special limiting case of Dense AMs with the softmax activation. Dense AMs have also been used to describe the entire transformer block (Hoover et al., 2023), as well as integrated in sophisticated energy-based neural architectures (Hoover et al., 2022). See Krotov (2023) for a review of these results. Dense AMs have also been recently studied with setwise connections (Burns & Fukai, 2023), as well as applied to hetero-associative settings (Liang et al., 2022). In our work, we will show that the recurrent dynamics and the large memory capacity of Dense AMs make them uniquely suitable for clustering.

3. Associative Memories for Clustering

In this section, we present (i) the mathematical framework for **ClAM** based on AMs, (ii) motivate its suitability for clustering, and (iii) present a way of learning the memories for good clustering. We put all this together in our novel prototype-based clustering algorithm, **ClAM**.

3.1. Associative memories: Mathematical Framework

In a d-dimensional Euclidean space, consider M memories $\boldsymbol{\rho}_\mu \in \mathbb{R}^d, \mu \in [M] \triangleq \{1, \dots, M\}$ (we will discuss later how the memories are learned). The critical aspects of this mathematical framework are the **energy function** and the **attractor dynamics** (Krotov & Hopfield, 2021; Millidge et al., 2022). A suitable energy for clustering should be a continuous function of a point (or a particle) $\boldsymbol{v} \in \mathbb{R}^d$. Additionally, the energy should have M local minima, corresponding to each memory. Finally, as a particle progressively approaches a memory, its energy should be primarily determined by that single memory, while the contribution of the remaining $M-1$ memories should be small. An energy function satisfying these requirements is given by

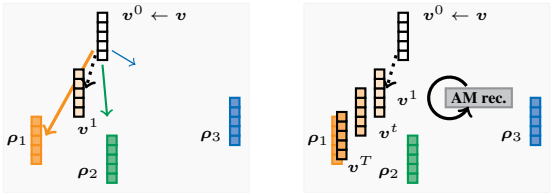
$$E(\boldsymbol{v}) = -\frac{1}{\beta} \log \left(\sum\nolimits_{\mu \in [M]} \exp(-\beta \|\boldsymbol{\rho}_\mu - \boldsymbol{v}\|^2) \right) \quad (1)$$

with a scalar $\beta > 0$ interpreted as an inverse "temperature". As β grows, the $\exp(\cdot)$ in equation 1 ensures that only the leading term remains significant, while the remaining $M-1$ terms are suppressed. Thus, the entire energy function will be described by a parabola around the closest memory. The space will be partitioned into M basins of attraction around each memory, and the shape of the energy function around each memory will be primarily defined by the closest memory, with small corrections from other memories.

The attractor dynamics control how $\boldsymbol{v}$ moves in the space over time, via $d\boldsymbol{v}/dt$, while ensuring that energy decreases. That is $dE(\boldsymbol{v})/dt < 0$, which ensures that a particle convergences to a local minimum – a fixed point of the dynamics – given the lower bounded energy. The particle dynamics is described by gradient descent on the energy landscape:

$$\tau \frac{d\boldsymbol{v}}{dt} = -\frac{1}{2} \nabla_{\boldsymbol{v}} E = \sum_{\mu \in [M]} (\boldsymbol{\rho}_\mu - \boldsymbol{v})\, \boldsymbol{\sigma}(-\beta \|\boldsymbol{\rho}_\mu - \boldsymbol{v}\|^2) \quad (2)$$

where $\tau > 0$ is a characteristic **time constant**, describing how quickly the particle moves on the energy landscape,



(a) Attraction and update. (b) Recursion to a fixed point.

Figure 2: **Attractor dynamics.** Figure 2a visualizes the attraction (direction & magnitude with solid colored arrows) of a particle $\boldsymbol{v}^0$ toward each of the memories $\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, \boldsymbol{\rho}_3$, and the resulting update $\boldsymbol{\delta}^1$ (black dotted arrow) from $\boldsymbol{v}^0 \to \boldsymbol{v}^1$ (equation 4). Figure 2b visualizes the recursive application (AM recursion) of equation 4 to the particle $\boldsymbol{v}^0 \to \boldsymbol{v}^1 \to \cdots \to \boldsymbol{v}^t \to \cdots$ till it converges to a memory $\boldsymbol{v}^{t=T} \approx \boldsymbol{\rho}_1$.

and $\boldsymbol{\sigma}(\cdot)$ is the softmax function over the scaled distances to the memories, defined as $\boldsymbol{\sigma}(z_\mu) = \exp(z_\mu)/\sum_{m=1}^{M} \exp(z_m)$ for any $\mu \in [M]$. This is visualized in figure 2a. This is guaranteed to reduce the energy since

$$\frac{dE(\boldsymbol{v})}{dt} \overset{\text{(a)}}{=} \nabla_{\boldsymbol{v}} E(\boldsymbol{v}) \cdot \frac{d\boldsymbol{v}}{dt} \overset{\text{(b)}}{=} -2\tau \left\| \frac{d\boldsymbol{v}}{dt} \right\|^2 \overset{\text{(c)}}{\leq} 0 \tag{3}$$

where (a) is the chain rule, (b) follows from equation 2, and the equality in (c) implying local stationarity. A valid update $\boldsymbol{\delta}^{t+1}$ for the point $\boldsymbol{v}$ from state $\boldsymbol{v}^t$ to $\boldsymbol{v}^{t+1} = \boldsymbol{v}^t + \boldsymbol{\delta}^{t+1}$ at a discrete time-step $t+1$ is via finite differences:

$$\boldsymbol{\delta}^{t+1} = \frac{dt}{\tau} \sum\nolimits_{\mu \in [M]} (\boldsymbol{\rho}_\mu - \boldsymbol{v}^t)\, \boldsymbol{\sigma}(-\beta \|\boldsymbol{\rho}_\mu - \boldsymbol{v}^t\|^2) \tag{4}$$

Given a dataset of points S, for each point $\boldsymbol{v} \in S$ one can corrupt it with some noise to produce a distorted point $\tilde{\boldsymbol{v}}$. This serves as an initial state of the AM network $\boldsymbol{v}^0 \leftarrow \tilde{\boldsymbol{v}}$. The AM dynamics is defined by the learnable weights $\boldsymbol{\rho}_\mu, \mu = 1, \ldots, M$, which also correspond to the fixed points of the dynamics (memories). The network evolves in time for T recursions according to equation 4, where T is chosen to ensure sufficient convergence to a fixed point (see example in figure 2b). The final state $\boldsymbol{v}^T$ is compared with the uncorrupted $\boldsymbol{v}$ to define the loss function

$$\mathcal{L} = \sum\nolimits_{\boldsymbol{v} \in S} \|\boldsymbol{v} - \boldsymbol{v}^T\|^2, \text{ where } \boldsymbol{v}^0 \leftarrow \tilde{\boldsymbol{v}} \tag{5}$$

which is minimized with backpropagation through time with respect to the AM parameters $\boldsymbol{\rho}_\mu$. In the context of clustering, the AM model naturally induces a partition of the data space into non-overlapping basins of attraction, implicitly defines a hard cluster assignment as the fixed point in each basin, and is achieved through the fully continuous and differentiable dynamics of AM, allowing learning with standard deep learning frameworks, as we discuss next.

3.2. AM as a differentiable discrete $\arg\min$ solver

Consider the original k-means objective with a dataset $S \subset \mathbb{R}^d$, where we learn k prototypes $\boldsymbol{R} \triangleq \{\boldsymbol{\rho}_\mu, \mu \in [k]\}$ with $[k] \triangleq \{1, \ldots, k\}$ by solving the following problem:

$$\min_{\boldsymbol{R}} \sum_{\boldsymbol{x} \in S} \|\boldsymbol{x} - \boldsymbol{\rho}_{\mu^\star_{\boldsymbol{x}}}\|^2, \text{s.t. } \mu^\star_{\boldsymbol{x}} = \arg\min_{\mu \in [k]} \|\boldsymbol{x} - \boldsymbol{\rho}_\mu\|^2 \tag{6}$$

The discrete selection of $\mu^\star_{\boldsymbol{x}}$ (for each $\boldsymbol{x}$) makes equation 6 a combinatorial optimization problem that cannot directly be solved via (stochastic) gradient descent. A common continuous relaxation of this problem is as follows:

$$\min_{\boldsymbol{R}} \sum\nolimits_{\boldsymbol{x} \in S} \sum\nolimits_{\mu \in [k]} w_\mu(\boldsymbol{x})\, \|\boldsymbol{x} - \boldsymbol{\rho}_\mu\|^2 \tag{7}$$

where (usually) $w_\mu(\boldsymbol{x}) \in [0, 1]$ and $\sum_{\mu \in [k]} w_\mu(\boldsymbol{x}) = 1 \, \forall \boldsymbol{x} \in S$. Hence, these weights $\{w_\mu(\boldsymbol{x}), \mu \in [k]\}$ define a probability over the k prototypes, and are designed to put the most weight on the closest prototype $\boldsymbol{\rho}_{\mu^\star_{\boldsymbol{x}}}$, and as small a weight as possible on the remaining prototypes. For example, the softmax function $\boldsymbol{\sigma}(\cdot)$ with distances to the prototypes has been used as $w_\mu(\boldsymbol{x}) = \boldsymbol{\sigma}(-\gamma \|\boldsymbol{x} - \boldsymbol{\rho}_\mu\|^2)$, where $\gamma > 0$ is a hyperparameter, and as $\gamma \to \infty$, $\sum_{\mu \in [k]} w_\mu(\boldsymbol{x}) \|\boldsymbol{x} - \boldsymbol{\rho}_\mu\|^2 \to \|\boldsymbol{x} - \boldsymbol{\rho}_{\mu^\star_{\boldsymbol{x}}}\|^2$. Other weighting functions have been developed with similar properties (Xie et al., 2016), and essentially utilize a weighted sum of distances to the learned prototypes, resulting in something different in essence to the discrete assignment $\mu^\star_{\boldsymbol{x}}$ in the original problem (equation 6). Another subtle point is that this **soft weighted assignment** of any $\boldsymbol{x} \in S$ across all prototypes **at training time** does not match the **hard cluster assignment** to the nearest prototype **at inference**, introducing an incongruity between training and inference.

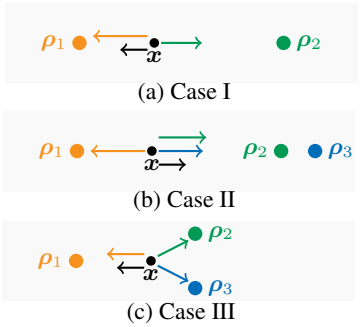
We propose a novel alternative continuous relaxation to the discrete k-means problem leveraging the AM dynamics (§3.1) that preserves the discrete assignment in the k-means objective. Given the prototypes (memories) $\boldsymbol{R} = \{\boldsymbol{\rho}_\mu, \mu \in [k]\}$, the dynamics (equation 2) and the updates (equation 4) ensure that any example (particle) $\boldsymbol{x} \in \mathbb{R}^d$ will converge *to exactly one of the prototypes* $\boldsymbol{\rho}_{\widehat{\mu}_{\boldsymbol{x}}}$ corresponding to a single basin of attraction – if we set $\boldsymbol{x}^0 \leftarrow \boldsymbol{x}$ and apply the update in equation 4 on $\boldsymbol{x}^0$ for T time-steps, then $\boldsymbol{x}^T \approx \boldsymbol{\rho}_{\widehat{\mu}_{\boldsymbol{x}}}$. Furthermore, for appropriate β, $\widehat{\mu}_{\boldsymbol{x}}$ matches the discrete assignment $\mu^\star_{\boldsymbol{x}}$ in the k-means objective (equation 6).

This implies that, for appropriately set β and T, $\boldsymbol{x}^T \approx \boldsymbol{\rho}_{\mu^\star_{\boldsymbol{x}}}$, allowing us to replace the desired per-example loss $\|\boldsymbol{x} - \boldsymbol{\rho}_{\mu^\star_{\boldsymbol{x}}}\|^2$ in the k-means objective (equation 6) with $\|\boldsymbol{x} - \boldsymbol{x}^T\|^2$, allowing us to rewrite k-means (equation 6) as the following continuous optimization problem:

$$\min_{\boldsymbol{R}} \sum\nolimits_{\boldsymbol{x} \in S} \left\|\boldsymbol{x} - \boldsymbol{x}^T_{\boldsymbol{R}}\right\|^2, \text{ where } \boldsymbol{x}^0 \leftarrow \boldsymbol{x} \tag{8}$$

where $\boldsymbol{x}^T_{\boldsymbol{R}}$ is obtained by applying update in equation 4 to any $\boldsymbol{x} \in S$ through T recursion steps, and the subscript '$\cdot_{\boldsymbol{R}}$' highlights the dependence on the prototypes $\boldsymbol{R}$.

