Deep Dirichlet Process Mixture Models

Naiqi Li[*1] **Wenjie Li**[*1] **Yong Jiang**[1,2] **Shu-Tao Xia**[1,2]

[1]Tsinghua Shenzhen International Graduate School, Tsinghua University, China
[2]Peng Cheng Laboratory, Shenzhen, China

Abstract

In this paper we propose the deep Dirichlet process mixture (DDPM) model, which is an unsupervised method that simultaneously performs clustering and feature learning. The traditional Dirichlet process mixture model can infer the number of mixture components, but its flexibility is restricted since the clustering is performed in the raw feature space. Our method alleviates this limitation by using the flow-based deep neural network to learn more expressive features. DDPM unifies Dirichlet processes and the flow-based model with Monte Carlo expectation-maximization, and uses Gibbs sampling to sample from the posterior. This combination allows our method to exploit the mutually beneficial relation between clustering and feature learning. The effectiveness of DDPM is demonstrated by thorough experiments in various synthetic and real-world datasets.

1 INTRODUCTION

Clustering is one of the most long-standing and fundamental tasks in computer science. Besides the well-known k-means algorithm [MacQueen, 1967] and Gaussian mixture models (GMMs) [Bishop, 2006], a plethora of methods have been proposed [Ester et al., 1996, Szekely et al., 2005, Frey and Dueck, 2007, Zhao et al., 2008]. In those early investigations, clustering is performed in the raw feature space, and the models lack the capacity of learning or improving the expressiveness of the representation.

With the recent success of deep neural networks (DNNs), a new line of research termed deep clustering has emerged [Xie et al., 2016, Yang et al., 2016, Jiang et al., 2017, Caron et al., 2018]. These works are based on the intuitive principle that good representation encourages better clustering, and similarly, good clustering can lead to better representation. Their methods take advantage of the successful deep neural network structures, such as convolution neural networks (CNNs) [Krizhevsky et al., 2012] and variational autoencoders (VAEs) [Kingma and Welling, 2014]. Their superior performance demonstrates the benefits of jointly clustering and representation learning.

However, all these methods share the same shortcoming – they consider the number of clusters (*i.e.*, k in k-means) as a hyperparameter that needs to be specified by the user. This brings severe restrictions to their applications: 1) most of these algorithms are sensitive to the choice of k; 2) users lack the prior knowledge of the number of clusters; 3) this value itself may be time-varying (*e.g.*, k may increase as more data is accumulated); 4) in some scenarios, particularly for large datasets, there is no golden ground truth for this value [Li et al., 2018].

The Dirichlet process mixture (DPM) model, which belongs to the Bayesian nonparametric family, is a popular method that can solve this conundrum [Antoniak, 1974]. Its solid mathematical background originates from the Dirichlet process [Ferguson, 1973], which is an infinite generalization of the Dirichlet distribution. DPM can model the data with a possibly infinite number of mixtures, and the exact number of mixtures is rigorously inferred by the Bayesian principle. It is thus desirable to combine the strengths of DPM and deep neural networks, which could lead to a clustering method that simultaneously adjusts the number of mixture components and learns better representation.

In this paper, we propose the deep Dirichlet process mixture (DDPM) model, which brings the above idea into realization. Two working examples of DDPM are presented in Fig. 1, where the potentials of DDPM are clearly demonstrated, particularly its ability to enjoy the mutually beneficial relationship between clustering and feature learning. Our method bridges the standard DPM with the recently proposed flow-based generative models [Dinh et al., 2015,

[*]Equal contributions.

Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022), PMLR 180:1138–1147.

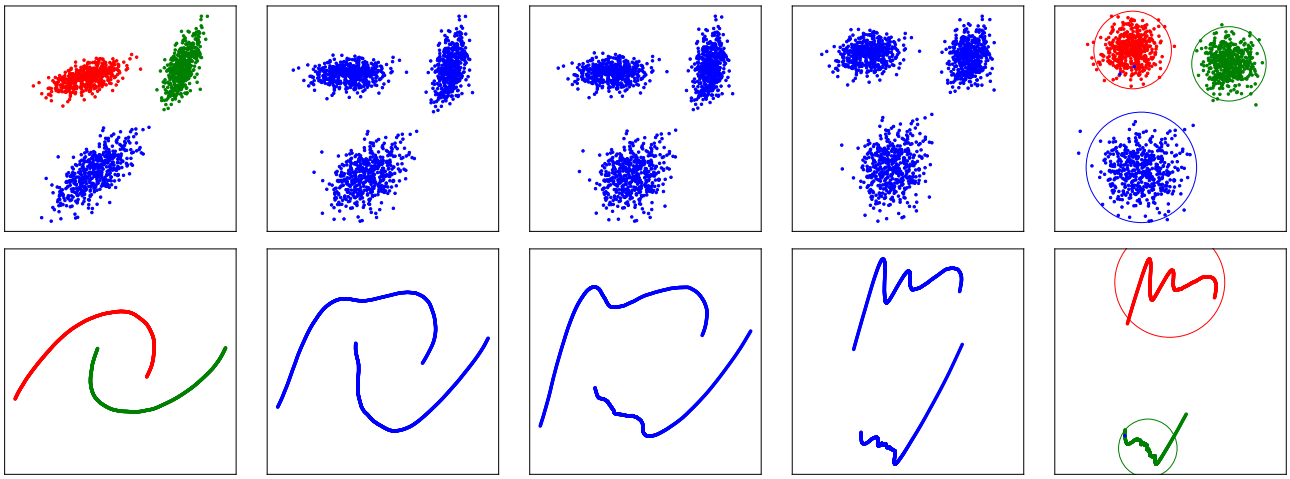


Figure 1: Demonstration of the clustering and representation learning process on two synthetic datasets. The leftmost figures are the ground truth clustering results. The middle figures show the latent representation learned by DDPM during the training. The rightmost figures show the final clustering results, with the circles denoting 2 standard deviations of the Gaussian distributions. We can see that DDPM is able to learn better representation during clustering. Particularly in the second example, the raw data representation is challenging for many centroid-based clustering methods, and the benefit of the new representation learned by DDPM is quite evident. Also note that the number of clusters is unknown in advance.

Kingma and Dhariwal, 2018], which is a special kind of invertible deep neural network that learns better representation through density transformation. The overall clustering process is guided by the Bayesian principle, while the optimization of model parameters is derived from the Monte Carlo expectation-maximization (EM) algorithm [Bishop, 2006]. During the iterations, Gibbs sampling is used to obtain samples from the posterior as in the standard DPM literature. DDPM also works as a generative model, so that unseen new samples could be obtained by introducing noises to the learned features. The source code for reproducing our main experiments is publicly available at `https://github.com/naiqili/DDPM`.

