STREAMLINING EM INTO AUTO-ENCODER NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a new deep neural network architecture, named EDGaM, for deep clustering. This architecture can seamlessly learn deep auto-encoders and capture common group features of complex inputs in the encoded latent space. The key idea is to introduce a differentiable Gaussian mixture network between an encoder and a decoder. In particular, EDGaM streamlines the iterative Expectation-Maximum (EM) algorithm of the Gaussian mixture models into network design and replaces the alternative update with a forward-backward optimization. Being differentiable, both network weights and clustering centroids in EDGaM can be learned simultaneously in an end-to-end manner through standard stochastic gradient descent. To avoid preserving too many sample-specific details, we use both the clustering centroid and the original latent embedding for decoding. Meanwhile, we distill the soft clustering assignment for each sample via entropy minimization such that a clear cluster structure is exhibited. Our experiments show that our method outperforms state-of-the-art unsupervised clustering techniques in terms of both efficiency and clustering performance.

1 INTRODUCTION

Clustering is one of the most important techniques for analyzing data in an unsupervised manner. Well-established approaches for unsupervised clustering, including k-means (Lloyd, 1982) and Gaussian Mixture Models (GMM) (Reynolds, 2009), are the building blocks for numerous applications due to their efficiency and simplicity. However, their distance metrics are limited to the input space, making them ineffective for high-dimensional data such as images (Ji et al., 2019; Li et al., 2019), sequence data like time series (Zhang et al., 2018).

Recent advances in deep learning have paved the way for obtaining a good feature embedding of data, usually of low dimensionality (Bengio et al., 2007; Vincent et al., 2010). This line of research has also been extended in unsupervised learning, where deep clustering has become a popular area of study. Deep clustering refers to the process of clustering with deep neural networks, typically via introducing a cluster loss over features learned from the raw data by pre-trained CNNs (Yang et al., 2016; Caron et al., 2018) or auto-encoders (Song et al., 2013; Xie et al., 2016; Peng et al., 2016; Tian et al., 2017; Guo et al., 2019; Shiran & Weinshall, 2019). These algorithms have reported significant performance gains on various benchmark tasks over conventional non-deep clustering algorithms.

Various challenges arise when applying deep clustering. First, simply minimizing a clustering-based loss on the feature representation would lead to cluster collapse, where the representation may collapse to the trivial solution of a single cluster. Many methods avoid cluster collapse by adding extra tasks to clustering, such as self-reconstruction during pre-training or entire training (Guo et al., 2017a; Tian et al., 2017; Ghasedi Dizaji et al., 2017; Ji et al., 2017; Guo et al., 2017b). However, self-reconstruction tends to overestimate the importance of low-level features (e.g. color, textures, or background), which usually contain much more sample-specific details that are unrelated to semantics. Meanwhile, the objective of the clustering loss, which aims to maximize the similarity of latent representations for samples within each cluster, conflicts with that of the reconstruction task, which tries to preserve as much as sample-specific information in latent representations for reconstructing its input. Therefore, the clustering process is not stable, and alternative optimization is adopted for the tricky multi-objective problem. Meanwhile, the extra clustering task is not memory efficient for large-scale applications, since a large batch or even full batch is required for conducting clustering to ensure reliability (Yang et al., 2016; Ghasedi Dizaji et al., 2017).

In this paper, we design a new end-to-end framework, named EDGaM, for deep clustering, which can capture the group structure of complex inputs in the encoded latent space. In particular, EDGaM introduces a differentiable Gaussian mixture network between the encoder and the decoder, which streamlines the iterative Expectation-Maximum (EM) algorithm of the Gaussian mixture models into network design. Being a part of auto-encoder, the clustering objective implied by EDGaM is consistent with the reconstruction loss, namely, capturing as much as common information for better reconstruction. Since the whole objective is dominated by the reconstruction task, the latent embedding inevitably reserves sample-specific details, which hinders EDGaM from learning a clear clustering structure. To avoid preserving too many sample-specific details, we introduce a skip connection to combine the original latent embedding and the clustering centroid for decoding. It can relieve EDGaM from the final reconstruction and focus more on the clustering task. Meanwhile, we propose to minimize the entropy over the soft cluster assignment for each sample to deduce a clear cluster structure. In summary, the main contributions of this paper are as follow:

- We introduce EDGaM for clustering on large-scale and high dimensional datasets. Instead of adopting the alternative update as most clustering methods, all parameters in EDGaM can be updated simultaneously and end-to-end through stochastic gradient descent.
- EDGaM reconciles the conflicts between clustering and self-reconstruction. Therefore, EDGaM enjoys a stable clustering process, namely, capturing as much as common information for better reconstruction. Meanwhile, EDGaM can be easily extended to the cluster-imbalanced scenario with no extra computation cost.
- We conduct extensive experiments on four benchmark datasets. Our EDGaM outperforms state-of-the-art deep clustering techniques in terms of both efficiency and clustering performance, due to the end-to-end training procedure.

2 Related Work

Deep Unfolding Deep unfolding refers to the strategy that reformulating an iterative algorithm into a neural network (Hershey et al., 2014), which can be then optimized via gradient descent instead of iteration. Apart from non-negative matrix factorization and belief propagation that introduced in the original paper, researchers have found its extension to topic model (Chien & Lee, 2017) and ADMM (Yang et al., 2018). However, these deep unfolding extensions are not compatible well with popular deep learning platforms, since they can only be optimized with explicitly full batch gradients. Recent literature, such as N-EM (Greff et al., 2017) and IODINE (Greff et al., 2019), proposed to unfold the EM algorithm of mixture distributions and the resultant network can be optimized end-to-end by SGD. Both N-EM and IODINE are designed for segmentation, i.e., pixel-level clustering over the raw features. They are not suitable for the (sample-level) deep clustering problem we consider in this paper, since semantically similar samples are not necessarily similar in the raw feature space the sample resides in.

Deep clustering The dominant and most successful approach to the clustering in recent years has been to incorporate the tasks of representation learning and clustering into the same framework. They can be further divided into three types. In the first category, a cluster loss is introduced over latent representations learned from the raw data by pre-trained CNNs or auto-encoders (Bengio et al., 2007; Vincent et al., 2010; Masci et al., 2011; Kingma & Welling, 2013; Zhao et al., 2015), such as DEC (Xie et al., 2016) and Deepcluster (Caron et al., 2018). However, simply minimizing a clustering-based loss on the feature representation would lead to cluster collapse. The second type of clustering method proposes to joint clustering training with extra tasks, e.g., self-reconstruction. Prominent works in the past years have been JULE (Yang et al., 2016), IDEC (Guo et al., 2017a), VaDE (Jiang et al., 2017), DCN (Yang et al., 2017) and DEPICT (Ghasedi Dizaji et al., 2017). The same issue also applies to the mixture-of-expert based deep clustering (Zhang et al., 2017; Chazan et al., 2019). The reconstruction loss of auto-encoder tends to overestimate the importance of low-level features, which usually contains much more sample-specific details that are unrelated to semantics. Apart from self-reconstruction, the third type learns semantic clustering joint with self-supervised tasks while using the information as criteria, such as IMSAT (Hu et al., 2017), ADC (Haeusser et al., 2018) and IIC (Ji et al., 2019). Semantics pairs are constructed beforehand for each sample through a collection of predefined semantics-invariant transformations. However, the performance of these deep cluster

models highly depends on the choice of invariant transformations, which requires domain knowledge and usually varies from dataset to dataset (Jing & Tian, 2019; Tian et al., 2020).