By the above discussion, the objective in equation 8 possesses the desired *discrete assignment to a single prototype* of the original k-means objective (equation 6) since



(a) Case I

(b) Case II

(c) Case III

Figure 3: **Collective attraction.** Examples of prototype configurations, with the attraction (direction & magnitude) of $\boldsymbol{x}$ towards each prototype shown by colored arrows, and the aggregate attraction shown by the black arrow. See §3.3.

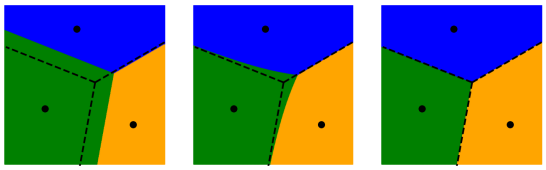
$\boldsymbol{x}_{\boldsymbol{R}}^{T}$ converges to exactly one of the prototypes. This is a significantly different relaxation of the k-means objective compared to the existing "weighted-sum" approaches. The tightness of this relaxation relies on the choices of β (inverse temperature) and T (the recursion depth) – we treat them as hyperparameters and select them in a data-dependent manner. For any given β and T, we can minimize equation 8 with SGD (and variants) via backpropagation through time.

3.3. Understanding the partitions induced by AMs

An equivalent interpretation of the discrete assignment $\mu_{\boldsymbol{x}}^{\star}$ for each $\boldsymbol{x} \in S$ in the k-means objective (equation 6) is that the prototypes $\boldsymbol{R}$ induce a Voronoi partition (Aurenhammer, 1991) of the whole input domain, and the examples $\boldsymbol{x} \in S$ are assigned to the prototype whose partition they lie in. Voronoi partitions have piecewise linear boundaries, with all points in a partition having the same closest prototype. Given prototypes $\boldsymbol{R}$, the AM dynamics induces a partition of the domain into non-overlapping basins of attraction. For an appropriately set β and T, the basins of attraction approximately match the Voronoi partition induced by the prototypes. However, when β is not "large enough", the cluster assignment happens in a more "collective" manner. We provide some intuitive examples here.

Consider two prototypes $\boldsymbol{\rho}_1, \boldsymbol{\rho}_2$, with a point $\boldsymbol{x}$ between them as shown in figure 3a, with $\boldsymbol{\rho}_1$ closest to $\boldsymbol{x}$. The attraction of $\boldsymbol{x}$ to $\boldsymbol{\rho}_1$ is inversely proportional to $\|\boldsymbol{x} - \boldsymbol{\rho}_1\|^2$ (vice versa for $\boldsymbol{\rho}_2$). In this scenario, these will work against each other and $\boldsymbol{x}$ will *move toward* the prototype with the highest attraction (based on equation 4), which is the prototype closest to $\boldsymbol{x}$ (in this case, $\boldsymbol{\rho}_1$). This further increases the attraction of $\boldsymbol{x}$ to $\boldsymbol{\rho}_1$ in the next time-step (and decreases the $\boldsymbol{\rho}_2$ attraction), resulting in $\boldsymbol{x}$ progressively converging to $\boldsymbol{\rho}_1$ through the T time-steps, and thus being "assigned" to the partition corresponding to $\boldsymbol{\rho}_1$, its closest prototype.

Consider an alternate Case-II with three prototypes in figure 3b, with $\boldsymbol{\rho}_2$ lying between $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_3$, and $\boldsymbol{x}$ lying between $\boldsymbol{\rho}_1$ and $\boldsymbol{\rho}_2$, though closer to $\boldsymbol{\rho}_1$. For a large enough β, $\boldsymbol{x}$ will converge to $\boldsymbol{\rho}_1$ through the T recursions. However, if β is not large enough, the collective attraction of $\boldsymbol{x}$ to $\boldsymbol{\rho}_2$ and $\boldsymbol{\rho}_3$ can overcome the largest single attraction of $\boldsymbol{x}$ to $\boldsymbol{\rho}_1$, forcing $\boldsymbol{x}$ to move away from $\boldsymbol{\rho}_1$, and eventually converge to $\boldsymbol{\rho}_2$ – while the dynamics guarantee convergence to exactly one of the prototypes, the collective effort of $\boldsymbol{\rho}_2$ and $\boldsymbol{\rho}_3$ causes $\boldsymbol{x}$ to **not converge to its closest prototype**.



(a) $\beta = 0.001$. (b) $\beta = 10$. (c) $\beta = 100$.

Figure 4: **Basins of attraction vs Voronoi partition.** Partitions induced by AM basins of attraction with given memories (black dots) for different β are shown by the colored regions (T=10). Dashed lines show the Voronoi partition.

While figure 3b provides an example where prototypes can combine their attractions, figure 3c presents Case-III with three prototypes where prototypes **cancel** their attraction. Here, prototypes $\boldsymbol{\rho}_2$ and $\boldsymbol{\rho}_3$ are closer to the example $\boldsymbol{x}$ than $\boldsymbol{\rho}_1$. However, for some appropriate value of β, the attraction of $\boldsymbol{x}$ to $\boldsymbol{\rho}_2$ and $\boldsymbol{\rho}_3$ will cancel each other, allowing the attraction of $\boldsymbol{x}$ to $\boldsymbol{\rho}_1$ to move the example $\boldsymbol{x}$ towards and finally converge to $\boldsymbol{\rho}_1$ (which is the farthest of the three).

These examples highlight that, for some values of β, the partition is a collective operation (involving all prototypes) leading to non-Voronoi partitions. We visualize the basins of attraction for different β in figure 4 (also §B.5, figure 11), contrasting them to the Voronoi partition. For a small β (figure 4a), the basins do not match the Voronoi partition – this mirrors the aforementioned behavior in Case-III (figure 3c). As β increases, the basins of attraction evolve to match the Voronoi partition (figure 4c). Furthermore, while the Voronoi partition boundaries are piecewise linear, the AM basins of attraction can have nonlinear boundaries (see green-orange and green-blue boundaries in figure 4b).

This behavior is an artifact of our novel continuous relaxation of discrete clustering. While we highlight this as a difference (ultimately, we are relaxing a discrete problem into a continuous one), this behavior requires specific values of β relative to the specific geometric positioning of the prototypes; this might not be as prevalent in multidimensional data. Moreover, even if the basins do not match the Voronoi partition, it will not affect the clustering objective unless some $\boldsymbol{x} \in S$ lies in the space where the partitions differ. Finally, as mentioned earlier, we treat β and T as hyperparameters, and select them in a data-dependent fashion, so that the AM partition sufficiently matches the Voronoi one.

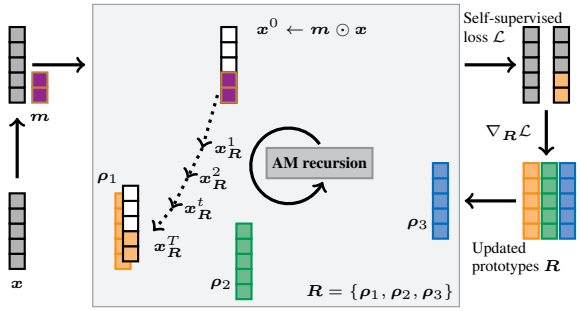


Figure 5: **Algorithm 1.** For $\boldsymbol{x} \in S$, we first apply a mask (in purple) $\boldsymbol{m}$ to $\boldsymbol{x}$ to get the initial iterate $\boldsymbol{x}^0$ for the AM recursion. With T recursions, we have a completed version $\boldsymbol{x}_{\boldsymbol{R}}^T$. The prototypes $\boldsymbol{R}$ are updated with the gradient $\nabla_{\boldsymbol{R}}\mathcal{L}$ on the self-supervised loss $\mathcal{L}$ (equation 9).

3.4. `ClAM`: Clustering with AMs and self-supervision

The AM framework allows for a novel end-to-end differentiable unconstrained continuous relaxation (equation 8) of the discrete k-means problem (equation 6). Next, we wish to leverage the aforementioned strong pattern completion abilities of AMs (§3.1). We use standard masking from self-supervised learning – we apply a (random) mask $\boldsymbol{m} \in \{0,1\}^d$ to a point $\boldsymbol{x} \in S \subset \mathbb{R}^d$ and utilize the AM recursion to complete the partial pattern $\boldsymbol{x}^0 = \boldsymbol{m} \odot \boldsymbol{x}$ and utilize the distortion between the ground-truth masked value and completed pattern as our loss as follows. Given a mask distribution $\mathcal{M}$ and denoting $\bar{\boldsymbol{m}}$ as the complement of $\boldsymbol{m}$:

$$\mathcal{L} = \sum\nolimits_{\boldsymbol{x} \in S} \mathbb{E}_{\boldsymbol{m} \sim \mathcal{M}} \|\bar{\boldsymbol{m}} \odot (\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{R}}^T)\|^2, \; \boldsymbol{x}^0 \leftarrow \boldsymbol{m} \odot \boldsymbol{x}, \quad (9)$$

where $\boldsymbol{x}_{\boldsymbol{R}}^T$ evolves from $\boldsymbol{x}^0$ with T recursions of equation 4. Prototypes $\boldsymbol{R}$ are learned by minimizing the self-supervised pattern completion loss (equation 9) via SGD. The precise learning algorithm is presented in the Train`ClAM` subroutine of algorithm 1, and visualized in figure 5. After randomly seeding the prototypes (line 2), we perform N epochs over the data (line 3) – to each $\boldsymbol{x}$ in a data batch B (lines 4-6), we apply a random mask (line 8), complete the pattern with the AM recursion using the current prototypes (lines 9-12), accumulate the batch loss as in equation 9 (line 13), and update the prototypes with the batch gradient (line 14).

Proposition 3.1. *The Train`ClAM` subroutine in algorithm 1 takes $O(dkTN|S|)$ time, where $|S|$ is the cardinality of S, converging to a $O(N^{-1/2})$-stationary point.*

The $N^{-1/2}$ term comes from the convergence rate of standard SGD for smooth non-convex problems (Nesterov, 2003) that can be improved to $N^{-2/3}$ with momentum (Fang et al., 2018), which is straightforward given the end-to-end differentiability of our proposal. The Infer`ClAM` subroutine of algorithm 1 assigns points $\boldsymbol{x} \in S$ to clusters using the learned prototypes $\boldsymbol{R}$ with a single pass over S. This is invoked once at the conclusion of the clustering.

Algorithm 1: `ClAM`: Learning k prototypes for clustering a d-dimensional dataset $S \subset \mathbb{R}^d$ into k clusters with T time-steps for N epochs, with inverse temperature β, learning rate ϵ, time step dt and time constant τ.