2 RELATED WORK

Clustering and representation learning are both among the most well-investigated topics in computer science. Besides the well-known k-means [MacQueen, 1967], GMMs [Bishop, 2006], and their variants [Zhao et al., 2008, Li et al., 2021], the recent success of deep neural networks have inspired a new paradigm called deep clustering. The work of [Yang et al., 2016, Caron et al., 2018] tries to take advantage of the convolution neural networks in computer vision tasks. Their major difference lies in the loss function and the training scheme. Other DNN structures are also exploited. Xie et al. [2016] proposed Deep Embedded Clustering (DEC), which jointly optimizes deep embeddings and performs clustering based on features extracted from deep autoencoders. In [Jiang et al., 2017], Variational Deep Embedding (VaDE) was introduced, which combines the variational autoencoder with the GMM model. However all these works share the same insufficiency, *i.e.*, the number of the clusters is a hyperparameter that needs to be specified by the user. As aforementioned this presents challenges for their applications in real-world scenarios, where prior knowledge about the number of clusters is generally unavailable.

The dirichlet process mixture model [Antoniak, 1974], which belongs to the Bayesian nonparametric family, is one of the most popular methods that are capable of inferring the number of clusters automatically. This distinguishing ability is further utilized and strengthened. Teh et al. [2006] proposed Hierarchical Dirichlet processes, which can share mixture components among different clusters. The maximum margin DPM (MMDPM) introduces a discriminate model for clustering, which bridges DPM and the SVM classifier [Chen et al., 2016]. All these methods operate in the raw feature space, without the ability to learn more expressive representation. Recently Echraibi et al. [2020] proposed the Dirichlet process deep latent Gaussian mixture model (DP-DLGMM), which combines the Dirichlet process prior with the deep latent Gaussian mixture model so that the number of mixture components can be adjusted. However, their work focuses on representation learning rather than clustering.

Naturally one may wonder whether it is possible to simultaneously infer the number of mixture components and learn better (possibly nonlinear) features. Ehsan Abbasnejad et al. [2017] proposed to use an infinite mixture of VAEs to model the data. Since the number of effective VAEs may increase as more data arrives, their model can adapt to the complexity of the data. In the paper of [Nalisnick and Smyth, 2017],

stick-breaking variational autoencoders (SB-VAEs) were presented, which model the latent variables in VAEs to be infinite-dimensional. Dirichlet processes and particularly the stick-breaking construction serve as the cornerstones in their method. Our work distinguishes from theirs in both goal and methodology: their works focus on improving the performance of semi-supervised classification tasks, while our research considers the unsupervised task of clustering where the number of mixtures and better representation needs to be jointly learned. Both of their models are based on VAEs while our method utilizes the recently proposed flow-based invertible deep neural network, which demonstrates superior performance in various density estimation and computer vision tasks [Dinh et al., 2015, Kingma and Dhariwal, 2018].

3 METHODOLOGY

3.1 OVERVIEW

Consider the input as a set $\mathbf{X} = \{\mathbf{x}_i \in \mathbb{R}^D\}_{i=1}^N$. Our first step is to use a standard dimension reduction technique to extract representative features, so that the following clustering can be performed in the lower dimensional **feature space**. In our work we use the stack autoencoder [Vincent et al., 2010, Xie et al., 2016] to extract the features as $\mathbf{Y} = \{\mathbf{y}_i = h_e(\mathbf{x}_i) \in \mathbb{R}^d\}_{i=1}^N$ ($d \ll D$), where h_e and h_d denote the encoder and decoder functions respectively, such that $h_d(h_e(\mathbf{x})) \approx \mathbf{x}$. Next the features are transformed by a nonlinear learnable function $f(\mathbf{y}; \boldsymbol{\theta})$ into $\mathbf{Z} = \{\mathbf{z}_i = f(\mathbf{y}_i; \boldsymbol{\theta}) \in \mathbb{R}^d\}_{i=1}^N$. Clustering is performed in this **transformed space**. We assume that each $\mathbf{z}_i$ follows an isotropic Gaussian distribution. Suppose that $\mathbf{z}_i$ belongs to the k-th cluster, the likelihood is given by:

$$\begin{aligned} p(\mathbf{z}_i|\boldsymbol{\mu}_k, \lambda_k) &= \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_k, \lambda_k^{-1}\mathbf{I}) \\ &= (\frac{\lambda_k}{2\pi})^{\frac{d}{2}} \exp\left(-\frac{\lambda_k}{2}||\mathbf{z}_i - \boldsymbol{\mu}_k||^2\right), \end{aligned} \quad (1)$$

where $\boldsymbol{\mu}_k$ and λ_k denote the mean and the precision of the k-th cluster. Note that the isotropic Gaussian assumption does not restrict the model's capacity since the transformation $f(\mathbf{y}_i; \boldsymbol{\theta})$ is nonlinear, which is implemented by a deep neural network in practice. The cluster assignment variables are denoted as $\mathbf{c} = \{c_i \in \{1, ..., K\}\}_{i=1}^N$, where $c_i = k$ indicates that $\mathbf{z}_i$ belongs to the k-th cluster. Here the total number of clusters K is unknown to us and could be arbitrarily large.

From the Bayesian perspective, the task of clustering is equivalent to the inference of

$$p(\mathbf{c}, \{\boldsymbol{\mu}_k\}_{k=1}^K, \{\lambda_k\}_{k=1}^K|\mathbf{Y}; \boldsymbol{\theta}, \boldsymbol{\Phi}). \quad (2)$$

Here $\boldsymbol{\Phi}$ is the set of hyperparameters that specify the prior, which will soon be introduced in the next section. We emphasize the challenges of the task: 1) the number of clusters K is unknown; 2) $\boldsymbol{\theta}$ parameterizes a deep neural network which needs to be learned; 3) the cluster information (*i.e.*, $\boldsymbol{\mu}_k$ and λ_k) need to be computed at the same time. In what follows we will see how the proposed method can address all these challenges under a unified framework.

3.2 MODEL SPECIFICATION

Likelihood in the feature space Eq. (1) describes the likelihood function in the transformed space. Now we derives the likelihood in the feature space (*i.e.*, before the transformation) as follows:

$$\begin{aligned} &p(\mathbf{Y}|\mathbf{c}, \{\boldsymbol{\mu}_k\}_{k=1}^K, \{\lambda_k\}_{k=1}^K; \boldsymbol{\theta}, \boldsymbol{\Phi}) \\ &= \prod p(\mathbf{y}_i|\boldsymbol{\mu}_{c_i}, \lambda_{c_i}; \boldsymbol{\theta}, \boldsymbol{\Phi}) \\ &= \prod p(\mathbf{z}_i|\boldsymbol{\mu}_{c_i}, \lambda_{c_i})|\det \frac{\partial f(\mathbf{y}_i; \boldsymbol{\theta})}{\partial \mathbf{y}_i}|. \end{aligned} \quad (3)$$

The last term in Eq. (3) is due to the change of variable $\mathbf{z}_i = f(\mathbf{y}_i; \boldsymbol{\theta})$. Recall that $p(\mathbf{z}_i|\boldsymbol{\mu}_{c_i}, \lambda_{c_i})$ is given in Eq (1).