Our work falls into the second category. We unify the goal of clustering and self-reconstruction with two contributions. First, we streamline the iterative EM algorithm of GMM and introduce it between the encoder and the decoder. Second, the common group feature is combined with the original latent embedding for decoding.

3 EDGAM: ENCODER WITH DIFFERENTIABLE GAUSSIAN MIXTURE NETWORK

In this section, we first streamline the iterative EM algorithm of GMM into network design, which enables end-to-end stochastic gradient optimization. Then we Encode the obtained Differentiable Gaussian Mixture network (EDGaM) within the auto-encoder framework so as to cluster over the latent representations instead of raw features.

Problem statement: Let $Z = \{z_n\}_{n=1}^N$ denote raw features or latent embedding for a collection of samples, drawn from heterogeneous populations. Assume the number of clustering size is known to be $K(\ll N)$. The clustering task aims to partition the N samples into K nonoverlap groups, where the samples in the same group are more similar to each other than to those in other groups. Particularly, we target for the large-scale scenario with high-dimensional raw features.

3.1 GAUSSIAN MIXTURE MODELS AND ITS EM SOLUTION

GMM is a classical clustering model, which models the distribution of each sample z_n as a linear superposition of Gaussians. Let $\theta = \{\mu_k, \Sigma_k, \pi_k\}_{k=1}^K$ denote the clustering parameters. Formally, the objective of GMM is formulated as follows

$$\mathcal{L}(\theta|Z) = -\sum_{n=1}^{N} \ln \big[\sum_{k=1}^{K} \pi_k \mathcal{N}(z_n|\mu_k, \Sigma_k)\big], \quad \text{where} \sum_k \pi_k = 1. \tag{1}$$

Direct minimization of equation 1 is quite difficult numerically, because of the sum of terms inside. The EM algorithm provides a simpler solution. Assuming the latent variable $\delta_{nk} \in \{0, 1\}$, let $\delta_{nk} = 1$ denote z_n is sampled from the k-th cluster, otherwise z_n comes from other clusters. Namely, $p(\delta_{nk} = 1) = \pi_k$ and $p(z_n|\delta_{nk} = 1, \theta) = \mathcal{N}(z_n|\mu_k, \Sigma_k)$. Note $\sum_{k=1}^K \delta_{nk} = 1$ since each sample can only be sampled from one cluster. Given an initial set of parameters, i.e., $\theta^1 = \{\mu_k^1, \Sigma_k^1, \pi_k^1\}_{k=1}^K$, The EM algorithm for GMM proceeds by alternating between two steps ($t = 1, 2, \ldots, T$):

- **Expectation step (E-step):** calculate the posterior expectation of the latent variable $\{\delta_{nk}\}_{k=1,n=1}^{K,N}$:

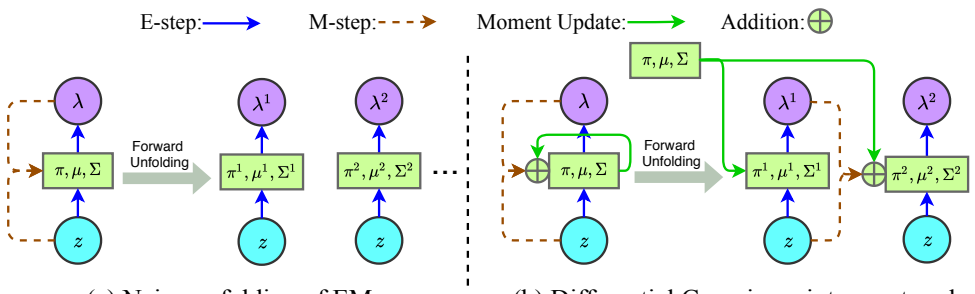
$$E_{p(\delta_{nk}|z_n,\theta)}[\delta_{nk}] = \frac{p(z_n, \delta_{nk} = 1|\theta)}{\sum_i p(z_n, \delta_{ni} = 1|\theta)} = \frac{\pi_k^t \mathcal{N}\left(z_n|\mu_k^t, \Sigma_k^t\right)}{\sum_{i=1}^K \pi_i^t \mathcal{N}\left(z_n|\mu_i^t, \Sigma_i^t\right)} \triangleq \lambda_{nk}^t.$$

- **Maximum step (M-step):** Update the model parameters $\theta = \{\mu_k, \Sigma_k, \pi_k\}_{k=1}^K$:

$$N_k = \sum_n \lambda_{nk},\ \pi_k^{t+1} = \frac{N_k}{N},\ \mu_k^{t+1} = \frac{1}{N_k} \sum_{n=1}^{N} \lambda_{nk}^t z_n,\ \Sigma_k^{t+1} = \frac{1}{N_k} \sum_{n=1}^{N} \lambda_{nk}^t (z_n - \mu_k^t)(z_n - \mu_k^t)^{\mathrm{T}},$$

After EM converges, sample z_n is assigned to its nearest cluster with the largest λ_{nk}. Namely, $S_k = \left\{z_n : \lambda_{nk} \geq \lambda_{ni}, n = 1, 2, \ldots, N, k = 1, 2, \ldots, K\right\}$.

However, the standard EM requires a full batch update at M-step, which is not suitable for large-scale (large N) applications. Further, the alternative update paradigm in EM would restrict its efficiency when joint learning GMM with other learning tasks, such as feature learning in deep clustering. Some previous work (Titterington, 1984; Chazan et al., 2018) proposed to enable the EM algorithm differentiable such that it can be further optimized using stochastic gradient descent. They fall into the same scope with the principle of deep unfolding (Hershey et al., 2014). Inspired by this, we consider unfolding the EM algorithm to avoid iteration in the following.



(a) Naive unfolding of EM (b) Differential Gaussian mixture network

Figure 1: Forward unfolding of various EM algorithms for Gaussian mixture models.

3.2 Streamlining the iterative EM algorithm into network design

The deep unfolding principle suggests reformulating an iterative algorithm into a neural network. In particular, the forward pass equals to the inference process, aiming to estimate a latent variable based on the current parameters; the backward propagation conducts the optimization, adjusting the model parameters based on the loss in pursuit of better representation.

3.2.1 Naively streamlining EM as a forward-backward optimization

Accordingly, a T-layer neural network can be naively constructed (See Fig. 1(a)), where $\theta^t = \{\mu_k^t, \Sigma_k^t, \pi_k^t\}_{k=1}^K$ and $\boldsymbol{\lambda}_n^t = \{\lambda_{nk}^t\}_{k=1}^K$ denote the network weights and the output of the t-th layer, respectively. In particular, each forward pass is exactly the same as the E-step:

$$\boldsymbol{\lambda}_n^t = \mathcal{F}(z_n, \theta^t, \boldsymbol{\lambda}_n^{t-1}) \overset{i}{\equiv} \mathcal{F}(z_n, \theta^t) \quad t = 1, 2, \ldots, T. \tag{2}$$

We involve $\boldsymbol{\lambda}_n^0$ as a placeholder for a better explanation. i denotes that each forward pass (equation 2) is independent of its input $\boldsymbol{\lambda}_n^{t-1}$ (i.e., the output of the previous layer), given the input z_n and model parameters θ^t. It means that simply streamlining the EM algorithm into the network design would lead to a network that is not successively connected (See Fig. 1(a)). Therefore, it is not a valid neural network since the gradient cannot backpropagate along the layers.