1 **Train`ClAM`**$(S, k, N, T, \beta, dt, \tau, \epsilon)$
2 Initialize prototypes $\boldsymbol{R} = \{\boldsymbol{\rho}_\mu \in \mathbb{R}^d, \mu \in [k]\}$ randomly
3 **for** *epoch* $n = 1, \ldots, N$ **do**
4 **for** *batch* $B \in S$ **do**
5 Batch loss $\mathcal{L}_B \leftarrow 0$
6 **for** *example* $\boldsymbol{x} \in B$ **do**
7 Sample random mask $\boldsymbol{m} \in \{0,1\}^d$
8 $\boldsymbol{x}^0 \leftarrow \boldsymbol{m} \odot \boldsymbol{x}$
9 **for** $t = 1, \ldots, T$ **do**
10 $\Delta_\mu \leftarrow \boldsymbol{\rho}_\mu - \boldsymbol{x}_{\boldsymbol{R}}^{t-1} \quad \forall \mu \in [k]$
11 $\boldsymbol{\delta}^t \leftarrow \bar{\boldsymbol{m}} \odot \frac{dt}{\tau} \sum_{\mu=1}^{k} \Delta_\mu \boldsymbol{\sigma}\left(-\beta \|\Delta_\mu\|^2\right)$
12 $\boldsymbol{x}_{\boldsymbol{R}}^t \leftarrow \boldsymbol{x}_{\boldsymbol{R}}^{t-1} + \boldsymbol{\delta}^t$
13 $\mathcal{L}_B \leftarrow \mathcal{L}_B + \|\bar{\boldsymbol{m}} \odot (\boldsymbol{x}_{\boldsymbol{R}}^T - \boldsymbol{x})\|^2$
14 $\boldsymbol{\rho}_\mu \leftarrow \boldsymbol{\rho}_\mu - \epsilon \nabla_{\boldsymbol{\rho}_\mu} \mathcal{L}_B \quad \forall \mu = 1, \ldots, k$
15 **return** *Prototypes* $\boldsymbol{R} = \{\boldsymbol{\rho}_\mu, \mu \in [k]\}$
16 **Infer`ClAM`**$(S, \boldsymbol{R}, T, \beta, dt, \tau)$
17 Cluster assignments $C \leftarrow \emptyset$
18 **for** $\boldsymbol{x} \in S$ **do**
19 $\boldsymbol{x}^0 \leftarrow \boldsymbol{x}$
20 **for** $t = 1, \ldots, T$ **do**
21 $\Delta_\mu \leftarrow \boldsymbol{\rho}_\mu - \boldsymbol{x}_{\boldsymbol{R}}^{t-1} \quad \forall \mu \in [k]$
22 $\boldsymbol{x}_{\boldsymbol{R}}^t \leftarrow \boldsymbol{x}_{\boldsymbol{R}}^{t-1} + \frac{dt}{\tau} \sum_{\mu=1}^{k} \Delta_\mu \boldsymbol{\sigma}\left(-\beta \|\Delta_\mu\|^2\right)$
23 $\widehat{\mu}_{\boldsymbol{x}} \leftarrow \arg\min_\mu \|\boldsymbol{\rho}_\mu - \boldsymbol{x}_{\boldsymbol{R}}^T\|^2$
24 $C \leftarrow C \cup \{\widehat{\mu}_{\boldsymbol{x}}\}$
25 **return** *Per-point cluster assignments* C

Proposition 3.2. *The Infer`ClAM` subroutine in algorithm 1 takes $O(dkT|S|)$ time, where $|S|$ is the cardinality of S.*

From Propositions 3.1 and 3.2, we can see, **`ClAM`** training and inference is linear in the number of points $|S|$ in the data set S, the number of dimensions d, and the number of clusters k. For N epochs over the data set S, **`ClAM`** training takes $O(dkTN|S|)$ where T is the AM recursion depth, while k-means training (with Lloyd's algorithm) takes $O(dkN|S|)$. Hence, both k-means and **`ClAM`** scale linearly with the number of points, though **`ClAM`** runtime has a multiplicative factor of T where T is usually less than 10. So k-means would intuitively be faster than **`ClAM`**. Note that k-means and **`ClAM`** are significantly more efficient than Spectral clustering and Agglomerative clustering which usually scale between quadratic and cubic in the number of samples. Note that, for the same number of epochs, we can establish convergence guarantees for our SGD based **`ClAM`** (which can be improved with the use of Momentum based SGD), while similar convergence guarantees are not available for k-means.

3.5. Flexibility and Extensions

Here, we highlight how **ClAM** naturally incorporates modifications. Specifically, we demonstrate how it can be modified to perform weighted clustering (different clusters have different "attractions"), and spherical clustering (using cosine similarity), highlighting the change in the energy function (equation 1) and the attractor dynamics (equation 4).

Weighted clustering. We allow different prototypes $\boldsymbol{\rho}_\mu$ to have different weights $\varepsilon_\mu > 0$, implying different attraction scaling, and the subsequent energy function is:

$$E(\boldsymbol{v}) = -\frac{1}{\beta}\log\left(\sum_{\mu\in[k]}\varepsilon_\mu \exp\left(-\beta\|\boldsymbol{\rho}_\mu - \boldsymbol{v}\|^2\right)\right) \quad (10)$$

with the finite difference update $\boldsymbol{\delta}^{t+1}$ at time-step $t+1$

$$\boldsymbol{\delta}^{t+1} = \frac{dt}{\tau}\sum_{\mu\in[k]}(\boldsymbol{\rho}_\mu - \boldsymbol{v}^t)\boldsymbol{\sigma}(-\beta\|\boldsymbol{\rho}_\mu - \boldsymbol{v}^t\|^2 + \log\varepsilon_\mu) \quad (11)$$

This update step can be directly plugged into algorithm 1 (line 11). The weights $\varepsilon_\mu, \mu \in [k]$ can be user-specified based on domain knowledge, or learned via gradient descent, and intuitively relate to the slopes of the basins of attraction, allowing different attraction scaling for each basin.

Spherical clustering. We can change the energy function to depend on the cosine similarity to get (without loss of generality, assume $\|\boldsymbol{v}\| = 1 \ \forall \boldsymbol{v} \in S$ in the following):

$$E(\boldsymbol{v}) = -\frac{1}{\beta}\log\left(\sum\nolimits_{\mu\in[k]}\exp\left(\beta\left\langle\tilde{\boldsymbol{\rho}_\mu}, \boldsymbol{v}\right\rangle\right)\right) \quad (12)$$

with $\tilde{\boldsymbol{\rho}}_\mu = \boldsymbol{\rho}_\mu / \|\boldsymbol{\rho}_\mu\|$, and the corresponding update step

$$\boldsymbol{\delta}^{t+1} = \frac{dt}{\tau}\sum\nolimits_{\mu\in[k]}\tilde{\boldsymbol{\rho}}_\mu\boldsymbol{\sigma}\left(\beta\left\langle\tilde{\boldsymbol{\rho}}_\mu, \boldsymbol{v}^t\right\rangle\right) \quad \tilde{\boldsymbol{v}} = \boldsymbol{v}^t + \boldsymbol{\delta}^{t+1}$$

and $\boldsymbol{v}^{t+1} = \tilde{v}/\|\tilde{v}\|$, plugged into algorithm 1 (line 11) gives us an end-to-end differentiable spherical clustering.

4. Empirical evaluation

We evaluate **ClAM** on 10 datasets of varying sizes – 7-16000 features, 101-60000 points. The number of clusters for each dataset are selected based on its number of underlying classes (the class information is not involved in clustering or hyperparameter selection). See details in Appendix A.1. First, we compare **ClAM** with established schemes, k-means (Lloyd, 1982), spectral & agglomerative clustering, and to DEC (Xie et al., 2016) and DCEC (Guo et al., 2017b) (for 3 image sets since DCEC uses CAE (§2); **ClAM** naively reshapes images into vectors). Then we ablate the effect of self-supervision in **ClAM**, and evaluate a **ClAM** extension. Qualitatively, we visualize the memory evolution through the learning.

For implementation, we use Tensorflow (Abadi et al., 2016) for **ClAM** and `scikit-learn` (Pedregosa et al., 2011) for the clustering baselines and quality metrics. DCEC/DEC seed the optimization with the prototypes from the Lloyd's solution; we also consider a randomly seeded DEC^r. Further details are in Appendix A.3. We perform an elaborate hyperparameter search for all methods (see Appendices B.1 and B.2), and utilize the configuration for each method corresponding to the best Silhouette Coefficient (SC) (Rousseeuw, 1987) on each dataset. The code is available at `https://github.com/bsaha205/clam`.

Q1: How does ClAM compare against baselines? We present the best SC obtained by all schemes in Table 1. **ClAM** consistently improves over k-means across all 10 datasets (up to 60%), and outperforms both versions of DEC, highlighting the advantage of the novel relaxation. Furthermore, Lloyd's seeding is critical in DEC – DEC^r does worse than base k-means (via Lloyd (1982)) in all cases – while ***ClAM*** *performs well with random seeding*. **ClAM** even improves upon DCEC, which uses rich image representations from a CAE. Overall, **ClAM** performs best on 5/10 datasets, showing significant improvements over even spectral and agglomerative clustering (these are not prototype-based, and hence can be more expressive). On the remaining datasets, both spectral and agglomerative clustering show improvements over **ClAM**. To understand this better, we also compare the clusters generated by the different methods to the ground-truth class structure in each of the datasets.[1] The Normalized Mutual Information (NMI) scores (Vinh et al., 2009) are shown in Table 2. The results indicate that the datasets on which spectral and agglomerative clustering have a significantly higher SC than k-means and **ClAM** (GCM, USPS, CTG, Segment), their corresponding NMI scores are significantly lower (see underlined entries in Table 2), indicating clusters that are misaligned with the ground-truth labels. Upon further investigation, we see that both spectral and agglomerative clustering end up with a single large cluster, and many really small (even singleton) ones, indicating that the clustering is overly influenced by geometric outliers. [2] See further results in Appendix B.4.

Q2: How beneficial is self-supervision in ClAM? In algorithm 1 (line 13), we use the pattern completion ability of AMs, and optimize for the self-supervised loss (equation 9). Here, we ablate the effect of this choice – we utilize a version of algorithm 1 that does not use any masking (re-

[1] Although the underlying geometry of the data does not necessarily align with the ground-truth class structure, this is a *post-hoc way* of evaluating how intuitive the discovered clusters are.

[2] For example, with CTG (2126 points), both spectral and agglomerative clustering find 10 clusters, but 8 of them of size ≤ 8 each, and 1 with size over 2000. k-means finds 1 cluster of size 7, with 8 of the remaining 9 clusters of size ≥ 91. **ClAM** finds 2 clusters with size ≤ 4; all remaining have size ≥ 55.

Table 1: **Silhoutte Coefficient (SC) obtained by ClAM, its variants and baselines (*higher is better*).** ▲(▼) show the performance gain (drop) by **ClAM**, given by $^{(a-b)}/_{b} \times 100\%$ where a is the SC for **ClAM**, and b is the SC for the baseline; positive (negative) values indicate performance gain (drop). The best performance for a dataset is in **boldface**. W/L denote Wins/Losses. Note ablations **ClAM** w/ (8) is **ClAM** without self-supervision, and **ClAM** w/ (11) is weighted **ClAM**.

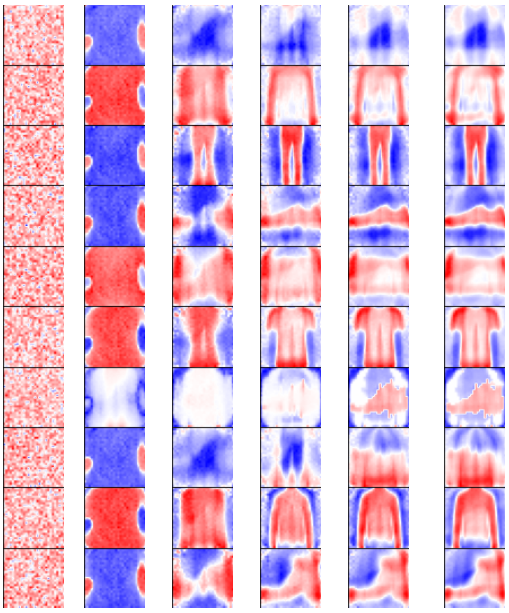
Dataset	k-means	Spectral	Agglo	DCEC	DEC	DEC^r	ClAM	w/ (8)	w/ (11)
Zoo	0.374 ▲10%	0.398 ▲3%	0.398 ▲3%	—	0.374 ▲10%	0.367 ▲12%	**0.412**	0.382	**0.412**
Ecoli	0.262 ▲26%	0.381 ▼13%	**0.404** ▼18%	—	0.262 ▲26%	0.255 ▲30%	0.331	0.301	*0.345*
MovLib	0.252 ▲3%	0.247 ▲5%	0.247 ▲5%	—	0.258 ▲1%	0.184 ▲41%	**0.260**	0.248	*0.262*
Yale	0.117 ▲10%	0.120 ▲8%	0.114 ▲13%	0.089 ▲45%	0.123 ▲5%	0.088 ▲47%	**0.129**	0.121	0.119
USPS	0.135 ▲1%	0.228 ▼40%	**0.231** ▼41%	0.119 ▲14%	0.135 ▲1%	0.108 ▲26%	0.136	0.127	0.136
Segment	0.357 ▲35%	**0.702** ▼31%	0.697 ▼31%	—	0.357 ▲35%	0.287 ▲68%	0.483	0.357	0.410
FMNIST	0.126 ▲10%	0.075 ▲84%	0.096 ▲44%	0.121 ▲14%	0.129 ▲7%	0.070 ▲97%	**0.138**	0.131	0.118
GCM	0.118 ▲62%	0.205 ▼7%	**0.288** ▼34%	—	0.115 ▲66%	0.054 ▲254%	0.191	0.142	0.156
MicePE	0.128 ▲56%	0.156 ▲28%	0.193 ▲4%	—	0.137 ▲46%	0.115 ▲74%	**0.201**	0.126	0.127
CTG	0.161 ▲53%	**0.449** ▼45%	0.387 ▼36%	—	0.164 ▲50%	0.130 ▲89%	0.246	0.147	0.158
ClAM W/L	10/0	5/5	5/5	3/0	10/0	10/0	—	10/0	6/2
ClAM w/ (8) W/L	5/3	3/7	3/7	3/0	5/4	10/0	0/10	—	2/8

Table 2: **Normalized Mutual Information (NMI) between ground-truth labels and clusters (*higher is better*).** The best performance for each dataset is in **boldface**. The underlined entries for **Spec**tral and **Aggl**omerative mark low NMI but high SC in Table 1 (note abbreviations k-means→k-m, DCEC→DC, DEC→D, DEC^r→D^r).

