题目：

Deep Clustering With an Unknown Number of Clusters

摘要

深度学习 (DL)在无监督聚类任务中显示出了巨大的前景。也就是说，虽然在经典（即非深度）聚类中，非参数方法的好处是众所周知的，但大多数深度聚类方法都是参数化的：即，它们需要预定义且固定数量的聚类，用K表示。然而，未知的是，使用模型选择标准来选择其最佳值可能会在计算上变得昂贵，特别是在深度学习中，因为训练过程必须重复多次。在这项工作中，我们通过引入一种有效的深度聚类方法来弥补这一差距，该方法不需要知道K的值，因为它在学习过程中推断出它。使用拆分/合并框架、适应不断变化的K的动态架构和新颖的损失，我们提出的方法优于现有的非参数方法.

1 引言

聚类是一项重要的无监督学习任务，与监督分类情况不同，类别标签不可用。此外，在这项工作关注的纯粹无监督的环境中，类的数量及其相对大小也是未知的。深度学习的出现并没有跳过聚类任务。深度学习方法通常比经典聚类方法更好、更高效地对大型高维数据集进行聚类。也就是说，虽然在经典聚类中，众所周知，非参数方法（即找到K的方法）比参数方法具有优势。只有少数非参数深度聚类方法。不幸的是，后者既不够可扩展，也不够有效。我们的工作通过提出一种有效的深度非参数方法来弥补这一差距。事实上，即使 K 已知，仍然能取得与领先的参数方法相当的结果（特别是在不平衡的情况下），尽管它们“不公平”。更一般地说，推断潜在 K的能力具有实际好处，包括以下好处。(1)如果没有很好地估计K，参数方法的性能可能会受到影响。 (2)如果没有很好地估计K，参数方法的性能可能会受到影响。(3)不知道K的常见解决方法是使用模型选择：即多次运行参数方法，在较大范围内使用不同的K值，然后通过无监督标准选择“最佳”K。然而，这种方法除了缺少上述潜在收益（无法进行大的移动）之外，还无法扩展，并且对于大型数据集（尤其是在深度学习中）通常不可行。此外，还必须注意模型选择方法的负面社会影响：在大型数据集上训练深度网络数十或数百次会消耗大量能源。(4)K本身可能是一个受欢迎的重要量。

2 相关工作

2.1 参数深度聚类方法

最近的此类工作可以分为两种类型：两步方法和端到端方法。在前者中，对任务中提取的特征进行聚类。例如，麦康维尔等人[1]. 。对预训练自动编码器的UMAP转换的嵌入运行K均值。虽然不可扩展，但在适用时取得了有竞争力的结果。另一个例子是SCAN ，它使用无监督的预训练特征提取器。端到端深度方法可能通过交替联合学习特征和聚类。一些作品使用了AE或变分AE，并带有额外的聚类损失 ；例如，DCN在预训练AE的嵌入上运行K均值，并使用由重建项和基于聚类的项组成的损失对其进行重新训练，以同时更新特征、聚类中心和分配。

2.2 非参数经典聚类

部分原因是缺乏高效的大规模推理工具。幸运的是，这种情况正在开始改变。例如，参中的高效DPM采样器或 可扩展流式DPM推理。值得注意的是，采样的一个重要替代方法是变分DPM推断 。流行的非参数方法的非贝叶斯示例是DBSCAN ，它是基于密度的并将紧密堆积的点组合在一起。虽然DBSCAN具有高效的实现，但它对其超参数高度敏感且难以调整。

2.3 非参数深度聚类

其中一些使用离线DPM推断伪标签来微调深度置信网络或AE。AdapVAE使用DPM先验作为VAE。在DCC中，特征学习和聚类像一样同时进行；然而，DCC使用最近邻图来对 AE 潜在空间中接近的点进行分组，而不是ELBO最小化。

3 本文方法

3.1 本文方法概述

DeepDPM可以看作是一种DPM推理算法。我们使用拆分和合并来更改K，其中对于每个集群我们维护一个子集群对。对于K的值，在混合模型中使用由新颖的EM摊销推理训练的深度网络。DeepDPM有两个主要部分。第一个是聚类网络，而第二个由K个子聚类网络组成（每个聚类k个，k ∈ {1,...,K}）如图1所示：