Remark 1 (Difference between the EM algorithm and deep neural network)**.** *Let us consider each E-step as one forward pass while each M-step as one backward propagation. The EM algorithm can be viewed as multiple alternative updates between the forward pass and the backward propagation until converge. Meanwhile, the deep neural network is constituted of multiple successive forward passes, followed by multiple successive backward propagation. The difference lies in that each forward pass of EM is independent of the output of the previous iteration, given the data and model parameters.*

3.2.2 Maximum step as a correction in loss

Inspired by Liang & Klein (2009), we introduce the moment update to bridge the connection between the multiple forward passes, using the the M-step omitted by the naive unfolding process. Let $\boldsymbol{\mu} = \{\mu_k\}_{k=1}^K$ denote the clustering centroids. For simplicity, we fix Σ_k to the identity matrix I and π_k to $\frac{1}{K}$ $\forall k = 1, 2, \ldots, K$. The new forward pass can be formulated as

$$\boldsymbol{\mu}^1 = \boldsymbol{\mu}, \quad \lambda_{nk}^t = \mathcal{F}(z_n, \boldsymbol{\mu}^t, \boldsymbol{\lambda}_n^{t-1}) = \frac{\exp\left(-\frac{1}{2}\|z_n - \mu_k^t\|_2^2\right)}{\sum_{i=1}^K \exp\left(-\frac{1}{2}\|z_n - \mu_i^t\|_2^2\right)}, \quad \boldsymbol{\mu}^{t+1} = \mathcal{M}(Z, \boldsymbol{\mu}, \boldsymbol{\lambda}_n^t), \tag{3}$$

where $k = 1, 2, \ldots, K,\ t = 1, 2, \ldots, T$. And T denotes the number of unfolded network layers. In particular, the moment update formulation $\boldsymbol{\mu}^{t+1} = \mathcal{M}(Z, \boldsymbol{\mu}, \boldsymbol{\lambda}_n^t)$ consists of the following two steps:

$$\tilde{\mu}_k^{t+1} = \frac{1}{\sum_i \lambda_{ik}^t} \sum_n \lambda_{nk}^t z_n, \ \forall k = 1, 2, \ldots, K, \quad \boldsymbol{\mu}^{t+1} = (1 - \alpha_{t+1})\boldsymbol{\mu} + \alpha_{t+1}\tilde{\boldsymbol{\mu}}^{t+1}. \tag{4}$$

$\alpha_{t+1} \in [0, 1]$ is a learnable hyperparameter to balance the two component.

Remark 2 (The simple setting for Σ and π is sufficient to capture the clustering structure in the latent space)**.** *(1) The covariance matrix Σ is useful for improving clustering when clusters are overlapped. When encountering a well-separated latent embedding (See Fig. 5(a)), the centroids*

alone are sufficient to capture the clustering structure. (2), The cluster weight π, denoting the size of each cluster, was designed to give higher preference to the large cluster when a sample is close to multiple cluster centroids simultaneously. Similarly, it is only useful for disambiguation when clusters are overlapped but is redundant for our well-separated latent embedding.

The moment update equation 4 links the estimation of latent variable $\boldsymbol{\lambda}_n^t$ to its previous state via $\boldsymbol{\mu}^t$, which ensures effective successive forward passes. However, it does not impose the corresponding update information on the clustering centroid $\boldsymbol{\mu}$. For the sake of analysis, we reorganize the moment update as the equivalent gradient descent formulation:

$$\boldsymbol{\mu}^{t+1} = \boldsymbol{\mu} - \alpha_{t+1}(\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}}^{t+1}) = \boldsymbol{\mu} - \alpha_{t+1} g(\boldsymbol{\mu}), \quad g(\boldsymbol{\mu}) = \boldsymbol{\mu} - \tilde{\boldsymbol{\mu}}^{t+1}, \tag{5}$$

where $g(\boldsymbol{\mu})$ is the gradient w.r.t. $\boldsymbol{\mu}$. Equation 5 reveals that the updating information can be imposed on the clustering centroids $\boldsymbol{\mu}$ via the equivalent gradient. Considering the gradient is usually derived from loss function, we integrate the gradient $g(\boldsymbol{\mu})$ over $\boldsymbol{\mu}$ and derive the corresponding loss function.

$$\mathcal{L}_{correction}(\boldsymbol{\mu}) = \frac{1}{2}\sum_{t=1}^{T-1} \alpha_{t+1} \|\boldsymbol{\mu} - \tilde{\boldsymbol{\mu}}^{t+1}\|_2^2 = \sum_{t=1}^{T-1} \|(\beta_{t+1}\boldsymbol{\mu} + (1-\beta_{t+1})\tilde{\boldsymbol{\mu}}^{t+1}) - \tilde{\boldsymbol{\mu}}^{t+1}\|_2^2. \tag{6}$$

where β_{t+1} is a new learnable hyperparameter which satisfies $\beta_{t+1} = \sqrt{\frac{\alpha_{t+1}}{2}}$.

3.2.3 Differential Gaussian mixture network

According to our analysis, we can derive a differentiable Gaussian mixture multi-layer network for clustering (See Fig. 1(b)). The multiple successive forward passes are completed by our new forward propagation formulation (i.e.,equation 3). Afterward, we calculate the loss at the last layer, which is derived from the clustering objective equation 1:

$$\mathcal{L}_{cluster}(\boldsymbol{\mu}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} \lambda_{nk} \ln[\frac{1}{K}\mathcal{N}(z_n|\mu_k, I)] \overset{i}{\approx} \frac{1}{2}\sum_{n=1}^{N} \boxed{\frac{N}{KN_k}} \sum_{k=1}^{K} \lambda_{nk}\|z_n - \mu_k\|_2^2, \tag{7}$$

where $N_k = \sum_{n=1}^{N} \lambda_{nk}^T$. i holds because (1) we omit the constant $N \ln K + \frac{Nd}{2}\ln(2\pi)$; (2) we reweigh the within-group variance with $\frac{N}{KN_k}$ to lend an ear to small clusters when clusters are imbalanced. Let $w_k = \frac{N}{KN_k} = \frac{\bar{N}}{N_k}$, where $\bar{N}$ is the average number of samples for each cluster and N_k is the number of samples belonging to cluster k. Regarding a cluster balance dataset, i.e., $w_k \approx 1$, the reweigh strategy becomes invalid and equation 7 reduces to regular cluster loss. Otherwise, equation 7 will adjust the penalty of within-group variance to lend an ear to small clusters when clusters are imbalanced.

In terms of the mini-batch update, the data statistics N, K, N_k for the whole dataset is not available. We replace it with the data statistics N', K', N_k' collected on each mini-batch data. To avoid sampling biased, we suggest adding a smoothing variable, i.e., $N_k' = \sum_n \lambda_{nk}^T + \Delta$, $N' = \sum_k N_k' + K'\Delta$, where Δ is set to 3 in the experiment.

Overall, the loss for the differential Gaussian mixture network (See Fig. 1(b)) consists of two components: the clustering loss (equation 7) and the correction loss (equation 6). Since the correction loss deduced from the backward propagation is already added in the loss, the backward propagation can be completed automatically following the stochastic gradient descent. The cluster performance is therefore iteratively improved by updating the network parameter, i.e., the centroids $\boldsymbol{\mu}$.