Data	k-m	Spec	Aggl	DC	D	D^r	ClAM
Zoo	0.83	0.89	0.84	N/A	0.83	0.80	**0.94**
Yale	0.60	0.57	**0.67**	0.54	0.55	0.51	0.64
GCM	0.44	0.17	0.19	N/A	0.42	0.39	**0.45**
Ecoli	0.63	0.66	**0.71**	N/A	0.63	0.57	0.66
MLib	0.60	0.61	0.61	N/A	0.60	0.51	**0.62**
MPE	0.24	0.01	0.26	N/A	0.29	0.30	**0.31**
USPS	0.54	0.08	0.02	**0.69**	0.54	0.45	0.56
CTG	**0.36**	0.04	0.04	N/A	0.35	**0.36**	0.32
Seg	0.58	0.01	0.01	N/A	0.59	**0.61**	0.55
FM	0.50	**0.64**	0.01	0.59	0.50	0.43	0.52

moving $\boldsymbol{m}$ and $\bar{\boldsymbol{m}}$ in lines 8 and 11 respectively) and learns memories by minimizing the loss in equation 8 (replacing line 13 with this loss). Table 1 (**ClAM** and **ClAM w/ (8)** columns) show their SC for all datasets, highlighting the positive effect of self-supervision, with around 10% gain in all cases.

Q3: How do weighted and vanilla clustering with ClAM compare against each other? We discussed the flexibility of the **ClAM** framework in §3.5. We evaluate one such extension – weighted clustering where different memories can have different relative attractions, inducing more imbalanced clustering which may be desirable in certain applications. **ClAM** handles relative weights that are (i) user-specified, or (ii) learned via SGD within algorithm 1 (with an additional update step for ε_μ). We consider case (ii) here, learning ε_μ via SGD, and compare to vanilla **ClAM** in Table 1 (**ClAM** and **ClAM w/ (11)** columns) on all datasets. The results indicate that the benefits are dataset-dependent. Our intent here is to show that **ClAM** handles such problems if needed, and can result in improvements (as in 2/10 cases).



(a) n_0 (b) n_5 (c) n_{10} (d) n_{20} (e) n_{50} (f) n_{100}

Figure 6: **Evolution of ClAM prototypes.** We visualize the prototypes at the n^{th} training epoch for $n = 0, 5, 10, 20, 50, 100$ (with $T = 10$). The images pixels are colored as red for positive, white for zero and blue for negative values, with their intensity denoting magnitude.

Q4: How to interpret ClAM? An interesting aspect of AMs is the prototype-based representation of memories (Krotov & Hopfield, 2016). We study this for **ClAM** with the Fashion-MNIST images. We use **ClAM** to partition the 60k images of fashion items (shirts, trousers, shoes, bags) into 10 clusters, and visualize the evolution of the 10 memories in figure 6 over the course of algorithm 1. We reshape

Table 3: **Clustering efficiency comparison between k-means and `ClAM`.**

Dataset	Silhoutte Coefficient (SC)		Time in second	
	k-means	`ClAM`	k-means	`ClAM`
Zoo	0.374	0.412	2	18
Ecoli	0.262	0.331	4	47
MovLib	0.252	0.260	6	67
Yale	0.117	0.129	9	88
CIFAR-10	0.022	0.031	2995	3419
JV	0.109	0.118	1973	3933

the 28×28 images into vectors in $\mathbb{R}^{784}$, learn prototypes in $\mathbb{R}^{784}$ and reshape them back to 28×28 solely for visualization. Each sub-figure in figure 6 corresponds to a particular epoch during the training, and shows all 10 memories stacked vertically. We can see that at epoch 0 (figure 6a) the memories are initialized with random values. At epoch 5 (figure 6b), the memories start to become less random, but only start showing some discernible patterns at epoch 10 (figure 6c) which get refined by epoch 20 (figure 6d) and further sharpen by epoch 50 (figure 6e). By epoch 100 (figure 6f), the shapes have stabilized although some learning might still be happening. Some memories evolve into individual forms (such as rows 3–pants, 4–shoes, 8–bags, 9–long-sleeve shirts and 10–boots), while others evolve into mixtures of 2 forms (such as rows 1–pants and shoes, 6–pants and short-sleeve shirts, 7–shoes and long-sleeve shirts). The remaining evolve into less distinguishable forms though we can still visualize some shapes (like in rows 2–long-sleeve shirts and 5–shoes).

Q5: How computationally expensive is `ClAM` compared to k-means? While comparing the actual runtimes of k-means and **`ClAM`**, we would like to note that we make use of the standard k-means implementation from scikit-learn, which utilizes numpy and scipy linear algebra packages under the hood, while our **`ClAM`** implementation makes use of the Tensorflow framework, leveraging their auto-grad capabilities. Furthermore, Tensorflow is designed to seamlessly utilize GPUs if available, while scikit-learn does not have such capabilities out of the box. With these differences, the actual runtimes are not directly comparable. This is the reason why we give importance to runtime complexities instead of actual runtimes. However, notwithstanding these caveats, we present the precise runtimes of k-means and **`ClAM`** (the Spectral and Agglomerative clustering baselines are significantly slower, and DEC has the same scaling as k-means for the same number of epochs) for six datasets. In Table 3, we present the Silhouette Coefficient (SC) and Training time for k-means (with Lloyd's algorithm) and **`ClAM`** for all of these six datasets. In all cases, **`ClAM`** improves over k-means in terms of the SC. For the smaller datasets, we can see that k-means is almost an order of magnitude faster than **`ClAM`**. However, a lot of it can be attributed to the overhead of leveraging the Tensorflow framework and SGD, instead of just performing small matrix multiplications with Numpy. We also consider a relatively larger dataset CIFAR-10 where the images are encoded into 1920-dimensional vectors with a pre-trained DenseNet (Huang et al., 2017) and a million row dataset, Japanese Vowels (JV), from `https://openml.org`. In these larger datasets, k-means is still faster than **`ClAM`** (as we had discussed earlier based on the theoretical runtime complexities) but the increase of **`ClAM`** 's runtime is only between 1-2$\times$.

5. Limitations and Future Work

In this paper, we present a novel continuous relaxation of the discrete prototype-based clustering problem leveraging the AM dynamical system that allows us to solve the clustering problem with SGD based schemes. We show that this relaxation better matches the essence of the discrete clustering problem, and empirical results show that our **`ClAM`** approach significantly improves over standard prototype-based clustering schemes, and existing continuous relaxations. We note that **`ClAM`** is still a prototype-based clustering scheme, hence inherits the limitations of prototype-based clustering.

Given the end-to-end differentiable nature of **`ClAM`**, we will extend it to clustering with kernel similarity functions and Mahanalobis distances, and to deep clustering where we also learn a latent space. We plan to explore new energy functions and update dynamics that enable spectral clustering. Finally, given **`ClAM`**'s flexibility, we want to automatically estimate the number of clusters on a per-dataset basis, much like Pelleg & Moore (2000) and Hamerly & Elkan (2003).

Acknowledgements

This work was supported by the Rensselaer-IBM AI Research Collaboration (`http://airc.rpi.edu`), a part of the IBM AI Horizons Network. The work was done during Bishwajit Saha's externship at MIT-IBM Watson AI Lab, IBM Research.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

Aguilar, J. S., Ruiz, R., Riquelme, J. C., and Giráldez, R. SNN: A supervised clustering algorithm. In *International Conference on Industrial, Engineering and Other Ap-*

plications of Applied Intelligent Systems, pp. 207–216. Springer, 2001.

Amit, D. J., Gutfreund, H., and Sompolinsky, H. Storing infinite numbers of patterns in a spin-glass model of neural networks. *Physical Review Letters*, 55(14):1530, 1985.

Aurenhammer, F. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

Bezdek, J. C., Ehrlich, R., and Full, W. Fcm: The fuzzy c-means clustering algorithm. *Computers & geosciences*, 10(2-3):191–203, 1984.

Bijuraj, L. Clustering and its applications. In *Proceedings of National Conference on New Horizons in IT-NCNHIT*, volume 169, pp. 172, 2013.

Burns, T. F. and Fukai, T. Simplicial hopfield networks. In *The Eleventh International Conference on Learning Representations*, 2023.

Cai, J., Wang, S., Xu, C., and Guo, W. Unsupervised deep clustering via contractive feature representation and focal loss. *Pattern Recognition*, 123:108386, 2022.

Chakraborty, S., Paul, D., and Das, S. Automated clustering of high-dimensional data with a feature weighted mean shift algorithm. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 6930–6938, 2021.

Chazan, S. E., Gannot, S., and Goldberger, J. Deep clustering based on a mixture of autoencoders. In *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6. IEEE, 2019.

Curtin, R., March, W., Ram, P., Anderson, D., Gray, A., and Isbell, C. Tree-independent dual-tree algorithms. In *International Conference on Machine Learning*, pp. 1435–1443. PMLR, 2013.

Curtin, R. R., Lee, D., March, W. B., and Ram, P. Plug-and-play dual-tree algorithm runtime analysis. *Journal of Machine Learning Research*, 16:3269–3297, 2015.

Dasgupta, S. *The hardness of k-means clustering*. Department of Computer Science and Engineering, University of California . . . , 2008.

Demircigil, M., Heusel, J., Löwe, M., Upgang, S., and Vermet, F. On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168(2): 288–299, 2017.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Donath, W. E. and Hoffman, A. J. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973. doi: 10.1147/rd.175.0420.

Dua, D., Graff, C., et al. Uci machine learning repository. 2017.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226–231, 1996.

Fang, C., Li, C. J., Lin, Z., and Zhang, T. Spider: Near-optimal non-convex optimization via stochastic path-integrated differential estimator. *Advances in Neural Information Processing Systems*, 31, 2018.

Guo, X., Gao, L., Liu, X., and Yin, J. Improved deep embedded clustering with local structure preservation. In *Ijcai*, pp. 1753–1759, 2017a.

Guo, X., Liu, X., Zhu, E., and Yin, J. Deep clustering with convolutional autoencoders. In *International conference on neural information processing*, pp. 373–382. Springer, 2017b.

Hamerly, G. and Elkan, C. Learning the k in k-means. *Advances in neural information processing systems*, 16, 2003.

Hardt, M., Recht, B., and Singer, Y. Train faster, generalize better: Stability of stochastic gradient descent. In *International conference on machine learning*, pp. 1225–1234. PMLR, 2016.

Hoover, B., Chau, D. H., Strobelt, H., and Krotov, D. A universal abstraction for hierarchical hopfield networks. In *The Symbiosis of Deep Learning and Differential Equations II*, 2022.

Hoover, B., Liang, Y., Pham, B., Panda, R., Strobelt, H., Chau, D. H., Zaki, M. J., and Krotov, D. Energy transformer. *arXiv preprint arXiv:2302.07253*, 2023.

Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Hopfield, J. J. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10): 3088–3092, 1984.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Hubert, L. and Arabie, P. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

Hull, J. J. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

Ji, P., Zhang, T., Li, H., Salzmann, M., and Reid, I. Deep subspace clustering networks. *Advances in neural information processing systems*, 30, 2017.

Johnson, S. C. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

Kim, J. and Park, H. Sparse nonnegative matrix factorization for clustering. Technical report, Georgia Institute of Technology, 2008. URL https://faculty.cc.gatech.edu/~hpark/papers/GT-CSE-08-01.pdf.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krotov, D. Hierarchical associative memory. *arXiv preprint arXiv:2107.06446*, 2021.

Krotov, D. A new frontier for hopfield networks. *Nature Reviews Physics*, pp. 1–2, 2023.

Krotov, D. and Hopfield, J. J. Dense associative memory for pattern recognition. *Advances in neural information processing systems*, 29, 2016.

Krotov, D. and Hopfield, J. J. Large associative memory problem in neurobiology and machine learning. In *International Conference on Learning Representations*, 2021.

Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., and Liu, H. Feature selection: A data perspective. *ACM computing surveys (CSUR)*, 50(6):1–45, 2017.