For a general nonlinear function $f(\mathbf{y}_i; \boldsymbol{\theta})$ implemented by a neural network, the Jacobian term in Eq. (3) is analytically intractable. To address this problem we utilize the NICE model [Dinh et al., 2015], which is a flow-based deep neural network with the appealing property that the determinant of Jacobian can be trivially computed. The basic idea of NICE is that, in each layer it splits the input $\mathbf{y}$ into two parts as $\mathbf{y} = \{\mathbf{y}_1, \mathbf{y}_2\}$, and defines the output as $\mathbf{z} = \{\mathbf{z}_1, \mathbf{z}_2\}$ where $\mathbf{z}_1 = \mathbf{y}_1$ and $\mathbf{z}_2 = \mathbf{y}_2 + \sigma(\mathbf{y}_1)$. It is easy to verify that for such function the determinant of Jacobian equals one. After stacking multiple such layers, we have a highly nonlinear function whose Jacobian term can be trivially cancelled out. Another useful property of NICE is that the transformation is invertible. Particularly if $\mathbf{z} = f(\mathbf{y}; \boldsymbol{\theta})$, $\mathbf{y} = f^{-1}(\mathbf{z}; \boldsymbol{\theta})$ can also be easily computed. Interested readers may refer to [Dinh et al., 2015] for further details.

Dirichlet process mixture model in the transformed space After the transformation $\mathbf{z}_i = f(\mathbf{y}_i; \boldsymbol{\theta})$, we can now perform clustering in this transformed space. In our problem the number of clusters K is unknown. The Dirichlet process mixture (DPM) models [Antoniak, 1974, Neal, 2000, Li et al., 2019] is one of the most popular tools in this situation. Here we present a concise review of DPM models, and in the next subsection we will see how to connect it with NICE.

Given a measurable space $(\Theta, \mathcal{A})$ where $\mathcal{A}$ is a σ-algebra defined on Θ, the Dirichlet process (DP) [Ferguson, 1973] is characterized by a probability measure G_0 on the measure space, and a positive scaling parameter α. A DP is a random probability measure over $(\Theta, \mathcal{A})$, denoted as $G \sim DP(G_0, \alpha)$, such that for any partition $(A_1, ..., A_r)$ of Θ we have

$$(G(A_1), ..., G(A_r)) \sim Dir(\alpha_0 G_0(A_1), ..., \alpha_0 G_0(A_r)),$$

where Dir is the finite-dimensional Dirichlet distribution. In other words, a DP is a "distribution over distribution".

The DPM model is based on DP. Under our formulation, it is defined as

$$G|G_0, \alpha \sim DP(G_0, \alpha) \tag{4}$$
$$\boldsymbol{\mu}_k, \lambda_k|G \sim G(\boldsymbol{\mu}_k, \lambda_k) \tag{5}$$
$$\mathbf{z}_i|\boldsymbol{\mu}_k, \lambda_k \sim p(\mathbf{z}_i|\boldsymbol{\mu}_k, \lambda_k) \tag{6}$$

The likelihood in (6) is given in Eq. (1), *i.e.*, in our work the DPM model is applied in the transformed space $\mathbf{Z} = \{\mathbf{z}_i = f(\mathbf{y}_i; \boldsymbol{\theta})\}_{i=1}^N$. We define the base distribution G_0 as the conjugate prior of the likelihood, which is the normal-gamma distribution [Bishop, 2006][1]:

$$\begin{aligned}\boldsymbol{\mu}_k,&\lambda_k|\boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0 \sim NG(\boldsymbol{\mu}_k, \lambda_k|\boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0)\\ &= \mathcal{N}(\boldsymbol{\mu}_k|\boldsymbol{\mu}_0, (\kappa_0\lambda_k)^{-1}\mathbf{I}) Gamma(\lambda_k|\alpha_0, \beta_0). \end{aligned} \tag{7}$$

A key property of DPM models is that the marginalized conditional distribution of the cluster assignment variable has closed from:

$$p\left(c_i = k \mid \mathbf{c}_{-i}, \alpha\right) = \begin{cases} \dfrac{n_{-i,k}}{N-1+\alpha}, & n_{-i,k} > 0 \\ \dfrac{\alpha}{N-1+\alpha}, & n_{-i,k} = 0 \end{cases} \tag{8}$$

where $\mathbf{c}_{-i} = \mathbf{c} \setminus \{c_i\}$, N is the number of all data points, and $n_{-i,k}$ is the size of the k-th cluster excluding the i-th datum. Intuitively the first formula is the probability of assigning the i-th datum into the k-th existing cluster, while the second formula is the probability of assigning it to a new cluster. The proof of this result is available in many related literature [Görür and Rasmussen, 2010, Chen et al., 2016, Li et al., 2019].

We collect all the hyperparameters in the DPM model as $\boldsymbol{\Phi} = \{\alpha, \boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0\}$, which was used in Eq. (2). To keep the representation succinct we will suppress $\boldsymbol{\Phi}$ in the following discussions, unless it is explicitly needed.

3.3 UNIFIED PARAMETER ESTIMATION

To combine NICE and DPM models, the key question is how to learn the deep neural network, or in other words how to optimize the parameters $\boldsymbol{\theta}$ in $f(\mathbf{y}_i; \boldsymbol{\theta})$. In this subsection we will address this challenge with the Monte Carlo expectation-maximization (MC-EM) algorithm. The whole process is summarized in Algorithm 1.

