

---

# Appendix for “Streamlining EM into Auto-Encoder Networks”

## 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 REWEIGHT STRATEGY

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  $\pi$ . 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 reweight the within-group variance in Eq.(8) with

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

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 belongs to cluster  $k$ . Then we have:

- $w_k \approx 1$ ,  $k = 1, 2, \dots, K$ . When samples from different clusters are balanced, the reweight 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.

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

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

## C EXPERIMENT SETTING

## 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.

Table 1: 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)

- **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:  $\beta_1, \beta_2$ , used in Eq.(7), and  $\eta_1, \eta_2, \eta_3$ , used in Eq.(10). The hyperparameter setting for four dataset are summarized in Table 2.

Table 2: Hyperparameter setting for four datasets

Hyperparameter	$\beta_1$	$\beta_2$	$\eta_1$	$\eta_2$	$\eta_3$
MNIST	0.9	0.9	$10^{-2}$	$10^{-2}$	$10^{-4}$
USPS	0.9	0.9	$10^{-1}$	$5 \times 10^{-2}$	$10^{-4}$
YTF	0.9	0.9	$5 \times 10^{-2}$	$5 \times 10^{-2}$	$10^{-5}$
Fashion	0.9	0.9	$5 \times 10^{-2}$	$10^{-3}$	$10^{-4}$

The learnable parameters  $\beta_1, \beta_2$  is initialized to 0.9 for all four datasets, which can be gradually adjusted during the learning process.

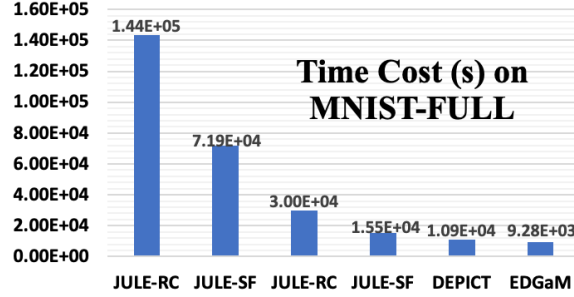
Some hits for initializing the trade-off hyperparameters  $\eta_1, \eta_2, \eta_3$ :

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  $\eta_1$  should be larger, e.g., YTF and Fashion. For a small dataset  $\eta_1, \eta_2$  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 ( $\eta_3$ ) 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 ( $7 \times 10^4$ ) 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. ?? . 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. 1, 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  $\lambda$ ) 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.

## REFERENCES

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.

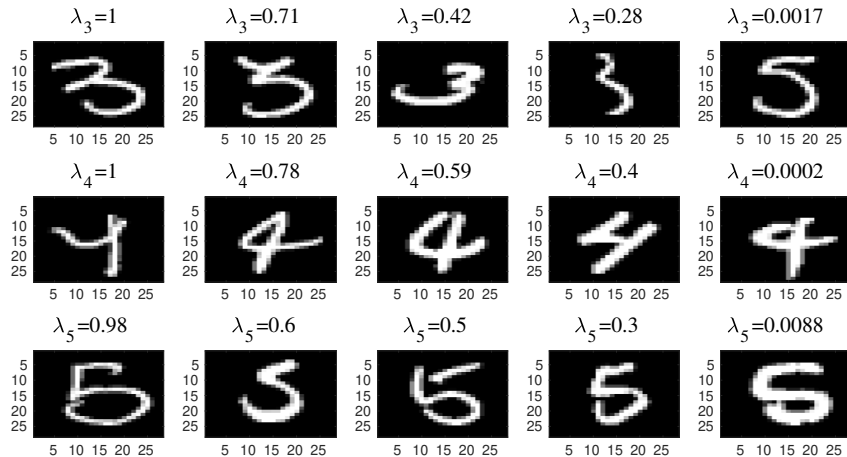


Figure 1: Uncertainty image clustering with soft assignment

- 
- 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.
- 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.