Liang, Y., Krotov, D., and Zaki, M. J. Modern hopfield networks for graph embedding. *Frontiers in big Data*, 5, 2022.

Lloyd, S. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

Lucibello, C. and Mézard, M. The exponential capacity of dense associative memories. *arXiv preprint arXiv:2304.14964*, 2023.

MacQueen, J. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pp. 281–297, 1967.

March, W. B., Ram, P., and Gray, A. G. Fast euclidean minimum spanning tree: algorithm, analysis, and applications. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 603–612, 2010.

McEliece, R., Posner, E., Rodemich, E., and Venkatesh, S. The capacity of the hopfield associative memory. *IEEE transactions on Information Theory*, 33(4):461–482, 1987.

Millidge, B., Salvatori, T., Song, Y., Lukasiewicz, T., and Bogacz, R. Universal hopfield networks: A general framework for single-shot associative memory models. *arXiv preprint arXiv:2202.04557*, 2022.

Nemirovski, A., Juditsky, A., Lan, G., and Shapiro, A. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

Nesterov, Y. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

Panahi, A., Dubhashi, D., Johansson, F. D., and Bhattacharyya, C. Clustering by sum of norms: Stochastic incremental algorithm, convergence and cluster recovery. In *International conference on machine learning*, pp. 2769–2777. PMLR, 2017. URL http://proceedings.mlr.press/v70/panahi17a/panahi17a.pdf.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Pelleg, D. and Moore, A. W. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, volume 1, pp. 727–734, 2000.

Peng, X., Xiao, S., Feng, J., Yau, W.-Y., and Yi, Z. Deep subspace clustering with sparsity prior. In *IJCAI*, pp. 1925–1931, 2016.

Ram, P. and Gray, A. G. Density estimation trees. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 627–635, 2011.

Ram, P., Lee, D., March, W., and Gray, A. Linear-time algorithms for pairwise statistical problems. *Advances in Neural Information Processing Systems*, 22, 2009.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., et al. Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*, 2020.

Ren, Y., Pu, J., Yang, Z., Xu, J., Li, G., Pu, X., Yu, P. S., and He, L. Deep clustering: A comprehensive survey. *arXiv preprint arXiv:2210.04142*, 2022.

Reynolds, D. A. Gaussian mixture models. *Encyclopedia of biometrics*, 741(659-663), 2009.

Rousseeuw, P. J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

Sander, J., Ester, M., Kriegel, H.-P., and Xu, X. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data mining and knowledge discovery*, 2(2):169–194, 1998.

Saxena, A., Prasad, M., Gupta, A., Bharill, N., Patel, O. P., Tiwari, A., Er, M. J., Ding, W., and Lin, C.-T. A review of clustering techniques and developments. *Neurocomputing*, 267:664–681, 2017.

Sibson, R. Slink: an optimally efficient algorithm for the single-link cluster method. *The computer journal*, 16(1): 30–34, 1973.

Silverman, B. W. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

Song, C., Liu, F., Huang, Y., Wang, L., and Tan, T. Auto-encoder based data clustering. In *Iberoamerican congress on pattern recognition*, pp. 117–124. Springer, 2013.

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Vassilvitskii, S. and Arthur, D. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, 2006.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vinh, N. X., Epps, J., and Bailey, J. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th annual international conference on machine learning*, pp. 1073–1080, 2009.

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

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

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

Xu, R. and Wunsch, D. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *international conference on machine learning*, pp. 3861–3870. PMLR, 2017.

Zhang, J., Li, C.-G., You, C., Qi, X., Zhang, H., Guo, J., and Lin, Z. Self-supervised convolutional subspace clustering network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5473–5482, 2019.

Zhou, S., Xu, H., Zheng, Z., Chen, J., Bu, J., Wu, J., Wang, X., Zhu, W., Ester, M., et al. A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions. *arXiv preprint arXiv:2206.07579*, 2022.

A. Experimental Details

A.1. Dataset details

To evaluate **ClAM**, we conducted our experiments on ten standard benchmark data sets. The datasets are taken from various sources such as Yale from ASU feature selection repository[3] (Li et al., 2017), USPS from Kaggle[4] (Hull, 1994), Fashion-MNIST from Zalando[5] (Xiao et al., 2017), GCM from Chakraborty et al. (2021) and the rest of the datasets from the UCI machine learning repository[6] (Dua et al., 2017). The statistics of datasets used in our experiment are given in Table 4.

Table 4: Descriptions of various benchmark datasets, used in our experiments.

Dataset	Short name	# Points	# Features	# Classes
Zoo	Zoo	101	16	7
Yale	Yale	165	1024	15
GCM	GCM	190	16063	14
Ecoli	Ecoli	336	7	8
Movement Libras	MovLib	360	90	15
Mice Protien Expression	MicePE	1080	77	8
USPS	USPS	2007	256	10
CTG	CTG	2126	21	10
Segment	Segment	2310	19	7
Fashion MNIST	FMNIST	60000	784	10

A.2. Metrics used

To evaluate the performance of **ClAM**, we use Silhouette Coefficient (SC) (Rousseeuw, 1987) as the unsupervised metric that is used to measure the quality of clustering. The score is between -1 and 1; a value of 1 indicates perfect clustering while a value of -1 indicates entirely incorrect clustering labels. A value of near 0 indicates that there exist overlapping clusters in the partition. To observe which existing clustering scheme is **ClAM** most similar to, we use Normalized Mutual Information (NMI) (Vinh et al., 2009) & Adjusted Rand Index (ARI) (Hubert & Arabie, 1985) between the obtained partition by **ClAM** and obtained partition by baselines. For NMI, a value of 1 indicates perfect clustering while a value of 0 indicates completely wrong class labels. ARI scores range between -1 and 1 and the interpretation is same as SC. We also use the NMI & ARI scores between the ground truth and the obtained partition from **ClAM** and the baselines to measure how they are aligned to true clustering labels.

A.3. Implementation Details

We use Tensorflow (Abadi et al., 2016) numerical machine learning library to implement and evaluate our model. We train **ClAM** on a single node with 1 NVIDIA Geforce RTX 3090 (24GB RAM), and 8-core 3.5GHz Intel Core-i9 CPUs (32GB RAM). We train on the original data with masking where we utilize the distortion between the ground-truth masked value and completed pattern from **ClAM** as the loss function to find the best model. In the forward pass, in each step, the feature vector is updated in such a way so that gradually it moves toward one of the stored memories. In the backward path, the memories are learned to minimize the loss. Hyperparameters are tuned for each dataset to find the best result. Full details of the hyperparameters used in our model are given in Table 5. For the baseline schemes of k-means, spectral, and agglomerative, we use the implementation from `scikit-learn` (Pedregosa et al., 2011) library and tune different hyperparameters to get the best results for each dataset. For DCEC (Guo et al., 2017b), as it is based on convolutional autoencoders (CAE) and works with only image dataset, we evaluate on three image datasets to compare with **ClAM** (we leverage their Tensorflow implementation[7]). For the soft-clustering part of DEC (Xie et al., 2016) (where they utilize KL

[3] http://featureselection.asu.edu/
[4] https://www.kaggle.com/datasets/bistaumanga/usps-dataset
[5] https://github.com/zalandoresearch/fashion-mnist
[6] https://archive.ics.uci.edu/ml/index.php
[7] https://github.com/XifengGuo/DCEC

divergence loss between predicted and target probability distribution), besides their k-means-initialized cluster centers, we also employ random-initialized cluster centers (DEC^r) to study how randomization works in DEC soft clustering network. The description of used hyperparameters and their roles in the baseline schemes are given in Table 7.

Table 5: Hyperparameters, their roles and range of values for **ClAM**.

Hyperparameter	Used Values
Inverse temperature, β	[10^{-5} - 5]
Number of layers, $T = {}^{1}/_{\alpha} = {}^{\tau}/_{dt}$	[2-20]
Batch size	[8, 16, 32, 64, 128, 256]
Adam initial learning rate, ϵ	[10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]
Reduce LR by factor	0.8
Reduce LR patience (epochs)	5
Minimum LR	10^{-5}
Reduce LR loss threshold	10^{-3}
Maximum Number of epochs	200
Number of restart	10
Mask probability	[0.1, 0.12, 0.15, 0.2, 0.25, 0.3]
Mask value	[‘mean’, ‘min’, ‘max’]

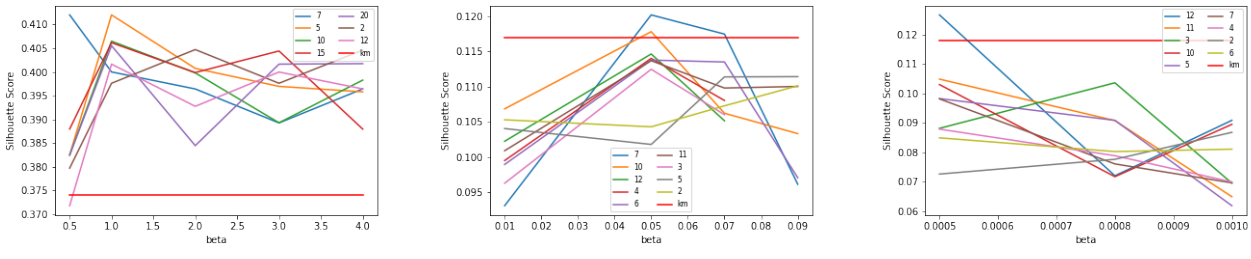
Table 6: Best hyperparameters for different datasets for **ClAM**.

Dataset	Inverse temperature, β	Layers, T	Initial learning rate	Batch size	Mask probability	Mask value
Zoo	2.4	10	0.1	8	0.2	’mean’
Yale	0.06	10	0.1	8	0.15	’mean’
GCM	0.0004	12	0.1	8	0.15	’max’
Ecoli	0.095	12	0.1	16	0.15	’mean’
MLib	0.7	7	0.01	8	0.15	’mean’
MPE	0.1	5	0.2	8	0.15	’mean’
USPS	0.1	5	0.001	16	0.15	’min’
CTG	0.4	12	0.1	8	0.1	’mean’
Segment	0.1	7	0.1	8	0.1	’mean’
FMNIST	0.0005	5	0.01	256	0.1	’max’

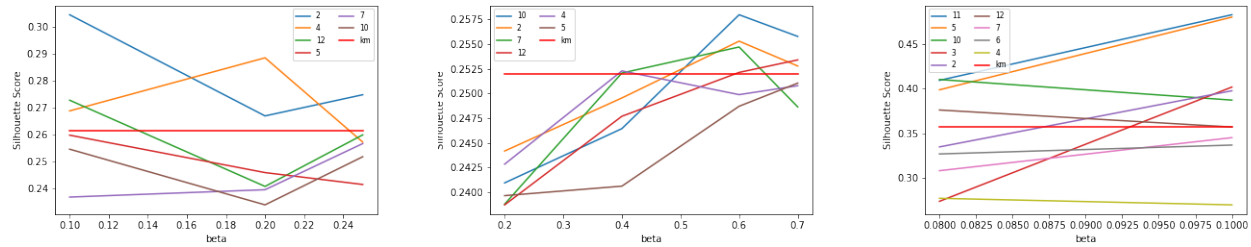
B. Additional Experimental Results

B.1. Hyperparameter Dependency for ClAM

To get the best result from **ClAM**, we tune the involved hyperparameters (Table 5) thoroughly. We use a range of $[10^{-5} - 2]$ for the inverse temperature β, which is the most critical hyperparameter for **ClAM**. Figure 7 shows the effect of β in measuring Silhouette Coefficient (SC) (Rousseeuw, 1987) for six datasets where each different plot indicates different number of steps (T) used in **ClAM**. While SC is greatly dependent on the inverse temperature β and steps T, we see that it is persistently competitive to the baseline k-means (red plot) for all the different configurations. We use the Adam optimizer and start with an initial learning rate, and we reduce the learning rate by a factor of 0.8 if the training loss does not ameliorate for a specific number of epochs until it reaches to the minimum learning rate threshold (10^{-5}). The effect of initial learning rate on **ClAM** is shown in figure 8. We set the number of epochs to 200 for each hyperparameter configuration, with number of restarts at 10 (with different random seeds), and keep track of the training loss at the end of each epoch. We pick the set of hyperparameters and the related model for the inference step which produces the least training loss. For masking the original data, we tune different mask probabilities $[0.1 - 0.3]$ for different datasets to obtain the best model with three different mask values (’mean’, ’min’, ’max’) for each feature. Figure 9 and figure 10 depict the effect of mask probabilities and mask values on **ClAM**, respectively. We can see that using ’mean’ value of each feature as the mask value gives the nest results for almost all datasets. Table 6 shows the best hyperparameters values for different datasets used in **ClAM**.