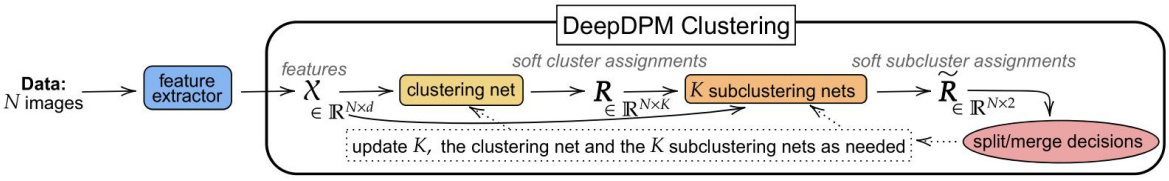

图 1. DeepDPM网络结构图

3.2 给定k下的DeepDPM

DeepDPM在训练过程中的前向传播。给定K的当前值，数据首先传递到聚类网络f_{c_1}，该网络为每个数据点x_i生成K个软聚类分配：

$$f_{\mathrm{cl}}(\mathcal{X}) = \boldsymbol{R} = (\boldsymbol{r}_i)_{i=1}^{N} \qquad \boldsymbol{r}_i = (r_{i,k})_{k=1}^{K}$$

其中 $r_{i,k} \in [0,1]$ 是 x_i 到集群k的软分配（也称为集群k对数据点x_i的责任）并且$r_{i,k} = 1$。从 $(r_i)_{i=1}^{N}$我们通过 $z_i = \mathrm{argmax}\ r_{i,k}$计算硬分配 $z = (z_i)_{i=1}^{N}$。接下来，每个子聚类网络 f_{sub}^{K}（其中 k ∈ {1, . . . ,K}）被馈送分配给其各自簇的数据（即， f_{sub}^{k}被馈送 $X_k = (x_i)_{i:z_{i=k}}$并生成软子簇分配：

$$f_{\mathrm{sub}}^{k}(\mathcal{X}_k) = \widetilde{\boldsymbol{R}}_k = (\widetilde{\boldsymbol{r}}_i)_{i:z_i=k} \qquad \widetilde{\boldsymbol{r}}_i = (\widetilde{r}_{i,j})_{j=1}^{2}$$

3. 3分离合并策略

分离：为了适应K的增加，如果聚类k接受分割提案，则聚类网络最后一层中的第k个单元以及将其连接到前一个隐藏层的权重将被复制，并且我们初始化使用通过子簇网络学习到的参数来计算两个新簇的参数：

$$\boldsymbol{\mu}_{k_1} \leftarrow \widetilde{\boldsymbol{\mu}}_{k,1}, \quad \boldsymbol{\Sigma}_{k_1} \leftarrow \widetilde{\boldsymbol{\Sigma}}_{k,1}, \quad \pi_{k_1} \leftarrow \pi_k \times \widetilde{\boldsymbol{\pi}}_{k,1}$$
$$\boldsymbol{\mu}_{k_2} \leftarrow \widetilde{\boldsymbol{\mu}}_{k,2}, \quad \boldsymbol{\Sigma}_{k_2} \leftarrow \widetilde{\boldsymbol{\Sigma}}_{k,2}, \quad \pi_{k_2} \leftarrow \pi_k \times \widetilde{\boldsymbol{\pi}}_{k,2}$$

其中k_1和k_2表示新簇的索引。然后，我们还向每个新集群添加一个新的子集群网络（动态分配内存）。

合并：考虑每个簇仅与其3个最近邻居的合并。使用黑斯廷斯比率 $H_m = 1/H_s$ 接受/拒绝合并提案。如果提案被接受，两个集群将被合并，并初始化一个新的子集群网络。从技术上讲，合并簇的最后一层单元之一以及将其连接到前一个隐藏层的网络权重都从 f_{c_1} 中删除，并且使用加权MAP估计来初始化新簇的参数和权重。

4 复现细节

4.1 与已有开源代码对比

本文所使用的代码已经开源，在本次实验中，主要使用了论文中的参数设置，如图2所示，为一部分参数代码截图：

```
parser.add_argument(
    "--lr", type=float, default=0.002, help="learning rate (default: 1e-4)"
)
parser.add_argument(
    "--batch-size", type=int, default=128, help="input batch size for training"
)
parser.add_argument(
    "--seed",
    type=int,
    default=None,
    help="random seed",
)
parser.add_argument(
    "--n-jobs", type=int, default=2, help="number of jobs to run in parallel"
)
```

图2. 参数代码部分截图

使用的聚类网络结构代码如图3所示：

```
    )
if self.current_epoch == 0 and (self.hparams.log_emb in ("every_n_epochs", "only_sampled") and
    perm = torch.randperm(self.train_gt.size(0))
    idx = perm[:10000]
    sampled_points = self.codes[idx]
    sampled_labeled = self.train_gt[idx] if self.hparams.use_labels_for_eval else None
    self.plot_utils.visualize_embeddings(
        self.hparams,
        self.logger,
        self.codes_dim,
        vae_means=sampled_points,
        vae_labels=sampled_labeled,
        val_resp=None,
        current_epoch=self.current_epoch,
        y_hat=None,
        centers=None,
        training_stage='train_sampled',
        UMAP=False
    )
```