3.3.1 The Overall MC-EM Framework

We consider $\mathbf{c}$, $\{\boldsymbol{\mu}_k\}_{k=1}^K$ and $\{\lambda_k\}_{k=1}^K$ in Eq. (2) as hidden variables, $\mathbf{Y}$ as the observed variables, and $\boldsymbol{\theta}$ as the set of parameters need to be optimized. To keep the representation succinct, we denote $\mathbf{H} = \{\{\boldsymbol{\mu}_k\}_{k=1}^K, \{\lambda_k\}_{k=1}^K\}$. Following the maximal likelihood principle, the optimal $\boldsymbol{\theta}^*$ is:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\mathbf{Y}|\boldsymbol{\theta}). \tag{9}$$

This can be solved by the expectation-maximization (EM) algorithm [Dempster et al., 1977], which iterates between the E-step and M-step until converges:

$$\text{E: } Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(old)}) = E_{\mathbf{H},\mathbf{c}|\mathbf{Y},\boldsymbol{\theta}^{(old)}}[\log p(\mathbf{H}, \mathbf{c}, \mathbf{Y}|\boldsymbol{\theta})] \tag{10}$$
$$\text{M: } \boldsymbol{\theta}^{(new)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(old)}) \tag{11}$$
$$\boldsymbol{\theta}^{(old)} \leftarrow \boldsymbol{\theta}^{(new)} \tag{12}$$

In the case that the E-step (10) has no closed-form solution, it can be numerically estimated as

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(old)}) \approx \frac{1}{G} \sum_g \log p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}|\boldsymbol{\theta}), \tag{13}$$

where $\mathbf{H}^{(g)}, \mathbf{c}^{(g)} \sim p(\mathbf{H}, \mathbf{c}|\mathbf{Y}, \boldsymbol{\theta}^{(old)})$ are i.i.d. samples and G is the sample size. This method is called the Monte Carlo EM algorithm [Bishop, 2006]. As $\boldsymbol{\theta}$ denotes the parameters of a deep neural network, we can use stochastic gradient descent to find the maximal value in Eq. (11):

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \lambda_s \frac{\partial}{\partial \boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(old)}) \tag{14}$$
$$\approx \boldsymbol{\theta}_t + \frac{\lambda_s}{G} \sum_g \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}|\boldsymbol{\theta}), \tag{15}$$

where λ_s is the learning rate. Finally to complete the picture, we need to:

- Present the analytical form of the complete data likelihood $p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}|\boldsymbol{\theta})$, and particularly the derivative of the log-likelihood in Eq. (15);
- Develop a method to obtain the samples $\mathbf{H}^{(g)}, \mathbf{c}^{(g)} \sim p(\mathbf{H}, \mathbf{c}|\mathbf{Y}, \boldsymbol{\theta}^{(old)})$.

These two questions will be addressed respectively in the following two subsections.

3.3.2 The Complete Data Likelihood

With our discussions in the model specification section, the complete data likelihood can be derived as follows:

$$\begin{aligned} &p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}|\boldsymbol{\theta})\\ =&p(\mathbf{Y}|\mathbf{H}^{(g)}, \mathbf{c}^{(g)}; \boldsymbol{\theta}) p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)})\\ =&p(\mathbf{Z}|\mathbf{H}^{(g)}, \mathbf{c}^{(g)}) p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}) \prod_i |\det \frac{\partial f(\mathbf{y}_i; \boldsymbol{\theta})}{\partial \mathbf{y}_i}|\\ =&p(\mathbf{Z}|\mathbf{H}^{(g)}, \mathbf{c}^{(g)}) p(\mathbf{H}^{(g)}) p(\mathbf{c}^{(g)})\\ =&\prod_i p(\mathbf{z}_i|\boldsymbol{\mu}_{c_i}^{(g)}, \lambda_{c_i}^{(g)}) \prod_k p(\boldsymbol{\mu}_k^{(g)}, \lambda_k^{(g)}) p(\mathbf{c}^{(g)}), \end{aligned}$$

[1] In our work we consider $\boldsymbol{\mu}_k$ as a vector, while in the standard normal-gamma distribution it is a scalar.

Algorithm 1 DDPM($\mathbf{X}, h_e(\cdot), \mathbf{\Phi}, \lambda_s$)

Require: Input dataset $\mathbf{X}$; encoder $h_e(\cdot)$; hyperparameters $\mathbf{\Phi}$; learning rate λ_s
Ensure: Cluster parameters $\{\boldsymbol{\mu}_k, \lambda_k\}_{k=1}^K$; cluster assignment vector $\mathbf{c}$.

Initialize neural network's parameters $\boldsymbol{\theta}$
$\mathbf{Y} \leftarrow \{h_e(\mathbf{x}_i)|\mathbf{x}_i \in \mathbf{X}\}_{i=1}^N$.
for *epoch* in {1, ..., EPOCHS} **do**
  $\mathbf{Z}^{(old)} \leftarrow \{f(\mathbf{y}_i; \boldsymbol{\theta})\}_{i=1}^N$
  \\ E-step; iterations for the Gibbs sampling
  for t in {1, ..., GIBBS_STEPS} **do**
    For each k sample $\boldsymbol{\mu}_k, \lambda_k \sim p(\boldsymbol{\mu}_k, \lambda_k|\mathbf{H} \setminus \{\boldsymbol{\mu}_k, \lambda_k\}, \mathbf{c}, \mathbf{Z}^{(old)})$ by Eq. (17)
    For each i sample $c_i \sim p(c_i|\mathbf{c}_{-i}, \mathbf{Z}^{(old)}, \mathbf{H})$ by Eq. (18) and Eq. (19)
  end for
  For each k sample $\boldsymbol{\mu}_k^{(g)}, \lambda_k^{(g)}$, for each i sample $c_i^{(g)}$
  \\ M-step; optimization of the NICE model
  for t in {1, ..., OPT_STEPS} **do**
    Sample a batch $\mathbf{Y}^{(b)} \leftarrow \{\mathbf{y}_i \in \mathbf{Y}\}_{i=1}^B$
    $\nabla_\theta \leftarrow \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}^{(b)}|\boldsymbol{\theta})$ (Eq. (16))
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda_s \nabla_\theta$
  end for
end for
Return sampled cluster parameters $\{\boldsymbol{\mu}_k^{(g)}, \lambda_k^{(g)}\}_{k=1}^K$, and the cluster assignment vector $\mathbf{c}^{(g)}$

where $p(\mathbf{z}_i|\boldsymbol{\mu}_{c_i}^{(g)}, \lambda_{c_i}^{(g)})$ is given in Eq. (1). As we are interested in optimizing the neural network's parameters $\boldsymbol{\theta}$, and the likelihood only involves $\boldsymbol{\theta}$ through $\mathbf{z}_i = f(\mathbf{y}_i; \boldsymbol{\theta})$, the derivative of the log-likelihood is:

$$\begin{aligned}
&\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathbf{H}^{(g)}, \mathbf{c}^{(g)}, \mathbf{Y}|\boldsymbol{\theta}) \\
=&\sum_i \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\mathbf{z}_i|\boldsymbol{\mu}_{c_i}^{(g)}, \lambda_{c_i}^{(g)}) \\
=&\sum_i -\lambda_{c_i}^{(g)}(\mathbf{z}_i - \boldsymbol{\mu}_{c_i}^{(g)}) \frac{\partial f(\mathbf{y}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \qquad (16)
\end{aligned}$$

Since $f(\mathbf{y}_i; \boldsymbol{\theta})$ is implemented by a DNN, its derivative can be automatically computed by many modern machine learning frameworks like PyTorch [Paszke et al., 2019].