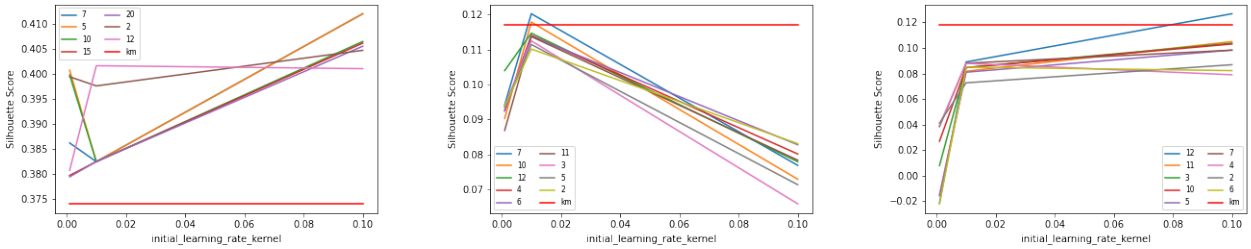


(a) Zoo (b) Yale (c) GCM

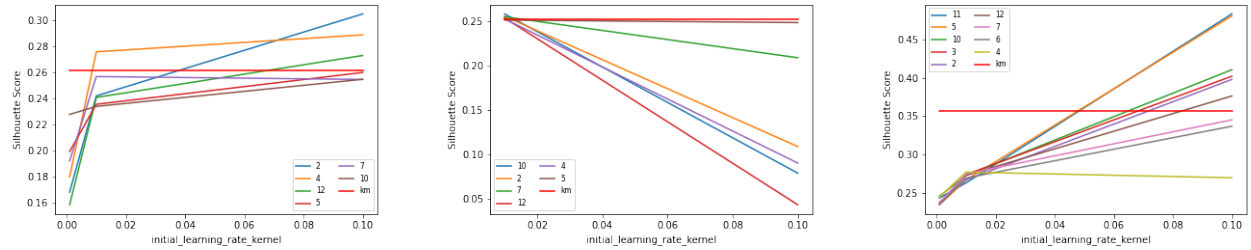


(d) Ecoli (e) Movement Libras (f) Segment

Figure 7: **Silhouette score vs inverse temperature (β) for six datasets and for different number of steps, T.** Label km refers to k-means.



(a) Zoo (b) Yale (c) GCM

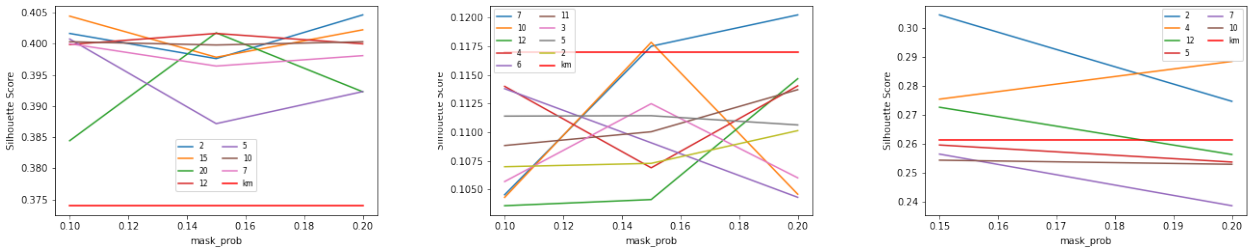


(d) Ecoli (e) Movement Libras (f) Segment

Figure 8: **Silhouette score vs initial learning rate for six datasets and for different number of steps, T.** Label km refers to k-means.

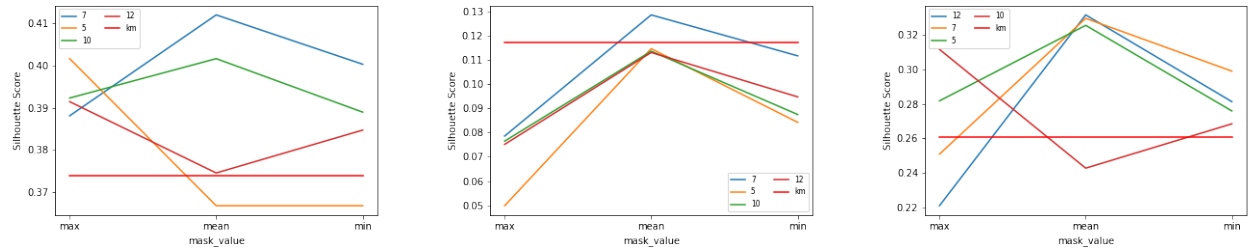
B.2. Hyperparameter Tuning for Baselines

We compare **ClAM** with three baseline clustering schemes (k-means, spectral, and agglomerative) from `scikit-learn` (Pedregosa et al., 2011), DCEC (Guo et al., 2017b) & soft DEC (Xie et al., 2016). For k-means, spectral and agglomerative, we perform a comprehensive search for tuning different hyperparameters available in `scikit-learn` and pick the best results. For DCEC and soft DEC, we use their suggested hyper-parameters with different update intervals



(a) Zoo (b) Yale (c) Ecoli

Figure 9: **Silhouette score vs mask probability for three datasets and for different number of steps,** T**.** km means k-means.



(a) Zoo (b) Yale (c) Ecoli

Figure 10: **Silhouette score vs mask values for three datasets and for different number of steps,** T**.** km is k-means.

(interval at which the predicted and target distributions get updated) to find the best results. Table 7 provides a brief description of the hyperparameters and their roles in the baseline schemes.

Table 7: Hyperparameters (HPs), their roles and range of values for the baseline clustering schemes.

Baseline	**HP**	**Role**	**Used Values**
k-means	n_clusters	Number of clusters to be formed	True number of clusters
	init	Initialization method	['k-means++', 'random']
	n_init	Number of time the k-means algorithm will be run	1000
Spectral	n_clusters	Number of clusters to be formed	True number of clusters
	affinity	The mechanism of constructing the affinity matrix	'nearest_neighbors', 'rbf'
	gamma	Kernel coefficient for 'rbf', ignored for 'nearest_neighbors'	[0.001, 0.01, 0.05, 0.1, 0.5, 0.75, 1, 2, 5, 10]
	assign_labels	Mechanism for assigning labels in the embedding space	['k-means', 'discretize']
	n_neighbors	Number of neighbors to consider when constructing the affinity matrix	[10, 15, 20, 50]
	n_init	Number of time the k-means algorithm will be run	1000
Agglomerative	n_clusters	Number of clusters to be formed	True number of clusters
	affinity	Metric used to compute the linkage	['euclidean', 'l1', 'l2', 'manhattan', 'cosine']
	linkage	Linkage criterion to use	['single', 'average', 'complete', 'ward']
DCEC/DEC/DEC^r	n_clusters	Number of clusters to be formed	True number of clusters
	batch_size	Size of each batch	256
	maxiter	Maximum number of iteration	2e4
	gamma	Degree of freedom of student's t-distribution	1
	update_interval	Interval at which the predicted and target distributions are updated	[1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 75, 100, 125, 140, 150, 200]
	tol	Tolerance rate	0.001

B.3. Which clustering baseline is ClAM most similar to?

We compare the "overlap" – the normalized mutual information (Vinh et al., 2009) and adjusted rand index (Hubert & Arabie, 1985) – between the clusterings produced by **ClAM** and the different baselines in Table 8. As is evident from the results, the clusterings generated by **ClAM** are significantly more aligned with those generated by k-means than the other baselines (with significant differences in 4/10 datasets). This is somewhat expected since both k-means and **ClAM** are minimizing some form of the mean-squared-error (equivalently, the squared Euclidean distance), albeit in a different manner (alternating combinatorial optimization in k-means versus gradient descent based optimization in **ClAM**).

Table 8: NMI and ARI score comparison between the clusters found by **ClAM** and the baseline clustering schemes.

Dataset	Metric	k-means	Spectral	Agglomerative
Zoo	NMI	**0.8808**	0.8315	0.8146
	ARI	**0.7736**	0.6691	0.6953
Yale	NMI	**0.8846**	0.8568	0.8840
	ARI	**0.7870**	0.7308	0.7621
GCM	NMI	**0.4402**	0.1400	0.1487
	ARI	**0.1832**	-0.0223	-0.0314
Ecoli	NMI	0.7448	0.7378	**0.7718**
	ARI	0.5868	0.7381	**0.8049**
Movement Libras	NMI	**0.9158**	0.7918	0.8426
	ARI	**0.8110**	0.6212	0.7009
Mice Protien Exp	NMI	**0.4671**	0.0043	0.4278
	ARI	**0.3652**	0.0003	0.2960
USPS	NMI	**0.8150**	0.1163	0.0227
	ARI	**0.7869**	0.0136	0.0023
CTG	NMI	**0.5335**	0.0449	0.0514
	ARI	**0.3588**	0.0106	0.0126
Segment	NMI	**0.6359**	0.0084	0.0.0070
	ARI	**0.5484**	-0.0013	-0.0010
FMNIST	NMI	**0.6391**	0.5469	0.0023
	ARI	**0.5182**	0.4251	0.0021

B.4. Complete comparison to baselines in terms of ground truth

Table 9 is a detailed version of Table 2, which shows the complete NMI and ARI score comparison between **ClAM** and the baseline clustering schemes in terms of ground truth. From the table, we see that **ClAM** not only performs well in terms of Silhouette Coefficient (SC), it performs equally well in terms of ground truth.

B.5. Comprehensive experiment on basins of attraction of ClAM vs Voronoi partition

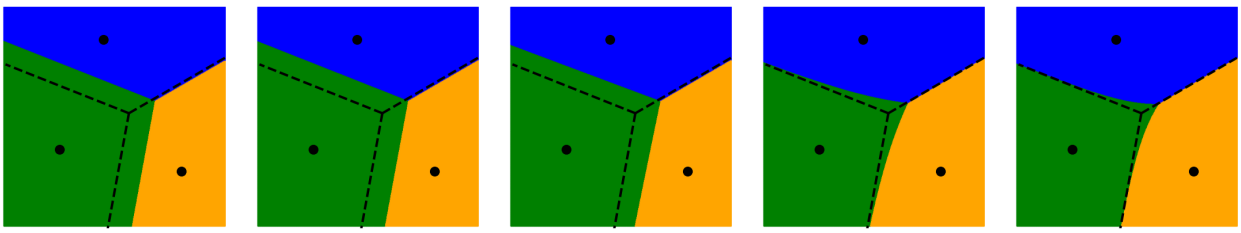
Figure 11 (for three clusters) and Figure 12 (for five clusters) represent more comprehensive versions of experiments described in figure 4 to understand the evolution of AM basins of attraction to voronoi tesselation from low β value (0.001) to high β value (100) for step (T) 10. From the figures, we see that starting with a completely non-voronoi partition $(\beta = 0.001)$, the basins of attraction of **ClAM** follows the voronoi tesselation with some interesting non-linear characteristics $(\beta = 10 - 30)$, and then matches voronoi gradually $(\beta = 50 - 100)$. The experiment strongly indicates that **ClAM** is equally able to find non-linear boundaries between the data points to find more compact partitions.

B.6. Comprehensive experiment on elongated shaped clusters

Figure 13 represents the further experiment, similar to that figure 1, for three elongated shaped clusters. Here, we can see **ClAM** is able to find all three clusters in nearly perfect way where the baseline algorithms struggle to find the right partitions.

Table 9: NMI and ARI score comparison among **ClAM** and the baseline clustering schemes in terms of ground truth.