Remark 3 (EM V.S. EM-unfolded network)**.** *The number of EM iterations affects differently from the number of EM-unfolded layers due to different optimization paradigms. Since EM is optimized per iteration, more iterations would contribute to better performance. Whereas multiple layers in EDGaM are optimized simultaneously via gradient descent, more layers would summarize the embedding excessively and lead to gradient vanish. However, this should not be a big issue since we empirically verified that* 3*-layers can consistently achieve good performance on all datasets.*

3.3 Encoded the differential Gaussian mixture network within auto-encoder

However, the differential Gaussian mixture network still falls in the scope of linear clustering, which limits its potential for a large proportion of real applications with complex input features. In this

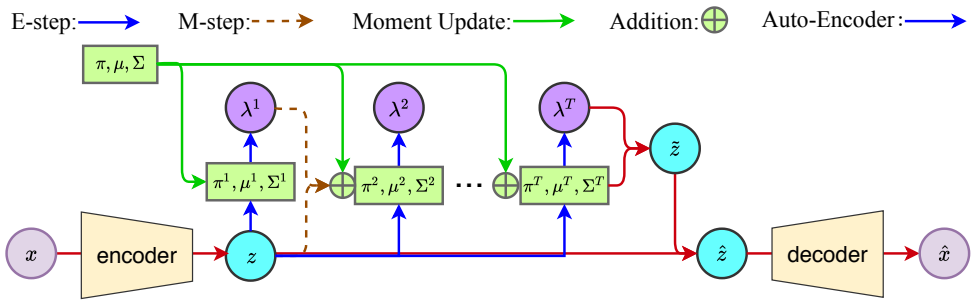


Figure 2: Network structure of the EDGaM model.

section, we consider encoding Fig. 1(b) into the auto-encoder framework, where the differential Gaussian mixture network is deployed over the latent representations instead of raw features.

3.3.1 Auto-Encoder for nonlinear feature extraction

The last layer output of the differential Gaussian mixture network is the soft assignment $\boldsymbol{\lambda}_n^T$, which is insufficient for the reconstruction task. Thus, we consider adding one more layer to reconstruct its input. Meanwhile, since the AE objective is dominated by the reconstruction task, the latent feature representation inevitably reserves sample-specific details, which hinders EDGaM from learning a good clustering structure. To avoid such issues, we use both the latent embedding reconstruction and the original latent embedding for decoding.

$$\tilde{z}_n = \sum_{k=1}^{K} \lambda_{nk}^T \mu_k^T, \qquad \hat{z}_n = f(z_n, \tilde{z}_n). \tag{8}$$

The skip connection in equation 8 can relieve EDGaM from the final reconstruction and focus on capturing the common group features. In our experiment, we implement $f(z_n, \tilde{z}_n)$ by simple concatenation along with one extra fully connection layer.

3.3.2 The whole structure of our EDGaM

Note the $\tilde{z}_n$ in equation 8 degenerates into the cluster centroid μ, when the group assignment λ is close to a one-hot vector. Thus, we propose to minimize the entropy of the soft assignment λ for each sample to distill a clear cluster structure. The overall loss of EDGaM (See Fig. 2) is summarized as:

$$\mathcal{L}(\boldsymbol{\mu}|X) = \sum_{n=1}^{N} \|x_n - \tilde{x}_n\|_2^2 + \eta_1 \mathcal{L}_{cluster}(\boldsymbol{\mu}) + \eta_2 \mathcal{L}_{correction}(\boldsymbol{\mu}) + \eta_3 \sum_{n=1}^{N}\sum_{k=1}^{K} \lambda_{nk} \log \lambda_{nk}. \tag{9}$$

where η_1, η_2, η_3 are the trade-off parameters. There are four types of loss calculated. The first one is the reconstruction loss, which ensures the nonlinear feature learning and avoids cluster collapse. The second term is the clustering loss (See equation 7), which aims to squeeze out the specialty of each sample while maximally recover their latent embedding using the captured common group features. The third one the correction loss (See equation 6), which imposes the same gradient information as the maximum steps. The fourth one is the entropy loss, which helps to distill a clear cluster structure.

Remark 4. *We streamline the EM algorithm of GMM and encode it into the AE framework. Our EDGaM is naturally suitable for large-scale and high-dimensional applications, since all parameters in EDGaM can be updated in parallel and end-to-end through SGD. Meanwhile, the common group feature is combined with the original latent embedding for decoding. Through this skip connection, we unify the goal of clustering and self-reconstruction so as to derive a better cluster performance.*

4 Time Complexity Analysis

This section analyzes the time complexity of our EDGaM and two popular AE-based deep clustering methods, JULE (Yang et al., 2016) and DEPICT (Ghasedi Dizaji et al., 2017).

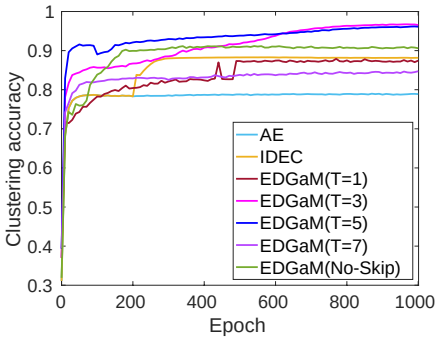


Figure 3: Ablation study for EDGaM.

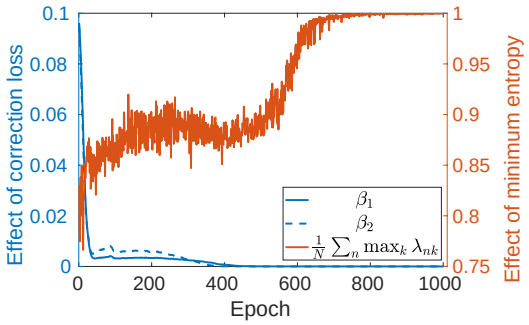


Figure 4: Effects of various losses on MNIST

Let M, B and K denote the number of iterations, the batch size, and the number of clusters, respectively. Following the same AE architecture with the identical latent feature size d, we only analyze the extra time cost incurred by the clustering module. (1) The extra complexity introduced by EDGaM is $\mathcal{O}(MBKdT_1)$, where T_1 denotes the number of EM-unfolded layers. (2) Note, DEPICT needs to run k-means to recompute the target assignment every v epochs on the full batch samples. The extra cost introduced by DEPICT is $\mathcal{O}(\frac{M}{v}NKdT_2)$, where T_2 is the number of k-means iterations. (3) JULE adopts the hierarchical agglomerative clustering during the representation learning. The extra time complexity incurred by the agglomerative clustering is $\mathcal{O}(N^3d)$ because it needs to exhaustively scan the $N \times N$ distance matrix for the lowest distance in each of N-K iterations.

To sum up, our EDGaM shows its superior computational efficiency for the large-scale (N) and high-dimensional dataset. JULE is not practical for large-scale applications due to its $\mathcal{O}(N^3d)$ time complexity. Meanwhile, compared to our EDGaM using mini-batch (B) samples, DEPICT is not memory efficient, since it needs to run the k-means algorithm with full batch ($N \gg B$) samples.

5 Experiment

Dataset: We evaluated our EDGaM on four image datasets, i.e., MNIST (LeCun et al., 1998), USPS (Hastie et al., 2009), YTF (Aggarwal & Zhai, 2012), Fashion (Xiao et al., 2017) and one text dataset, i.e., Reuters10K (Xie et al., 2016). The statistics of datasets are introduced in Table 1.