3.3.3 Gibbs Sampling

Next we consider how to obtain the samples $\mathbf{H}^{(g)}, \mathbf{c}^{(g)} \sim p(\mathbf{H}, \mathbf{c}|\mathbf{Y}, \boldsymbol{\theta}^{(old)}) = p(\mathbf{H}, \mathbf{c}|\mathbf{Z}^{(old)})$, where we define that $\mathbf{Z}^{(old)} = \{\mathbf{z}_i^{(old)} = f(\mathbf{y}_i; \boldsymbol{\theta}^{(old)})\}_{i=1}^N$. This can be achieved by Gibbs sampling, which states that we can sample from the joint distribution by iteratively sampling from the conditional distribution of each variable while keeping others fixed [Bishop, 2006]. So in what follows we will derive the conditional distribution of each variable.

Conditional distribution of $\boldsymbol{\mu}_k$ and λ_k:

$$\begin{aligned}
&p(\boldsymbol{\mu}_k, \lambda_k|\mathbf{H} \setminus \{\boldsymbol{\mu}_k, \lambda_k\}, \mathbf{c}, \mathbf{Z}^{(old)}) \\
=&p(\boldsymbol{\mu}_k, \lambda_k|\mathbf{c}, \mathbf{Z}^{(old)}) \\
=&p(\boldsymbol{\mu}_k, \lambda_k|[\mathbf{Z}^{(old)}]_k) \\
=&NG(\boldsymbol{\mu}_k, \lambda_k|\boldsymbol{\mu}_n, \kappa_n, \alpha_n, \beta_n), \qquad (17)
\end{aligned}$$

where $[\mathbf{Z}^{(old)}]_k = \{\mathbf{z}_i \in \mathbf{Z}^{(old)}|c_i = k\}$ denotes all the latent variables which are assigned to the k-th cluster. So the result is a normal-gamma distribution, with parameters given by:

$$\begin{aligned}
&n_k = \#[\mathbf{Z}^{(old)}]_k, \quad \bar{\boldsymbol{z}}_{\boldsymbol{k}} = \frac{1}{n_k} \sum_{\mathbf{z}_i \in [\mathbf{Z}^{(old)}]_k} \mathbf{z}_i, \\
&\boldsymbol{\mu}_n = \frac{\kappa_0 \boldsymbol{\mu}_0 + n_k \bar{\boldsymbol{z}}_{\boldsymbol{k}}}{\kappa_0 + n_k}, \quad \kappa_n = \kappa_0 + n_k, \\
&\alpha_n = \alpha_0 + \frac{n_k d}{2}, \\
&\beta_n = \beta_0 + \frac{1}{2} \sum_{\mathbf{z}_i \in [\mathbf{Z}^{(old)}]_k} ||\mathbf{z}_i - \bar{\boldsymbol{z}}_{\boldsymbol{k}}||_2^2 + \frac{\kappa_0 n_k ||\bar{\boldsymbol{z}}_{\boldsymbol{k}} - \boldsymbol{\mu}_0||_2^2}{2(\kappa_0 + n_k)}.
\end{aligned}$$

Conditional distribution of c_i:

• If $n_{-i,k} > 0$ (assign to an existing cluster):

$$\begin{aligned}
&\log p(c_i = k|\mathbf{c}_{-i}, \mathbf{Z}^{(old)}, \mathbf{H}) \\
=&\log p\left(c_i = k \mid \mathbf{c}_{-i}, \alpha\right) + \log p(\mathbf{z}_i|\boldsymbol{\mu}_k, \lambda_k) + \text{const} \\
=&\log \frac{n_{-i,k}}{N - 1 + \alpha} + \log \mathcal{N}(\mathbf{z}_i|\boldsymbol{\mu}_k, \lambda_k^{-1}\mathbf{I}) + \text{const} \qquad (18)
\end{aligned}$$

• If $n_{-i,k} = 0$ (assign to a new cluster):

$$\begin{aligned}
&\log p(c_i = k|\mathbf{c}_{-i}, \mathbf{Z}^{(old)}, \mathbf{H}) \\
=&\log p(\mathbf{z}_i|\boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0) p\left(c_i = k \mid \mathbf{c}_{-i}, \alpha\right) + \text{const} \\
=&\log \int p(\mathbf{z}_i|\boldsymbol{\mu}, \lambda) NG(\boldsymbol{\mu}, \lambda|\boldsymbol{\mu}_0, \kappa_0, \alpha_0, \beta_0) d\boldsymbol{\mu} d\lambda + \\
&\log p\left(c_i = k \mid \mathbf{c}_{-i}, \alpha\right) + \text{const} \\
=&\log \Gamma(\alpha_n') - \log \Gamma(\alpha_0) + \alpha_0 \log \beta_0 - \alpha_n' \log \beta_n' + \\
&\frac{1}{2}(\log \kappa_0 - \log \kappa_n') - \frac{nd}{2} \log 2\pi + \\
&\log \frac{\alpha}{N - 1 + \alpha} + \text{const}, \qquad (19)
\end{aligned}$$

where

$$\begin{aligned}
\kappa_n' &= \kappa_0 + 1, \\
\alpha_n' &= \alpha_0 + d/2, \\
\beta_n' &= \beta_0 + \frac{\kappa_0 ||\mathbf{z}_i - \boldsymbol{\mu}_0||_2^2}{2(\kappa_0 + 1)}.
\end{aligned}$$

Since the normal-gamma distribution is the conjugate prior of the likelihood, the integration in the third line of Eq. (19) is analytically tractable [Murphy, 2007].

Table 1: Hyperparameters of the prior distribution.

Dataset	α	μ_0	κ_0	α_0	β_0
MNIST	1.0E-03	0.0	0.005	2000	1000
HHAR	1.0E-10	0.0	0.005	6000	1000
STL-10	1.0E-10	0.0	0.005	10000	1000
REU-10K	1.0E-10	0.0	0.005	6000	1000

4 EXPERIMENTS

4.1 SYNTHETIC DATASETS

We begin by demonstrating DDPM's potential on two synthetic datasets, and the results are shown in Fig. 1.

Non-isotopic Gaussian dataset: In the first example we generate three clusters. The data points in each cluster are sampled from a non-isotopic Gaussian distribution with different covariance matrices. As the training continues, the data points gradually "concentrate" and approximate the standard Gaussian distribution, which is a more convenient representation for clustering.

Intertwined moon dataset: In this example the initial raw data consists of two intertwined clusters of moon shapes. Note that such representation is challenging for many centroid-based clustering methods, including k-means and DPM. Interestingly, during the training of DDPM, the clusters in the latent representation space are automatically disentangled and finally be successfully identified. This example shows that DDPM can learn better representation during the training, and also articulates the mutually beneficial relationship between clustering and feature learning.