Dataset	Metric	k-means	Spectral	Agglomerative	DCEC	DEC	DEC^r	ClAM
Zoo	NMI	0.8330	0.8891	0.8429	N/A	0.8330	0.7992	**0.9429**
	ARI	0.7373	0.8664	0.9109	N/A	0.7373	0.6755	**0.9642**
Yale	NMI	0.6034	0.5744	**0.6723**	0.5428	0.5499	0.5068	0.6418
	ARI	0.3644	0.3226	**0.4567**	0.2817	0.2920	0.2398	0.4230
GCM	NMI	0.4385	0.1715	0.1882	N/A	0.4228	0.3923	**0.4476**
	ARI	0.1308	-0.0119	-0.0071	N/A	0.1188	0.0880	**0.2492**
Ecoli	NMI	0.6332	0.6606	**0.7111**	N/A	0.6332	0.5707	0.6633
	ARI	0.4997	0.6505	**0.7261**	N/A	0.4997	0.3893	0.7027
Movement Libras	NMI	0.6044	0.6118	0.6086	N/A	0.5959	0.3147	**0.6142**
	ARI	0.3260	0.3236	0.3144	N/A	0.3147	0.2223	**0.3351**
Mice Protien Exp	NMI	0.2373	0.0056	0.2596	N/A	0.2873	0.2951	**0.3108**
	ARI	0.1260	0.0019	0.1558	N/A	0.1756	0.1796	**0.1652**
USPS	NMI	0.5368	0.0777	0.0180	**0.6961**	0.5376	0.4538	0.5566
	ARI	0.4304	-0.0033	0.0002	**0.5907**	0.4306	0.3226	0.4537
CTG	NMI	0.3581	0.0391	0.0419	N/A	0.3507	**0.3587**	0.3154
	ARI	0.1780	0.0060	0.0078	N/A	0.1766	**0.1818**	0.1736
Segment	NMI	0.5846	0.0102	0.0085	N/A	0.5853	**0.6102**	0.5489
	ARI	0.4607	0.0005	0.0003	N/A	0.4612	**0.5038**	0.4331
FMNIST	NMI	0.5036	**0.6429**	0.0051	0.5948	0.5008	0.3339	0.5183
	ARI	0.3461	**0.4307**	0.0005	0.4113	0.3369	0.2279	0.3665



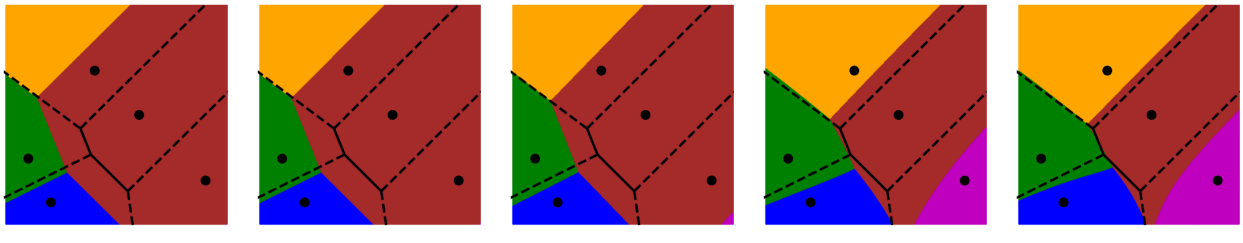
(a) $\beta = 0.001$. (b) $\beta = 0.1$. (c) $\beta = 1$. (d) $\beta = 10$. (e) $\beta = 15$.

(f) $\beta = 20$. (g) $\beta = 25$. (h) $\beta = 30$. (i) $\beta = 50$. (j) $\beta = 100$.

Figure 11: **Basins of attraction vs Voronoi partition for three clusters.** Partitions induced by AM basins of attraction with given memories (black dots) for different β are shown by the colored regions (T=10). Dashed lines show the Voronoi partition.

B.7. Histogram of entropy of involved datasets

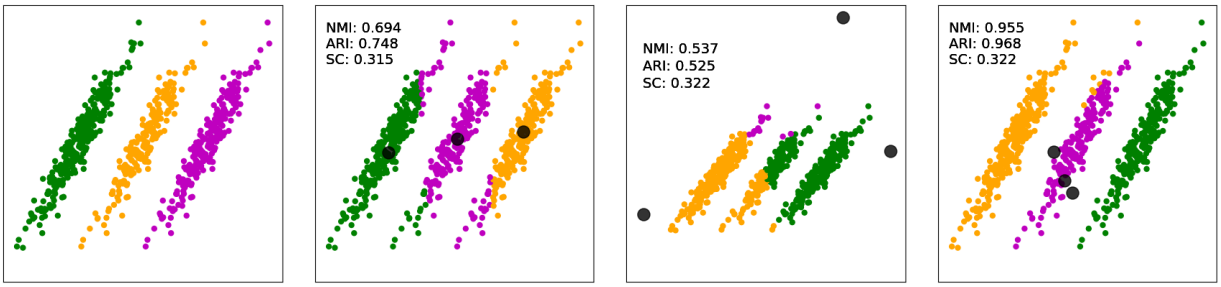
Figure 14 shows the histograms of entropy for each of the the ten datasets used to evaluate **ClAM**. Equation 13 denotes how entropy is calculated on every data point $\boldsymbol{x}$ in the dataset where we use the optimum β (from Table 6) and learned memories



(a) $\beta = 0.001$. (b) $\beta = 0.1$. (c) $\beta = 1$. (d) $\beta = 10$. (e) $\beta = 15$.

(f) $\beta = 20$. (g) $\beta = 25$. (h) $\beta = 30$. (i) $\beta = 50$. (j) $\beta = 100$.

Figure 12: **Basins of attraction vs Voronoi partition for five clusters.** Partitions induced by AM basins of attraction with given memories (black dots) for different β are shown by the colored regions (T=10). Dashed lines show the Voronoi partition.



(a) True (b) Lloyd's (c) DEC (d) **ClAM**

Figure 13: **Clustering with ClAM.** Three clusters (figure 13a), and solutions found by k-means (Lloyd, 1982) (figure 13b), DEC relaxation (Xie et al., 2016)(figure 13c), and our proposed end-to-end differentiable SGD-based **ClAM** (figure 13d). The black dots indicate the learned prototypes. ***ClAM*** *discovers the ground-truth clusters while the baselines cannot.*

($\boldsymbol{\rho}_\mu$) after training for each dataset.

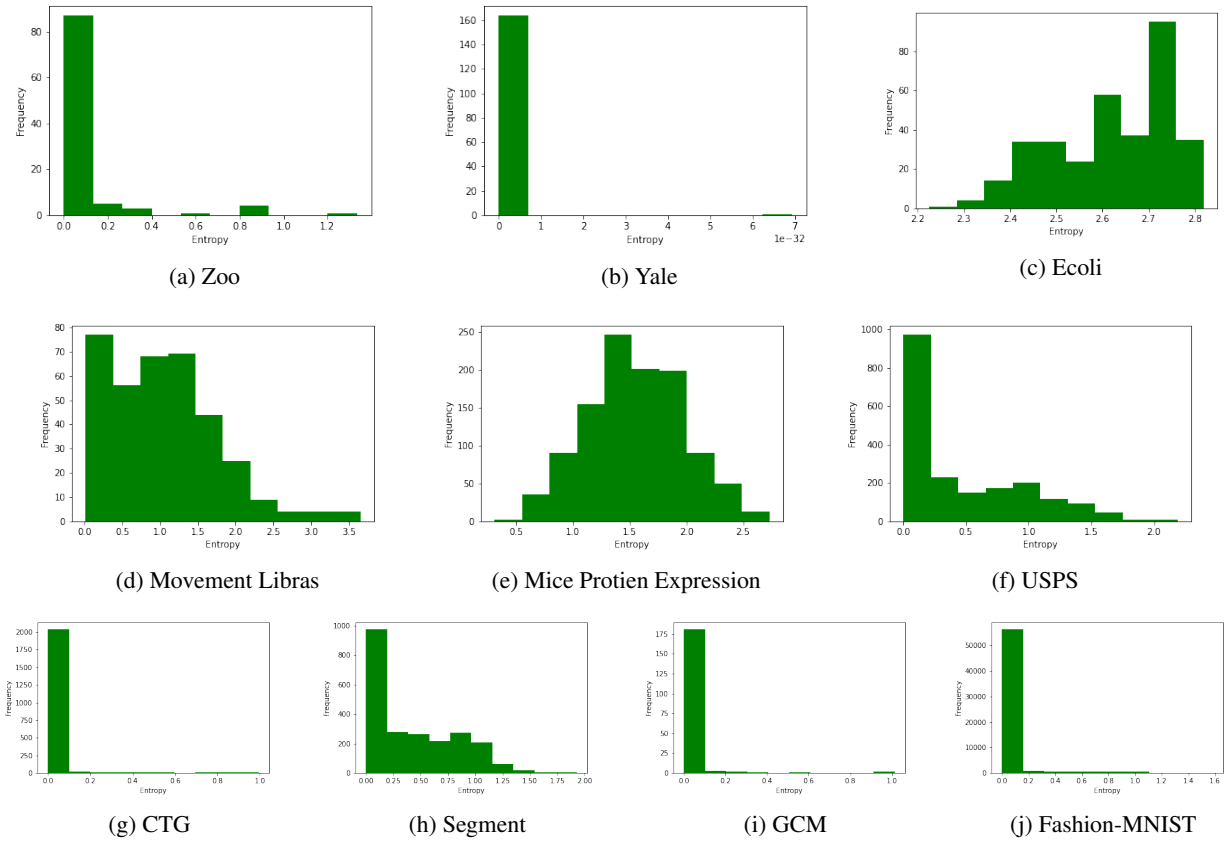
$$H(\boldsymbol{x}) = -\sum_{\mu\in[M]} \mathbf{p}_\mu(\mathbf{x}) \log(\mathbf{p}_\mu(\mathbf{x})) \qquad \text{where} \quad \mathbf{p}_\mu(\mathbf{x}) = \boldsymbol{\sigma}(-\beta\|\boldsymbol{\rho}_\mu - \mathbf{x}\|^2) \tag{13}$$

These histograms represent the sharpness of the softmax function that explains how much the partition found by **ClAM** is like a Voronoi partition where a value of zero means it always assigns the points to its closest memories and exactly matches with Voronoi.

B.8. How does ClAM handle noise in the input data?

Observe that **ClAM** considers the same objective function as the traditional k-means combinatorial optimization problem, and utilizes a new novel relaxation that allows us to solve the problem in an end-to-end differentiable manner in place of the well-established iterative discrete Lloyd's algorithm. Therefore, robustness of **ClAM** would be tied to the robustness of the original k-means objective to noise, and will of course depend on the nature of the noise.

One relevant form of noise in the context of clustering is the presence of outliers, and it is known that k-means is generally sensitive to outliers given that the k-means objective is effectively penalizing variance, which is sensitive to outliers. Spectral



(a) Zoo (b) Yale (c) Ecoli

(d) Movement Libras (e) Mice Protien Expression (f) USPS

(g) CTG (h) Segment (i) GCM (j) Fashion-MNIST

Figure 14: **Histogram of entropy for each of the ten datasets used to evaluate `ClAM`.**

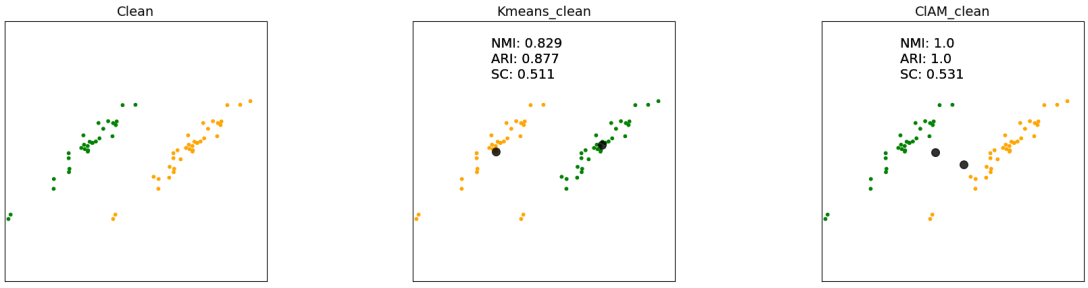
.

Table 10: **Noise handling in input data.**

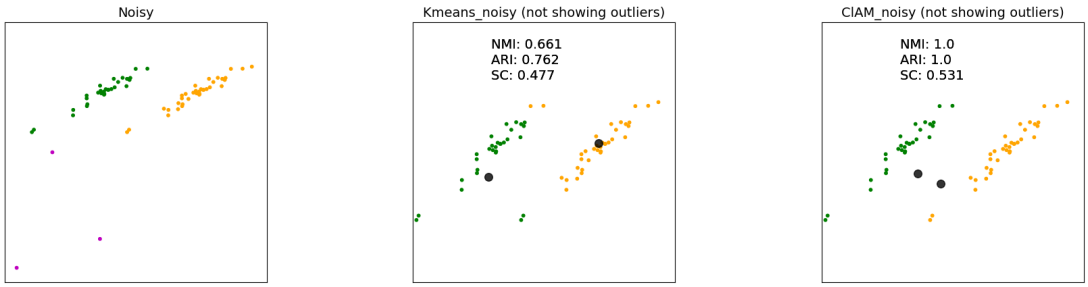
Method	SC (clean data)	SC (noisy data)
k-means	0.511	0.477
`ClAM`	0.531	0.531

clustering can also be sensitive to outliers. Given that, we expect **`ClAM`** to be potentially sensitive to outliers in the data. If we instead consider a more robust clustering objective in **`ClAM`** (for example, by replacing the squared Euclidean distance in equation (6) with a Manhattan distance), it would be more robust to outliers.