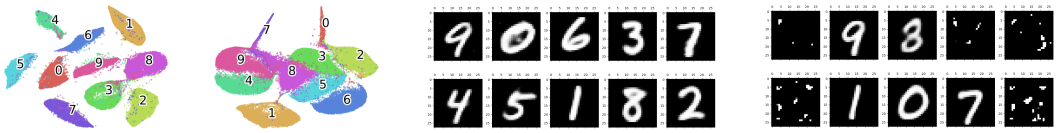
Baselines: We compare our EDGaM against auto-encoder based clustering methods. For all the baselines, results were retrieved from the literature or computed by us when not found and possible to recompute. Algorithms with the missing scores on some datasets are because the original paper did not test on this dataset and it was not easily possible to get a satisfying score.

Table 1: The statistics of datasets

Dataset	#sample	#cluster	#dim
MNIST	$70,000$	10	$1 \times 28 \times 28$
USPS	$9,298$	10	$1 \times 16 \times 16$
YTF	$10,000$	40	$3 \times 55 \times 55$
Fashion	$70,000$	10	$1 \times 28 \times 28$
Reuters10K	$10,000$	4	$1 \times 2,000$

Metric: For all methods, we set the number of clusters to the ground truth categories and evaluate performance with unsupervised clustering accuracy: Accuracy (ACC), Normalized mutual information (NMI). The best mapping can be efficiently computed by the Hungarian algorithm (Kuhn, 1955). For both two metrics, values range between 0 and 1, where higher value indicates better performance.

Experiment Settings: We implement EDGaM with PyTorch (Paszke et al., 2017). In terms of MNIST,USPS and Reuters10K, EDGaM is built upon the AE architecture described in (Xie et al., 2016). The encoder is a fully connected multi-layer perceptron with dimensions d-500-500-2000-c. d is the dimension of input and c is the dimension of centroids. The decoder network is a mirror of the encoder. All layers use ReLU activation (Nair & Hinton, 2010). In terms of YTF and Fashion, we adopt a convolution structure (See Appendix for more details). For all datasets, the dimension of centroids is fixed to 10. The optimizer is Adam (Kingma & Ba, 2014). The learning rate is 0.001 and the training epoch is $1,000$. The mini-batch size is set to 128, excepted for YTF whose is set to 256. We streamline EM into a three-layer network for all datasets, i.e., $T = 3$.



(a) EDGaM Embed (b) IDEC Embed (c) EDGaM Centroids (d) IDEC Centroids

Figure 5: (a-b) t-SNE for MNIST embedding. (c-d) The reconstruction of MNIST centroids.

5.1 Efficacy of different components on EDGaM using the MNIST dataset

In Fig. 3, we compare different variants of EDGaM with most related baselines, i.e., AE and IDEC, to analyze the efficacy of different components in EDGaM. T denotes the number of EM-unfolded layers (Fig. 1(b)). "No-Skip" denotes only the reconstruction outputted by EM is used for decoding (equation 8). We showcase the MNIST dataset.

Fig. 3 shows that: (1) All EDGaM variants outperform AE, which means that streamlining EM into AE would not undermine the latent embedding. (2) The best T should between 3-5, since a shallow network ($T = 1$) equals to impose the cluster loss directly on the embedding space, similar to IDEC; while a deep network ($T \geq 7$) would summarize the embedding so many times that the cluster loss cannot backpropagate to the embedding space. (3) Decoding only with the reconstruction outputted by EM can also achieve good cluster results (> 0.9). It demonstrates that streamlining EM for GMM as a neural network can capture the group structure in the dataset.

In Fig. 4, we visualize coefficients β_1, β_2 (both initialized to be 0.1 in Eq. 6), and the average of the maximum group assignment $\frac{1}{N}\sum_n \max_k \lambda_{nk}$. It shows that: (1) both β_1, β_2 decrease to a small value (< 0.05) after the first 50 iterations, which helps EDGaM to gradually adjust the model parameters μ according to the mini-batch samples. (2) $\frac{1}{N}\sum_n \max_k \lambda_{nk}$ quickly increases to above 0.85 and remains stable, when self-reconstruction and the clustering loss dominant the learning process. Then, it continually increases to one, when the cross-entropy loss dominates the learning process and aims to create a well-separate latent embedding. (3) After 800 iterations, both the correction loss and cross-entropy loss decreases to near zero, since $\beta_1, \beta_2 \to 0$ and $\frac{1}{N}\sum_n \max_k \lambda_{nk} \to 1$.

5.2 Time Efficiency

To evaluate the efficiency of our EDGaM in dealing with large-scale (7×10^4) and high dimensional (784) data, we collect the time cost of EDGaM with its competing algorithms JULE and DEPICT in Table 2. Both JULE and its fast version JULE(fast) are evaluated.

Table 2: Time Cost (s) of 10^3 iteration on MNIST

Baseline	JULE	JULE(fast)	DEPICT	EDGaM
Time(s)	1.44×10^5	3.00×10^4	1.09×10^4	$\mathbf{9.28 \times 10^3}$

Table 2 shows that: (1) the time cost of various approaches is consistent with our analysis in Sect. 4. (2) Our EDGaM achieves superior efficiency while both JULE and its fast version need to spend significantly higher times. (3) In terms of imbalanced datasets, DEPICT suggests adopting the highly energy-consuming agglomerative clustering instead of k-means to ensure good performance. Therefore, DEPICT will suffer the same issues as JULE for large-data imbalanced datasets.

5.3 Clustering visualization of EDGaM on the MNIST dataset

We visualize the latent embedding (Fig. 5(a-b)) and the corresponding ten centroids (Fig. 5(c-d)) of the MNIST dataset using EDGaM and IDEC, respectively. We do not compare with DEPCIT (Ghasedi Dizaji et al., 2017), since it adopts the same objective as IDEC but varies a lot on different datasets. Meanwhile, the JULE is not a partition-based clustering method, which does not have centroids. Fig. 5(a) shows that there exists a clear gap between different clusters. This is the reason that EDGaM can achieve good clustering results. Meanwhile, it is interesting that the reconstruction of the centroids is exactly the MNIST digits (Fig. 5(c)), which demonstrates our EDGaM indeed captures common group information and performs semantic clustering in the latent space, while IDEC does not. Further, as reported in the original paper (App. Fig. 4c), its embedding of different classes are overlapping.

5.4 COMPARISONS ON THE BENCHMARK DATASETS

In Table 3, we compare EDGaM with the popular shallow clustering methods, the first type deep clustering method, and the latest auto-encoder based second type deep clustering methods.

Table 3: Comparisons of our method with popular clustering algorithms.

Method	MNIST		USPS		YTF		Fashion		Average Rank	
	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI
k-means (Lloyd, 1982)	0.532	0.500	0.668	0.627	0.601	0.776	0.474	0.512	10.33	10
DeepCluster (Caron et al., 2018)	0.797	0.661	0.562	0.540	–	–	0.542	0.510	7.67	9
GMM (Reynolds, 2009)	0.433	0.366	0.551	0.530	–	–	0.556	0.557	8.67	8.67
AE+k-means (Song et al., 2013)	0.760	0.669	0.715	0.651	–	–	–	–	7.5	7.5
DEC (Xie et al., 2016)	0.863	0.834	0.762	0.767	0.371	0.446	0.518	0.546	5.5	5.75
DCN (Yang et al., 2017)	0.830	0.810	0.688	0.683	–	–	0.501	0.558	7.33	5.67
VaDE (Jiang et al., 2017)	0.945	0.876	0.566	0.512	–	–	0.578	0.630	5	5.67
IDEC (Guo et al., 2017a)	0.881	0.867	0.761	0.785	–	–	0.529	0.557	5.33	4.67
DEPICT (Ghasedi Dizaji et al., 2017)	0.965	0.917	**0.964**	**0.927**	0.621	0.802	0.392	0.392	4	4
JULE (Yang et al., 2016)	0.964	0.913	0.950	0.913	**0.684**	**0.848**	0.563	0.608	2.5	2.25
EDGaM	**0.967**	**0.922**	0.953	0.894	0.679	0.825	**0.644**	**0.653**	**1.5**	**1.75**

Table 3 shows that: (1) EDGaM achieves the state of the art clustering results on the MNIST and Fashion datasets. (2) EDGaM is inferior to JULE but still outperforms rest baselines on the YTF dataset. This is because YTF is an extremely imbalanced dataset. Therefore, the partition-based EDGaM may encounter the cluster collapse, while the hierarchical-based JULE can escape this by adopting the bottom-up clustering strategy. However, JULE needs to traverse the whole dataset to merge two clusters, which suffers from high computation cost (See Sect. 4 and Table 2).