4.2 REAL-WORLD DATASETS

4.2.1 Datasets and settings

Datasets: We evaluate our method on 4 widely used real-world datasets including MNIST [LeCun et al., 1998], HHAR [Stisen et al., 2015], STL-10 [Coates et al., 2011] and REU-10K [Lewis et al., 2004]. MNIST is a handwritten digit database, containing 10 classes of 786-dimensional training samples with 7000 samples for each class. HHAR is a sensor signal classification dataset, containing 10 classes of 561-dimensional training samples and 10200 samples in total. STL-10 is an image recognition dataset, which contains unlabeled data for unsupervised learning. It contains 10 classes of 2048-dimensional training samples and 1300 samples for each class. REU-10K is a text classification dataset consisting of the TF-IDF features of the word. It has 4 classes of 2000-dimensional training instances, and 10000 samples in total.

Model structure and settings: We train the autoencoder by following the prior work of [Jiang et al., 2017]. The network structure is d-500-500-2000-10 for encoder and 10-2000-500-500-d for decoder, where d denotes the dimension of preprocessed input samples. The encoded features are normalized to have 0 mean and 1 standard deviation. The NICE model has 6 layers, each containing 512 units. The training starts with running the standard DPM model for 3 epochs, and each epoch sweeps through the whole dataset for 3 times. After that the main DDPM algorithm runs for 5 epochs. In each epoch the loop of Gibbs sampling sweeps the dataset 3 times, and the NICE model is trained for $0.2N$ iterations (N is the size of the dataset). The batch size is 128 and the learning rate is set to be 1.0E-6. The hyperparameters of the prior are listed in Table 1.

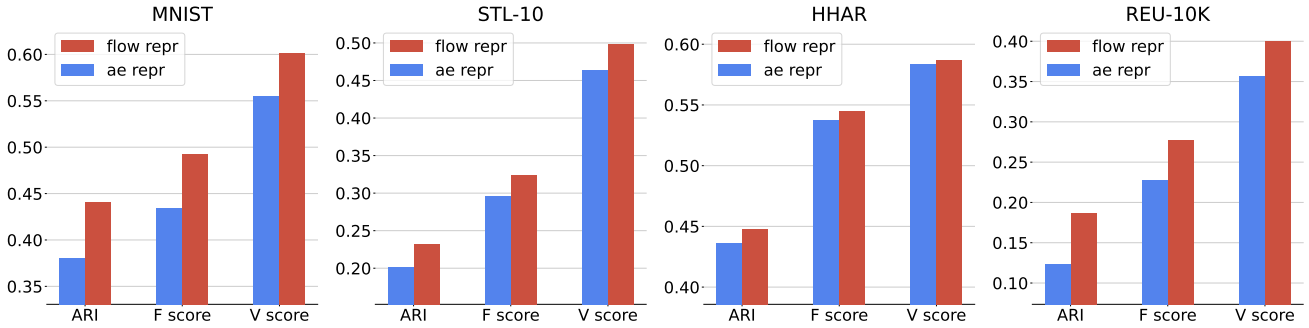
Table 2: Performance comparison on real-world datasets.

Dataset	Methods	ARI	F score	V score
MNIST	G-means	0.1126	0.1255	0.5314
	DPM	0.3974	0.4511	0.5571
	DDPM	**0.4400**	**0.4917**	**0.6016**
HHAR	G-means	0.0904	0.1146	0.4358
	DPM	0.4342	0.5385	0.5761
	DDPM	**0.4473**	**0.5449**	**0.5865**
STL-10	G-means	0.2140	0.2512	0.4830
	DPM	0.2156	0.3073	0.4679
	DDPM	**0.2269**	**0.3193**	**0.4917**
REU-10K	G-means	0.0581	0.0933	0.3147
	DPM	0.1406	0.2365	0.3662
	DDPM	**0.1827**	**0.2756**	**0.3918**

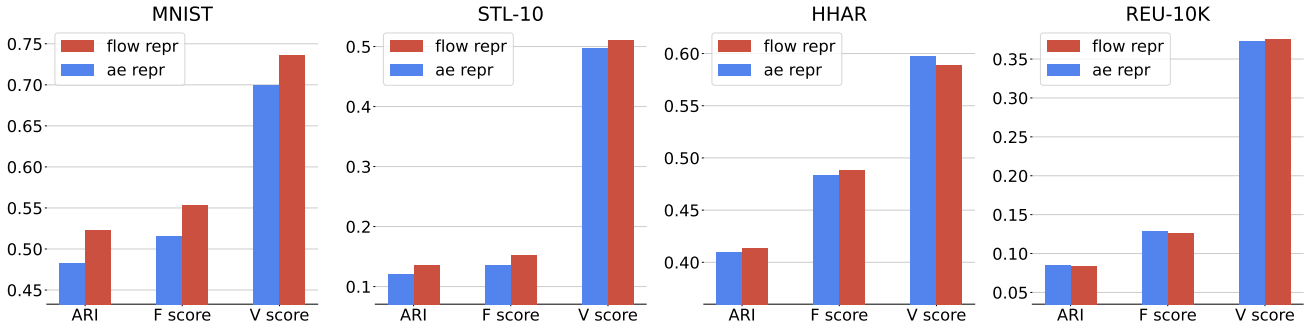
4.2.2 Numerical Results

In our study, we assume the number of clusters K is unknown, so we focus on three K-agnostic evaluation metrics for performance comparison: *adjust random index (ARI)* [Steinley, 2004], *clustering F1 score (F score)*, and *V-measure score (V score)* [Rosenberg and Hirschberg, 2007]. For all these metrics larger means better, and their definitions are presented as follows.

- **ARI** is an adjusted version of the Rand Index (RI). Suppose that C is the ground truth class assignment and K is the predictive clustering. We define a as the number of pairs of elements that are in the same set in C and in the same set in K, and define b as the number of pairs of elements that are in different sets in C and in different sets in K. RI is then given by $\mathrm{RI} = \frac{a+b}{C_N^2}$, where C_N^2 is the total number of possible data pairs. Finally, ARI is defined as $\mathrm{ARI} = \frac{\mathrm{RI} - E[\mathrm{RI}]}{\max(\mathrm{RI}) - E[\mathrm{RI}]}$.
- **F score** for clustering evaluation is just a traditional F1 score calculated based on a pair confusion matrix.



(a) The comparison of DDPM and DPM. The benefit of better feature learning is significant.



(b) The comparison of applying k-means to DDPM's learned feature and the raw feature.