To evaluate the robustness to outliers, we consider a toy-example (Figure 15) with two clusters and we study the performance of k-means and **`ClAM`** in the presence of outliers. Figure 15a refers to the original dataset and Figure 15d refers to the dataset with outliers (which are shown in magenta color). Next, we run k-means and **`ClAM`** on both the original and noisy datasets (Figure 15b. Figure 15c refers to the results on the original dataset, and Figure 15e and Figure 15f refer to the results on the noisy dataset). We then compute the clustering quality via SC, NMI and ARI metrics. However, for the noisy dataset, we compute the metrics only on the inliers (i.e., excluding the outliers), to see how much the outliers affect the clustering quality of the actual inliers. Table 10 shows the performance of k-means and **`ClAM`** as noise (outliers) is added to the data. The results indicate that, as expected, the addition of outliers reduces the clustering quality of k-means. However, **`ClAM`** is able to cluster the data points correctly even with the outliers, with NMI and ARI both equal to 1.0. This indicates that **`ClAM`** turns out to be robust to the outliers in this example.



(a) Clean (b) k-Means clean (c) **ClAM** clean



(d) Noisy (e) k-Means noisy (f) **ClAM** noisy

Figure 15: **Noise handling in input data:** top row – clustering on original data, and bottom row – clustering with outliers.

C. Technical details

C.1. Proof of Proposition 3.1

Here we will provide a proof for Proposition 3.1.

First consider the following optimization problem:

$$\min_{\theta} \sum_{\boldsymbol{x} \in S} f(\boldsymbol{x}; \theta) \tag{14}$$

Lemma C.1 (Nesterov (2003); Fang et al. (2018))**.** *For a smooth function f (with respect to θ), SGD converges to a ϵ-stationary solution with $O|S|\epsilon^{-2})$ queries to the stochastic gradient oracle.*

Corollary C.2. *SGD converges to a ϵ-stationary solution with $O(\epsilon^{-2})$ epochs over the dataset S.*

We use these for proving Proposition 3.1:

Proof. For each point $\boldsymbol{x} \in B$ in a batch, the computation cost of T AM recursions, with each recursion taking $O(dk)$ time, takes a total of $O(dkT)$ time. Within one epoch, we perform the T AM recursions for each $\boldsymbol{x} \in S$, hence requiring $O(dkt|S|)$ time, where $|S|$ is the cardinality of S. Thus N epochs in Train**ClAM** takes $O(dkTN|S|)$ time.

If the objective in equation 9 is smooth, then Corollary C.2 tells us that the optimization will converge to a $O(N^{-1/2})$-stationary point. So what remains to be shown is that the objective in equation 9 is smooth, or more specifically, given $\boldsymbol{R} = \{\boldsymbol{\rho}_\mu, \mu \in [k]\}$ and $\boldsymbol{R}' = \{\boldsymbol{\rho}'_\mu, \mu \in [k]\}$, there exists a universal constant $C > 0$

$$\mathbb{E}_{\boldsymbol{m}\sim\mathcal{M}} \|\bar{\boldsymbol{m}} \odot (\boldsymbol{x} - \boldsymbol{x}^T_{\boldsymbol{R}})\|^2 - \mathbb{E}_{\boldsymbol{m}\sim\mathcal{M}} \|\bar{\boldsymbol{m}} \odot (\boldsymbol{x} - \boldsymbol{x}^T_{\boldsymbol{R}'})\|^2 \leq C \sum_{\mu\in[k]} \|\boldsymbol{\rho}_\mu - \boldsymbol{\rho}'_\mu\| \tag{15}$$

First, for binary masks $\boldsymbol{m} \sim \mathcal{M}$

$$\mathbb{E}_{\boldsymbol{m}\sim\mathcal{M}} \|\bar{\boldsymbol{m}} \odot (\boldsymbol{x} - \boldsymbol{x}^T_{\boldsymbol{R}})\|^2 - \mathbb{E}_{\boldsymbol{m}\sim\mathcal{M}} \|\bar{\boldsymbol{m}} \odot (\boldsymbol{x} - \boldsymbol{x}^T_{\boldsymbol{R}'})\|^2 \leq \|\boldsymbol{x} - \boldsymbol{x}^T_{\boldsymbol{R}}\|^2 - \|\boldsymbol{x} - \boldsymbol{x}^T_{\boldsymbol{R}'}\|^2 \tag{16}$$

For $\boldsymbol{x} \in S$ and bounded $\boldsymbol{R}, \boldsymbol{R}'$, triangle inequality gives us the following for some univeral constant C_1 such that

$$\|\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{R}}\|^2 - \|\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{R}'}\|^2 \leq C_1 \|\boldsymbol{x}^T_{\boldsymbol{R}} - \boldsymbol{x}^T_{\boldsymbol{R}'}\| \tag{17}$$

thus, we need to show that $\|\boldsymbol{x}_{\boldsymbol{R}}^T - \boldsymbol{x}_{\boldsymbol{R}'}^T\| \leq C_2 \sum_{\mu\in[k]} \|\boldsymbol{\rho}_\mu - \boldsymbol{\rho}'_\mu\|$ for some universal constant C_2.

Now, we have from the AM recursion and Cauchy-Schwartz inequality that:

$$\|\boldsymbol{x}_{\boldsymbol{R}}^T - \boldsymbol{x}_{\boldsymbol{R}'}^T\| \leq \sum_{t=1}^{T} \|\boldsymbol{\delta}_{\boldsymbol{R}}^t - \boldsymbol{\delta}_{\boldsymbol{R}'}^t\| \tag{18}$$

where $\boldsymbol{\delta}_{\boldsymbol{R}}^t \triangleq \boldsymbol{x}_{\boldsymbol{R}}^t - \boldsymbol{x}_{\boldsymbol{R}}^{t-1}$, and $\boldsymbol{\delta}_{\boldsymbol{R}'}^t \triangleq \boldsymbol{x}_{\boldsymbol{R}'}^t - \boldsymbol{x}_{\boldsymbol{R}'}^{t-1}$.

Considering the first AM update $\boldsymbol{\delta}_{\boldsymbol{R}}^1, \boldsymbol{\delta}_{\boldsymbol{R}'}^1$, we have

$$\begin{aligned}
\|\boldsymbol{\delta}_{\boldsymbol{R}}^1 - \boldsymbol{\delta}_{\boldsymbol{R}'}^1\| &\leq \sum_{\mu\in[k]} \left\|(\boldsymbol{\rho}_\mu - \boldsymbol{x})\exp(-\beta\|\boldsymbol{\rho}_\mu - \boldsymbol{x}\|^2) - (\boldsymbol{\rho}'_\mu - \boldsymbol{x})\exp(-\beta\|\boldsymbol{\rho}'_\mu - \boldsymbol{x}\|^2)\right\| &(19)\\
&\leq \sum_{\mu\in[k]} \left(\exp(-\beta\|\boldsymbol{\rho}_\mu - \boldsymbol{x}\|^2)\left\|\boldsymbol{\rho}_\mu - \boldsymbol{\rho}'_\mu\right\| + \|\boldsymbol{\rho}'_\mu - \boldsymbol{x}\|\left|\exp(-\beta\|\boldsymbol{\rho}_\mu - \boldsymbol{x}\|^2) - \exp(-\beta\|\boldsymbol{\rho}'_\mu - \boldsymbol{x}\|^2)\right|\right)\\
&\leq \sum_{\mu\in[k]} \left(\left\|\boldsymbol{\rho}_\mu - \boldsymbol{\rho}'_\mu\right\| + \|\boldsymbol{\rho}'_\mu - \boldsymbol{x}\|\left|\exp(-\beta\|\boldsymbol{\rho}_\mu - \boldsymbol{x}\|^2) - \exp(-\beta\|\boldsymbol{\rho}'_\mu - \boldsymbol{x}\|^2)\right|\right)\\
&\leq C_3 \sum_{\mu\in[k]} \left\|\boldsymbol{\rho}_\mu - \boldsymbol{\rho}'_\mu\right\| &(20)
\end{aligned}$$

for some universal C_3 for bounded $\boldsymbol{x}, \boldsymbol{\rho}_\mu, \boldsymbol{\rho}'_\mu$ since the $\exp(\cdot)$ function is also smooth.

For the $(t+1)^{\text{th}}$ AM update $\boldsymbol{\delta}_{\boldsymbol{R}}^{t+1}, \boldsymbol{\delta}_{\boldsymbol{R}'}^{t+1}$, we have

$$\begin{aligned}
\|\boldsymbol{\delta}_{\boldsymbol{R}}^{t+1} - \boldsymbol{\delta}_{\boldsymbol{R}'}^{t+1}\| &\leq \sum_{\mu\in[k]} \left\|(\boldsymbol{\rho}_\mu - \boldsymbol{x}_{\boldsymbol{R}}^t)\exp(-\beta\|\boldsymbol{\rho}_\mu - \boldsymbol{x}_{\boldsymbol{R}}^t\|^2) - (\boldsymbol{\rho}'_\mu - \boldsymbol{x}_{\boldsymbol{R}'}^t)\exp(-\beta\|\boldsymbol{\rho}'_\mu - \boldsymbol{x}_{\boldsymbol{R}'}^t\|^2)\right\| &(21)\\
&\leq \sum_{\mu\in[k]} \left\|\boldsymbol{\rho}_\mu - \boldsymbol{\rho}'_\mu\right\| + \left\|\boldsymbol{x}_{\boldsymbol{R}}^t - \boldsymbol{x}_{\boldsymbol{R}'}^t\right\| + \|\boldsymbol{\rho}'_\mu - \boldsymbol{x}_{\boldsymbol{R}'}^t\|\left|\exp(-\beta\|\boldsymbol{\rho}_\mu - \boldsymbol{x}_{\boldsymbol{R}'}^t\|^2) - \exp(-\beta\|\boldsymbol{\rho}'_\mu - \boldsymbol{x}_{\boldsymbol{R}'}^t\|^2)\right|\\
&\leq C_3 \sum_{\mu\in[k]} \left\|\boldsymbol{\rho}_\mu - \boldsymbol{\rho}'_\mu\right\| + k\left\|\boldsymbol{x}_{\boldsymbol{R}}^t - \boldsymbol{x}_{\boldsymbol{R}'}^t\right\| &(22)\\
&\leq C_3 \sum_{\mu\in[k]} \left\|\boldsymbol{\rho}_\mu - \boldsymbol{\rho}'_\mu\right\| + k\sum_{i=1}^{t}\left\|\boldsymbol{\delta}_{\boldsymbol{R}}^i - \boldsymbol{\delta}_{\boldsymbol{R}'}^i\right\| &(23)
\end{aligned}$$

in the same way as with $\boldsymbol{\delta}_{\boldsymbol{R}}^1, \boldsymbol{\delta}_{\boldsymbol{R}'}^1$ except with the additional recursive sum $\sum_{i=1}^t \left\|\boldsymbol{\delta}_{\boldsymbol{R}}^i - \boldsymbol{\delta}_{\boldsymbol{R}'}^i\right\|$. By induction, we can show that

$$\|\boldsymbol{\delta}_{\boldsymbol{R}}^{t+1} - \boldsymbol{\delta}_{\boldsymbol{R}'}^{t+1}\| \leq C_3(k+1)^t \sum_{\mu\in[k]} \left\|\boldsymbol{\rho}_\mu - \boldsymbol{\rho}'_\mu\right\| \tag{24}$$

which we can apply to the RHS of equation 18 to get the desired

$$\|\boldsymbol{x}_{\boldsymbol{R}}^T - \boldsymbol{x}_{\boldsymbol{R}'}^T\| \leq C_2 \sum_{\mu\in[k]} \left\|\boldsymbol{\rho}_\mu - \boldsymbol{\rho}'_\mu\right\| \tag{25}$$

for an universal constant C_2. This equation 25 combined with equation 17 gives us the desired condition in equation 15 for smoothness of the **`ClAM`** self-supervised loss function in equation 9. This completes the proof. □

C.2. Proof of Proposition 3.2

Proof. The proof of Proposition 3.2 follows for combining the computation cost of T AM recursions, with each recursion taking $O(dk)$ time, leading to a total of $O(dkT)$ time per point $\boldsymbol{x} \in S$. Then the total runtime of Infer**`ClAM`** is $O(dkT|S|)$, where $|S|$ is the cardinality of S. □