The third type deep clustering method, e.g., IMSAT (Hu et al., 2017) and IIC (Ji et al., 2019), can achieve state-of-the-art clustering results on various (image) datasets. However, their performance highly depends on the choice of invariant transformations, which requires domain knowledge and usually varies from dataset to dataset. To justify our claim, we compared some recent self-supervised based deep clustering methods with AE-based deep clustering in Table 4. For the best comparison, we present the best-reported results from their original papers or from Guo et al. (2017a).

Table 4: Comparisons to third type of deep clustering methods.

ACC	Self-Supervised		Graph	Auto-Encoder			
	IMSAT	IIC	Kindar	AE+k-means	IDEC	VaDE	EDGaM
MNIST	0.984	**0.992**	0.985	0.818	0.881	0.945	0.967
Reuters10K	0.710	–	0.705	0.705	0.756	0.794	**0.807**

Table 4 shows that: (1) self-supervised based deep clustering approaches highly depend on the choice of invariant transformations. If no effective invariant transformations are available, like the text data, their clustering performance would degenerate as AE+k-means. (2) Kindar (Gupta et al., 2019) is highly dependent on the original latent embedding output by AE. If a good latent embedding is not available, its clustering performance also degenerates as AE+k-means. (3) Joint learning of deep embedding and clustering can mutually promote each other, especially when the AE embedding is not good enough. (4) Graph-based Kindar need to respectively build multiple graphs with the whole dataset, which is not suitable for the large-scale application that we target.

6 CONCLUSION

In this paper, we present a new neural network architecture, named EDGaM, for deep clustering. EDGaM streamlines the iterative EM algorithm of GMM into auto-encoder networks, and can capture the cluster structure of complex inputs in the encoded latent space. Since GMM induces the cluster structure via minimizing the variance of all samples within each cluster, only when the semantic labels exhibit the most variance direction, can the learned cluster structure be consistent with the semantics labels. Otherwise, the cluster results would not be interpreted. Many methods can be adapted to enlarge the ratio of the variance that is consistent with semantic labels, such as constrained clustering and self-supervision tasks. These methods are complementary to our work, which can be incorporated into our EDGAM to deduce a semantic consistent cluster structure.

REFERENCES

Charu C Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. In *Mining text data*, pp. 77–128. Springer, 2012.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pp. 153–160, 2007.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 132–149, 2018.

Shlomo E Chazan, Sharon Gannot, and Jacob Goldberger. Training strategies for deep latent models and applications to speech presence probability estimation. In *International Conference on Latent Variable Analysis and Signal Separation*, pp. 319–328. Springer, 2018.

Shlomo E Chazan, Sharon Gannot, and Jacob Goldberger. 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.

Jen-Tzung Chien and Chao-Hsi Lee. Deep unfolding for topic models. *IEEE transactions on pattern analysis and machine intelligence*, 40(2):318–331, 2017.

Kamran Ghasedi Dizaji, Amirhossein Herandi, Cheng Deng, Weidong Cai, and Heng Huang. Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *Proceedings of the IEEE international conference on computer vision*, pp. 5736–5745, 2017.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Klaus Greff, Sjoerd Van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization. In *Advances in Neural Information Processing Systems*, pp. 6691–6701, 2017.

Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, pp. 2424–2433, 2019.

Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 1753–1759. AAAI Press, 2017a.

Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep clustering with convolutional autoencoders. In *International conference on neural information processing*, pp. 373–382. Springer, 2017b.

Xifeng Guo, Xinwang Liu, En Zhu, Xinzhong Zhu, Miaomiao Li, Xin Xu, and Jianping Yin. Adaptive self-paced deep clustering with data augmentation. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

Divam Gupta, Ramachandran Ramjee, Nipun Kwatra, and Muthian Sivathanu. Unsupervised clustering using pseudo-semi-supervised learning. In *International Conference on Learning Representations*, 2019.

Philip Haeusser, Johannes Plapp, Vladimir Golkov, Elie Aljalbout, and Daniel Cremers. Associative deep clustering: Training a classification network with no labels. In *German Conference on Pattern Recognition*, pp. 18–32. Springer, 2018.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

John R Hershey, Jonathan Le Roux, and Felix Weninger. Deep unfolding: Model-based inspiration of novel deep architectures. *arXiv preprint arXiv:1409.2574*, 2014.

Weihua Hu, Takeru Miyato, Seiya Tokui, Eiichi Matsumoto, and Masashi Sugiyama. Learning discrete representations via information maximizing self-augmented training. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1558–1567. JMLR. org, 2017.

Pan Ji, Tong Zhang, Hongdong Li, Mathieu Salzmann, and Ian Reid. Deep subspace clustering networks. In *Advances in Neural Information Processing Systems*, pp. 24–33, 2017.

Xu Ji, João F Henriques, and Andrea Vedaldi. Invariant information clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9865–9874, 2019.

Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: an unsupervised and generative approach to clustering. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 1965–1972, 2017.

Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey. *arXiv preprint arXiv:1902.06162*, 2019.

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

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Xia Li, Zhisheng Zhong, Jianlong Wu, Yibo Yang, Zhouchen Lin, and Hong Liu. Expectation-maximization attention networks for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9167–9176, 2019.

Percy Liang and Dan Klein. Online em for unsupervised models. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, pp. 611–619, 2009.

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

Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International conference on artificial neural networks*, pp. 52–59. Springer, 2011.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Xi Peng, Shijie Xiao, Jiashi Feng, Wei-Yun Yau, and Zhang Yi. Deep subspace clustering with sparsity prior. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pp. 1925–1931, 2016.

Douglas A Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, 741, 2009.

Guy Shiran and Daphna Weinshall. Multi-modal deep clustering: Unsupervised partitioning of images. *arXiv preprint arXiv:1912.02678*, 2019.

Chunfeng Song, Feng Liu, Yongzhen Huang, Liang Wang, and Tieniu Tan. Auto-encoder based data clustering. In *Iberoamerican Congress on Pattern Recognition*, pp. 117–124. Springer, 2013.

Kai Tian, Shuigeng Zhou, and Jihong Guan. Deepcluster: a general clustering framework based on deep learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 809–825. Springer, 2017.

Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning. *arXiv preprint arXiv:2005.10243*, 2020.

D Michael Titterington. Recursive parameter estimation using incomplete data. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(2):257–267, 1984.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7794–7803, 2018.

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

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pp. 478–487, 2016.

Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3861–3870. JMLR. org, 2017.

Jianwei Yang, Devi Parikh, and Dhruv Batra. Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5147–5156, 2016.