图3. 聚类网络部分截图

4.2 实验环境搭建

实验硬件环境：4060显卡Windows系统PC电脑一台
实验软件环境：PyCharm软件

4.3 界面分析与使用说明

本实验在Pycharm平台下进行，使用界面如图4所示：

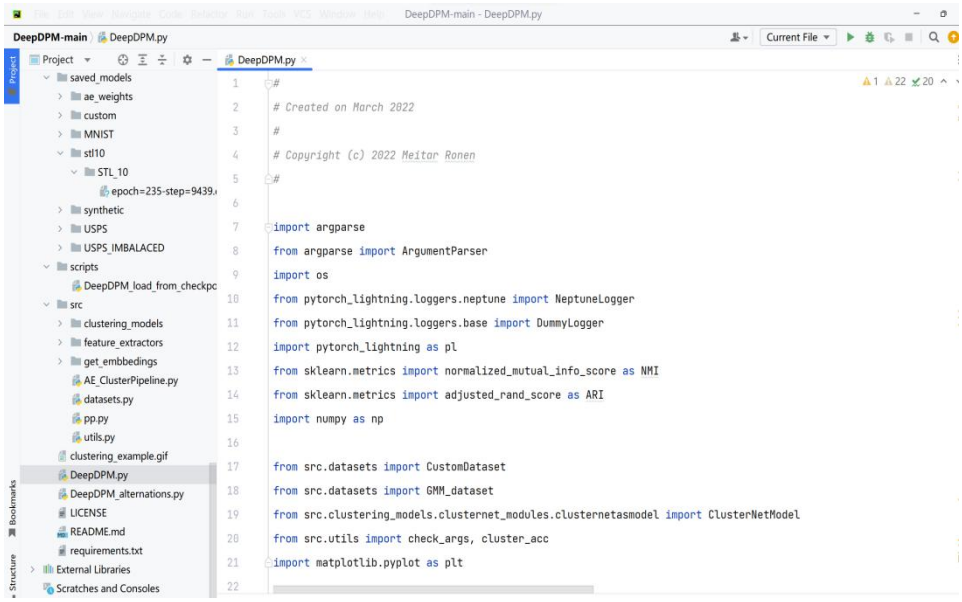

图 4. 操作界面示意

4.4 创新点

本文尝试改进网络模型结构，通过增加线性层结构，改变其维度来优化网络。

5 实验结果分析

通过初始化聚类数目，并设计一个拥有15个高斯分布，并对其进行采样10000个样本点，如图3显示其聚类结果，可以看到高维高斯分布在平面显示为椭圆形状，其中黑色点表示聚类中心，颜色相同的为同一个类别。

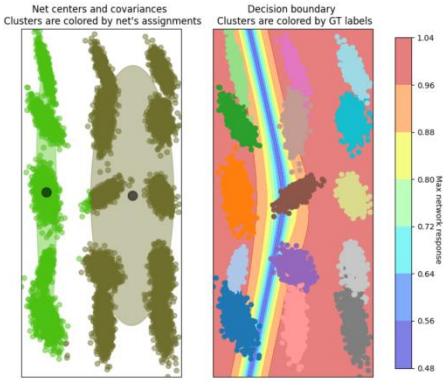

图3：合成数据聚类效果示意图

通过在MNIST、FASHION-MNIST、USPS数据集下，运行300epoch，实验结果如图表1所示：

数据集	epochs	NMI	ARI	ACC	final: K
MNIST	300	0.93443	0.9249	0.9677	11
FASHION-MNIST	300	0.93878	0.96374	0.96101	8
USPS	300	0.79728	0.66086	0.66046	6

表1. 实验数据比较图

6 总结与展望

本文提出了一种深度非参数聚类方法、一种适应不同K值的动态架构，以及一种基于混合模型中新摊销推理的新颖损失。该方法优于深度和非深度非参数方法并实现了SOTA结果。与大多数聚类方法一样，如果DeepDPM的输入特征很差，它将很难恢复。此外，如果K已知并且数据集是平衡的，参数方法可能是更好的选择。未来的工作。使DeepDPM适应流数据或分层设置。此外，如果有一个更复杂的框架来构建分割提案，结果可能会有所改善。

参考文献

[1]. Ryan McConville, Raul Santos-Rodriguez, Robert J Piechocki, and Ian Craddock. N2d:(not too) deep clustering ia clustering the local manifold of an autoencoded embedding. In ICPR, 2020. 2, 3, 6