Figure 2: The performance of clustering using the raw autoencoder features (ae repr) and DDPM's learned features (flow repr). (a) DDPM significantly outperforms DPM by learning better representation. (b) By using the learned features in the standard k-means, all metrics in almost all the datasets are improved, and the improvement in the MNIST dataset is particularly significant. This demonstrates DDPM's ability to learn better and transferable representation.

Similar to ARI, The pair confusion matrix (Hubert and Arabie [1985]) computes a 2 by 2 similarity matrix between two clusters by considering all pairs of samples and counting pairs that are assigned into the same or into different clusters under the ground truth cluster assignment.

- **V score** is defined based on the homogeneity term $h = 1 - \frac{H(C|K)}{H(C)}$ and the completeness term $c = 1 - \frac{H(K|C)}{H(K)}$. Here $H(C)$ is the entropy of the classes and $H(C \mid K)$ is the conditional entropy of the classes, defined as

$$H(C) = -\sum_{c=1}^{|C|} \frac{n_c}{N} \cdot \log\left(\frac{n_c}{N}\right),$$

$$H(C \mid K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|} \frac{n_{c,k}}{N} \cdot \log\left(\frac{n_{c,k}}{n_k}\right)$$

Homogeneity encourages each cluster contains only members of a single class, and completeness prefers all members of a given class to be assigned to the same cluster. The V score is finally defined as $v = 2 \cdot \frac{h \cdot c}{h + c}$.

We compare with two other clustering baselines that can infer the number of clusters K, *i.e.*, the standard Dirichlet Process Mixture (DPM) Model and G-means [Zhao et al., 2008]. The numerical results are presented in Table 2, where the best results are highlighted with bold font. We can see that DDPM consistently outperforms other baselines across various datasets and metrics.

4.2.3 Representation quality

To show that the learned representation of DDPM is better than DPM and even transferable to other algorithms, we examine and compare the features before and after processing by the model in all datasets. We additionally apply the k-means clustering algorithms in the feature space (*i.e.*, $\{\mathbf{y}_i = h_e(\mathbf{x}_i) \in \mathbb{R}^d\}_{i=1}^N$) and the transformed space (*i.e.*, $\{\mathbf{z}_i = f(\mathbf{y}_i; \boldsymbol{\theta}) \in \mathbb{R}^d\}_{i=1}^N$), with the number of clusters K specified by DDPM. The results are presented in Fig. 2. It is obvious that the DPM performs better in all cases when the features learned from the DDPM are used. Moreover, with the same prior of K, the k-means can also benefit from the learned representation (e.g. particularly evident in the MNIST dataset), indicating the transferability of the enhanced features. This means that the feature learned from DDPM is not only suitable for DPM but also for other algorithms. We also visualize the learning process of DDPM in Fig. 3. We randomly select 5,000 samples from the MNIST datasets and visualize their t-SNE embedding over different epochs. We can see the clusters become denser and more concentrated as the training progresses, which is potentially

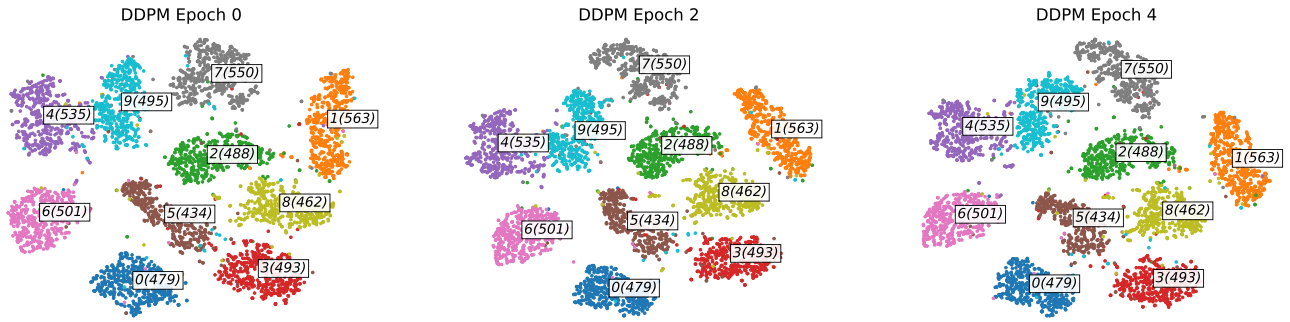


Figure 3: The representation learning process of DDPM on MNIST. The clearest example is the cluster of number 1, which becomes denser and more concentrated as the training process progresses. Its shape gradually changes from a crescent to a circle, making it easier to distinguish from other clusters.

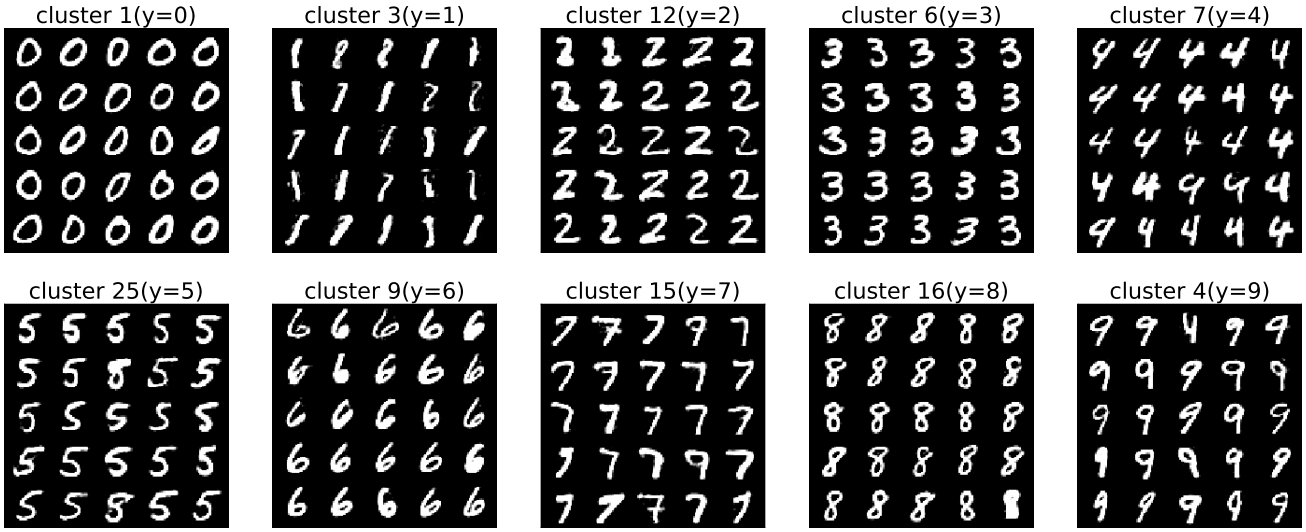


Figure 4: The generated handwritten digits in the MNIST dataset.

beneficial for clustering and label discrimination.