Yan Yang, Jian Sun, Huibin Li, and Zongben Xu. Admm-csnet: A deep learning approach for image compressive sensing. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

Dejiao Zhang, Yifan Sun, Brian Eriksson, and Laura Balzano. Deep unsupervised clustering using mixture of autoencoders. *arXiv preprint arXiv:1712.07788*, 2017.

Qin Zhang, Jia Wu, Peng Zhang, Guodong Long, and Chengqi Zhang. Salient subsequence learning for time series clustering. *IEEE transactions on pattern analysis and machine intelligence*, 41(9): 2193–2207, 2018.

Junbo Zhao, Michael Mathieu, Ross Goroshin, and Yann Lecun. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*, 2015.

A THE DEFINITION OF TWO METRICS: ACC AND NMI

- **Accuracy (ACC):** For sample i, let R_i denote its ground truth label and C_i be its label obtained by clustering.

$$ACC = \frac{\sum_{n=1}^{N} \delta(R_n, C_n)}{N} \times 100\%, \quad \text{where} \delta(x, y) = \begin{cases} 1 & x = y \\ 0 & \text{otherwise} \end{cases},$$

  N denotes the total number of samples.

- **Normalized mutual information (NMI):** Let R denote the ground truth label and C be the label obtained by clustering. The NMI is defined as follows:

$$NMI = \frac{2MI(R, C)}{H(R) + H(C)},$$

  where $H(X)$ is the entropy of X, and $MI(X, Y)$ is the mutual information of X and Y.

B DISCUSSION ON THE ADVANTAGE OF CLUSTER REWEIGH STRATEGY

First of all, we give the detailed derivation of equation (7) in the following for better understanding.

$$\begin{aligned}
\mathcal{L}_{cluster}(\boldsymbol{\mu}) &= -\sum_{n=1}^{N}\sum_{k=1}^{K} \lambda_{nk} \ln[\frac{1}{K}\mathcal{N}(z_n|\mu_k, \mathbb{I})] \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K} \lambda_{nk} \ln K - \sum_{n=1}^{N}\sum_{k=1}^{K} \lambda_{nk} \ln \frac{1}{\sqrt{(2\pi)^d|\mathbb{I}|}} \mathrm{e}^{-\frac{1}{2}(z_n-\mu_k)'\mathbb{I}^{-1}(z_n-\mu_k)} \\
&= (\ln K + \frac{d}{2}\ln(2\pi)) \sum_{n=1}^{N}\sum_{k=1}^{K} \lambda_{nk} + \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K} \lambda_{nk}\|z_n - \mu_k\|_2^2 \\
&\approx \boxed{N \ln K + \frac{Nd}{2}\ln(2\pi)} + \frac{1}{2}\sum_{n=1}^{N} \boxed{\frac{N}{KN_k}} \sum_{k=1}^{K} \lambda_{nk}\|z_n - \mu_k\|_2^2.
\end{aligned}$$

where $N = \sum_{n=1}^{N}\sum_{k=1}^{K} \lambda_{nk}$. $N \ln K + \frac{Nd}{2}\ln(2\pi)$ is constant.

Note that the vanilla k-means is designed for the scenario when the samples from different clusters are balanced, while vanilla GMM would enhance the strength of the majority cluster with the group ratio π. However, when samples from different clusters are imbalanced and the minority cluster matters, both k-means and GMM would underestimate the importance of minority clusters and output inferior clustering performance. Therefore, we propose to pay more attention to the minority clusters, by increasing the penalty of wrongly clustering the sample from a minority cluster and decreasing the penalty of wrongly clustering the sample from a majority cluster.

To be specific, we suggest to reweigh the within-group variance in Eq.(8) with

$$w_k = \frac{N}{KN_k} = \frac{\bar{N}}{N_k}, \quad k = 1, \ldots, K,$$

where N is the number of samples, K is the number of clusters, $\bar{N}$ is the average number of samples for each cluster, and N_k is the number of samples belonging to cluster k. Then we have:

- $w_k \approx 1, k = 1, 2, \ldots, K$. When samples from different clusters are balanced, the reweigh becomes invalid and Eq.(8) reduces to regular cluster loss.
- $w_k > 1$. For a minority cluster, we increase the penalty of its within-group variance, being inversely proportional to its group size.
- $w_k < 1$. For a majority cluster, we decrease the penalty of its within-group variance, also being inversely proportional to its group size. Therefore, the loss with regards to the majority cluster would not dominate the whole training process.

In terms of the mini-batch update, the data statistics N, K, N_k for the whole dataset is not available. We replace it with the data statistics N', K', N'_k collected on each mini-batch data.

$$w_k = \frac{N'/K' + \Delta}{N'_k + \Delta}, \quad k = 1, 2, \ldots, K.$$

To avoid instability, we introduce the Δ which is empirically set to 3 in the experiment.

C EXPERIMENT SETTING

Table 5: Network structure for all dataset

Network	MNIST / USPS	YTF / Fashion
Encoder	(0): Dropout(p=0.25, inplace=False) (1): Linear(dim, 500, bias=True) (2): ReLU() (3): Linear(500, 500, bias=True) (4): ReLU() (5): Linear(500, 2000, bias=True) (6): ReLU() (7): Linear(2000, 10, bias=True)	(0): Conv2d(channel, 16, kernel_size=3, stride=1, padding=1) (1): BatchNorm2d(16) (2): ReLU() (3): Conv2d(16, 32, kernel_size=3, stride=2, padding=1) (4): BatchNorm2d(32,) (5): ReLU() (6): Conv2d(32, 32, kernel_size=3, stride=1, padding=1) (7): BatchNorm2d(32) (8): ReLU() (9): Conv2d(32, 16, kernel_size=3, stride=2, padding=1) (10): BatchNorm2d(16) (11): ReLU() (12): Linear(Inner-dim, 256, bias=True) (13): ReLU() (14): Linear(256, 10, bias=True)
Streamlining EM	Iter 0: E-step (Eq.(3)) Iter 1:T-1: For i in range(T-1): Moment update (Eq.(5)) Correction loss (Eq.(7)) E-step (Eq.(3)) Iter T: Reconstruction (Eq.(9))	
Decoder	(0)Linear(10, 2000, bias=True) (1): ReLU() (2): Linear(2000, 500, bias=True) (3): ReLU() (4): Linear(500, 500, bias=True) (5): ReLU() (6): Linear(500, dim, bias=True) (7): Sigmoid()	(0): Linear(10, 256, bias=True) (1): ReLU() (2): Linear(256, Inner-dim, bias=True) (3): ConvTranspose2d(16, 32, kernel_size=3, stride=2, padding=1) (4): BatchNorm2d(32) (5): ReLU() (6): ConvTranspose2d(32, 32, kernel_size=3, stride=1, padding=1) (7): BatchNorm2d(32) (8): ReLU() (9): ConvTranspose2d(32, 16, kernel_size=3, stride=2, padding=1) (10): BatchNorm2d(16) (11): ReLU() (12): ConvTranspose2d(16, channel, kernel_size=3, stride=1)

D DISCUSSION ON THE ADVANTAGES OF STREAMLINING EM FOR GMM

We streamline the EM algorithm of the Gaussian mixture model and derive a differential Gaussian mixture network for clustering. In the following, we compare it to the standard/stochastic EM algorithm of the Gaussian mixture model to show its superiority.