4.2.4 DDPM as a generative model

Benefiting from the reversibility of the flow model, DDPM can be utilized as a generative model. We select the largest cluster for each ground truth label, and generate 25 random samples by adding scaled noise to the cluster centers. Specifically for each selected cluster k, we obtain $\hat{\boldsymbol{\mu}} = \boldsymbol{\mu}_k + n\epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$ is a Gaussian noise and n is the noise scale. Since the flow model is invertible, we can obtain the sample as $\hat{\mathbf{x}} = h_d(f^{-1}(\hat{\boldsymbol{\mu}}; \boldsymbol{\theta}))$. The visualization results are presented in Fig. 4. It can be observed that DDPM is capable of generating clear handwritten digits with sharp edges.

4.2.5 Impact of the autoencoder

In this subsection we study the impact of the autoencoder's quality to the performance of our clustering method. We trained the autoencoder for a different number of iterations and repeated our experiments. As we previously discussed, DPM performs clustering directly on the features extracted by the autoencoder, while DDPM further applies the flow model. The results are presented below (the fully trained autoencoder is optimized for 100000 iterations). We can see that the performance of the clustering methods generally improves as the autoencoder is better trained. Nonetheless, for all the #iter. settings, DDPM consistently outperforms DPM, and the improvement is robust. So our method is effective even if the autoencoder is not sufficiently trained or has low quality.

5 CONCLUSION

In this paper we proposed the deep Dirichlet process mixture (DDPM) model, which jointly achieves clustering and feature learning in an unsupervised fashion. Our method combines the strengths of the traditional DPM models and deep neural networks. Based on the Dirichlet process, DDPM inherits the ability to adapt the number of mixture compo-

Table 3: Impact of the autoencoder trained with different number of iterations.

#iter.	ARI			F SCORE			V SCORE		
	DPM	DDPM	↑	DPM	DDPM	↑	DPM	DDPM	↑
250000	0.2352	0.2618	11%	0.3309	0.3549	7%	0.3754	0.3928	5%
500000	0.3001	0.3316	10%	0.3844	0.4071	6%	0.4306	0.4481	4%
750000	0.2584	0.2815	9%	0.3567	0.3790	6%	0.4117	0.4488	9%
1000000	0.3974	0.4400	11%	0.4511	0.4917	9%	0.5571	0.6016	8%

nents, which is an important and useful feature in many real-world scenarios where prior knowledge of the clusters is unavailable. The invertible flow-based deep neural network component further enables DDPM to learn complex and nonlinear features. Experimental results suggested that DDPM can learn more expressive representation, and achieve better clustering performance compared to other baselines. As for future work, we would like to improve the training efficiency by employing other inference techniques, such as the variational inference framework [Blei and Jordan, 2006]. It is also interesting to extend our method to more sophisticated DPM models like the hierarchical Dirichlet processes [Teh et al., 2006].

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China under Grant 62171248, and the PCNL KEY project (PCL2021A07). It's also supported in part by Shenzhen Science and Technology Innovation Commission (Research Center for Computer Network (Shenzhen) Ministry of Education).

References

Charles E Antoniak. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The annals of statistics*, pages 1152–1174, 1974.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

David M Blei and Michael I Jordan. Variational inference for Dirichlet process mixtures. *Bayesian analysis*, 1(1): 121–143, 2006.

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

Gang Chen, Haiying Zhang, and Caiming Xiong. Maximum margin Dirichlet process mixtures for clustering. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International conference on artificial intelligence and statistics*, pages 215–223, 2011.

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

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: non-linear independent components estimation. In *Workshop of International Conference on Learning Representations (ICLR workshop)*, 2015.

Amine Echraibi, Joachim Flocon-Cholet, Stéphane Gosselin, and Sandrine Vaton. On the variational posterior of dirichlet process deep latent gaussian mixture models. In *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2020.

M Ehsan Abbasnejad, Anthony Dick, and Anton van den Hengel. Infinite variational autoencoder for semi-supervised learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5888–5897, 2017.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, volume 96, pages 226–231, 1996.

Thomas S Ferguson. A Bayesian analysis of some nonparametric problems. *The annals of statistics*, pages 209–230, 1973.

Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

Dilan Görür and Carl Edward Rasmussen. Dirichlet process Gaussian mixture models: Choice of the base distribution. *Journal of Computer Science and Technology*, 25(4):653–664, 2010.

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

Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

Diederik P Kingma and Prafulla Dhariwal. Glow: generative flow with invertible 1×1 convolutions. In *International Conference on Neural Information Processing Systems (NeurIPS)*, pages 10236–10245, 2018.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems (NeurIPS)*, 25:1097–1105, 2012.

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.

David D Lewis, Yiming Yang, Tony Russell-Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

Xiaopeng Li, Zhourong Chen, Leonard KM Poon, and Nevin L Zhang. Learning latent superstructures in variational autoencoders for deep multidimensional clustering. In *International Conference on Learning Representations (ICLR)*, 2018.

Yiming Li, Yang Zhang, Qingtao Tang, Weipeng Huang, Yong Jiang, and Shu-Tao Xia. t-k-means: A robust and stable k-means variant. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3120–3124. IEEE, 2021.

Yuelin Li, Elizabeth Schofield, and Mithat Gönen. A tutorial on Dirichlet process mixture modeling. *Journal of mathematical psychology*, 91:128–144, 2019.

James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, pages 281–297, 1967.

Kevin P Murphy. Conjugate Bayesian analysis of the Gaussian distribution, 2007.

Eric T. Nalisnick and Padhraic Smyth. Stick-breaking variational autoencoders. In *International Conference on Learning Representations (ICLR)*, 2017.

Radford M Neal. Markov chain sampling methods for Dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8024–8035, 2019.

Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Joint conference on empirical methods in natural language processing and computational natural language learning*, pages 410–420, 2007.

Douglas Steinley. Properties of the Hubert-Arable adjusted rand index. *Psychological methods*, 9(3):386, 2004.

Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kjærgaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. In *ACM conference on embedded networked sensor systems*, pages 127–140, 2015.

Gabor J Szekely, Maria L Rizzo, et al. Hierarchical clustering via joint between-within distances: Extending Ward's minimum variance method. *Journal of classification*, 22 (2):151–184, 2005.

Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Hierarchical Dirichlet processes. *Journal of the american statistical association*, 101(476):1566–1581, 2006.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

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

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

Zhonghua Zhao, Shanqing Guo, Qiuliang Xu, and Tao Ban. G-means: a clustering algorithm for intrusion detection. In *International Conference on Neural Information Processing (NeurIPS)*, pages 563–570, 2008.