- **Mini-batch update.** Different from the standard EM algorithm which requires the full batch to update, we adopt the moment update formulation in Eq.(4) which allows the mini-batch noisy update. Meanwhile, the mini-batch update is helpful to escape the local minimum, yielding a better solution (Liang & Klein, 2009).
- **Streamlining alternative updates into forward propagation.** Many online/stochastic EM algorithms have been proposed to apply the GMM model for large-scale applications. However, all of them still adopt an alternative update paradigm between the E-step and M-step. It still restricts its efficiency when joint learning GMM with other learning tasks. On

the contrary, we streamline the EM algorithm of GMM into a network design and replace the M-step with a correction on the loss. Therefore, our differential Gaussian mixture network can be end-to-end updated.

- **Portability with popular deep learning platform.** Since our differential Gaussian mixture network can be end-to-end updated with standard stochastic gradient descent, it can be easily implemented with popular deep learning platforms. To be specific, we initialize network work weights, i.e., the centroids, as usual. Then, the forward propagation executes the E-step (Eq.(3)) and the modified M-step (Eq.(4)) alternatively in T times. After the loss being calculated following Eq.(7), we can optimize the network weight with any integrated optimizer. In particular, we implement our work with PyTorch and adopt Adam to optimize the network weight.
- **Backbone architecture for capturing multiple modalities.** Similar to other backbone architectures, our differential Gaussian mixture network can also be incorporated into existing network structures. In particular, it helps to induce a latent space with a significant group structure. Meanwhile, due to the skip connection-based reconstruction, i.e., Eq.(9), the whole structure can capture the common group information in the latent space with little information loss. From the overall structure, it is similar to the non-local network (Wang et al., 2018), but enjoys more interpretability (Li et al., 2019).

E Parameter Initialization

There are two types of parameters in our EDGaM network, i.e., neural network weights and trade-off parameters. In the following, we discuss the initialization of these two types of parameters, respectively.

E.1 Neural network weights

All neural network weight, including the cluster centroids, are initialized using a uniform distribution following (Glorot & Bengio, 2010).

E.2 Trade-off parameters

There are five trade-off hyperparameters in EDGaM: β_1, β_2, used in Eq.(7), and η_1, η_2, η_3, used in Eq.(10). The hyperparameter setting for four dataset are summarized in Table 6.

Table 6: Hyperparameter setting for four datasets

Hyperparameter	β_1	β_2	η_1	η_2	η_3
MNIST	0.9	0.9	10^{-2}	10^{-2}	10^{-4}
USPS	0.9	0.9	10^{-1}	5×10^{-2}	10^{-4}
YTF	0.9	0.9	5×10^{-2}	5×10^{-2}	10^{-5}
Fashion	0.9	0.9	5×10^{-2}	10^{-3}	10^{-4}

The learnable parameters β_1, β_2 is initialized to 0.9 for all four datasets, which can be gradually adjusted during the learning process.

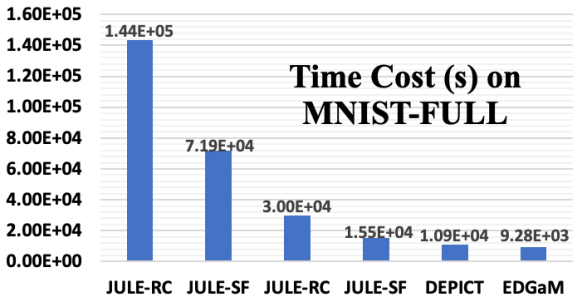
Some hits for initializing the trade-off hyperparameters η_1, η_2, η_3:

1. When a small validation dataset is available, the validation dataset can be used for initializing the hyperparameter. In particular, we find that the cluster accuracy calculated with the soft assignment in EDGaM is close to the k-means cluster accuracy on the obtained latent embedding when converge. It means we can online compare the clustering performance instead of evaluating it offline each time, which will improve the efficiency for setting the hyperparameters.
2. A suggest hyperparameter setting is $\eta_1 = 10^{-2}, \eta_2 = 10^{-2}, \eta_3 = 10^{-4}$. For a complex dataset η_1 should be larger, e.g., YTF and Fashion. For a small dataset η_1, η_2 should be larger, e.g., USPS and YTF.

3. Since EDGaM relies on the AE structure to extract the nonlinear features, the hyperparameter setting should not hinder the reconstruction loss from achieving its optimum values. Especially, the skip structure ensures the reconstruction loss can achieve its optimum.
4. The trade-off hyperparameters of the entropy loss (η_3) should be appropriate, to ensure the average of the maximum group assignment $\frac{1}{N}\sum_n \max_k \lambda_{nk}$ is around 0.5 in the first few iterations. The entropy loss would lead the average of the maximum group assignment $\frac{1}{N}\sum_n \max_k \lambda_{nk}$ to approximate 1 gradually during the learning process.

F TIME EFFICIENCY

To evaluate the efficiency of our EDGaM in dealing with large-scale (7×10^4) and high dimensional (784) data, we compare EDGaM its most competing algorithms JULE and DEPICT. All four versions of JULE are evaluated. We run JULE and DEPICT using their released codes, respectively. In particular, we run our EDGaM and other baselines for 10^3 iterations on the cluster (GeForce RTX 2080 Ti), and collect the time cost of each method, respectively. The mini-batch size is set to 128 for all methods.



The time cost of all methods is consistent with our analysis in Sect. 4. Note since the computation cost of EDGaM is comparable with DEPICT since the extra complexity introduced by EDGaM and DEPICT is not significant compared to that of the basic AE framework during the whole training process. However, due to the lack of an efficient mechanism for dealing with imbalanced datasets, DEPICT suggests adopting the highly energy-consuming agglomerative clustering instead of k-means to ensure good performance. Therefore, DEPICT will suffer the same issues as JULE for large-data imbalanced datasets.

G FAILURE CASES ANALYSIS FOR UNCERTAIN IMAGE CLUSTERING

In Fig. 6, we select the images with the ground truth label 3, 4, 5, and sort them in descending order according to its group assignment of the ground-truth cluster. It is clear that our EDGaM can confidently (high λ) group the images to its correct cluster as long as images show sufficient identities. Otherwise, EDGaM would wrongly group the images to the corresponding cluster which they really look like, for which even humans would make a similar guess.

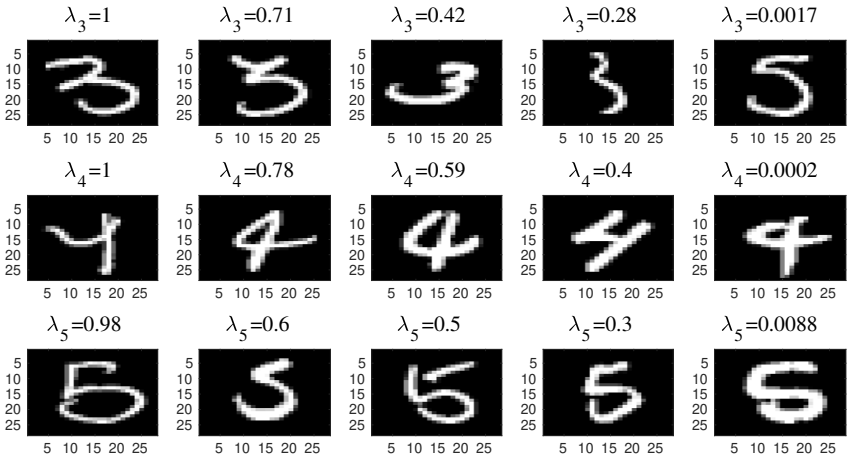


Figure 6: Uncertainty image clustering with soft assignment