利用共享表征实现个性化联邦学习

摘要

深度神经网络已经充分体现了从图像和文本等数据中提取通用特征表示的能力，这些特征表示对各种学习任务都非常有用。然而表征学习的成果尚未在联邦环境中完全实现。尽管联邦设置中的数据通常是非独立同分布的。集中式深度学习的成功表明，数据通常共享全局特征表示，而客户端或任务之间的统计异质性集中在标签中。基于这种直觉，我们提出了一种新的联邦学习框架和算法，用于学习跨客户端的共享数据表示和每个客户端的唯一本地头。我们的算法利用客户端的分布式计算能力来执行许多本地更新的表示的每一个更新的低维局部参数。我们证明了该方法在线性设置中以接近最优的样本复杂度线性收敛到地面真实表示，表明它可以有效地降低每个客户端的问题维度。此外，我们提供了大量的实验结果，证明了我们的方法在异构环境中的替代个性化联邦学习方法的改进。

关键词：联邦学习；个性化

1 引言

集中式学习在现代机器学习领域取得了巨大成功，模型是在大量集中存储的数据上训练。然而越来越多的应用场景需要保护数据的隐私性。联合学习旨旨在通过不共享数据的前提下，让多个参与方共同训练一个共享的机器学习模型。与传统的集中式学习不同，联邦学习将数据保留在本地设备或边缘服务器上，仅在本地进行模型训练，然后通过聚合算法将各方的模型更新进行合并，形成一个全局的模型。各个客户端通过利用所有客户端的本地计算能力、存储器和数据来协作以学习针对每个客户端的有效模型。

然而，联邦学习中最重要的挑战之一是数据异构性问题，客户端的底层数据分布可能彼此存在很大差异。在这样的设置中，如果服务器和客户端学习单个共享模型（例如通过最小化平均损失），所得到的模型对于网络中的许多客户端可能表现不佳事实上，对于某些客户端来说，最好使用他们自己的本地数据（即使很小）来训练本地模型。联邦训练模型可能无法很好地推广到未参与训练过程的未见过的客户端。以上问题引出了这样一个问题："如何利用数据异构环境中所有客户端的数据和计算能力，为每个客户端学习个性化模型?"

本文通过利用客户端之间的共同表征信息来解决这个问题。具体而言，可以将数据异构联邦学习问题看作 n 个并行学习任务，人物之间具有某种公共结构，本文的目标是学习和利用这种公共表征来提高每个客户端模型的质量。事实上这正符合我们对于集中式学习的理解。在集中式学习场景下，通过利用流行机器学习任务中的公共（低维）表示来同时训练多个任务已经取得了成功（例如图像分类、词预测）。

本文提出了一种新的联邦学习框架 FedRep 和相应的的数据异构设置。FedRep 利用跨客户端存储的全部数据量，使用基于梯度的更新来学习全局的低维表征。此外，它使每个客户

端能够计算个性化的低维分类器，这里称之为客户端的本地头，负责学习每个客户端本地数据的独特标签。以下是 FedRep 模型的优点。

1. 本地更新次数增多。通过降低问题的维度，各个客户端可以在每个通信回合进行多次本地更新，这种设置有利于学习自己的模型的本地头部。而在标准的联邦学习中，由于数据异构设置中，多轮次的本地更新会使每个客户端远离最佳平均表示，从而降低模型性能。

2. 客户端之间的合作促进训练。d 表示为数据维度，n 表示客户端的数量。从样本复杂度界限可以得出使用 FedRep 框架，每个客户端的样本复杂度为 $log\, n + d/n$ 。另一方面，本地学习（没有任何协作）的样本复杂度为 d 。因此，如果 $1 \ll n \ll e^d$ ，通过联邦进行协作有助于模型训练。当 d 很大时，e^d 会呈指数级增长，联邦对每个客户端都有帮助。

3. 易于推广到新的客户端。对于新的客户端，由于可以利用现有的现成的全局表征，因此客户端仅需要学习具有维度 k 的低维表征的模型头部。因此当样本复杂度仅为 $klog\ 1/k$ 时，可以达到不超过 1/2 的误差。

FedRep 模型利用跨客户端存储的全部数据以及基于梯度的更新方式学习全局共享表征，各个客户端利用本地独特的标签数据计算个性化的低维分类器，FedRep 模型如图 1所示。

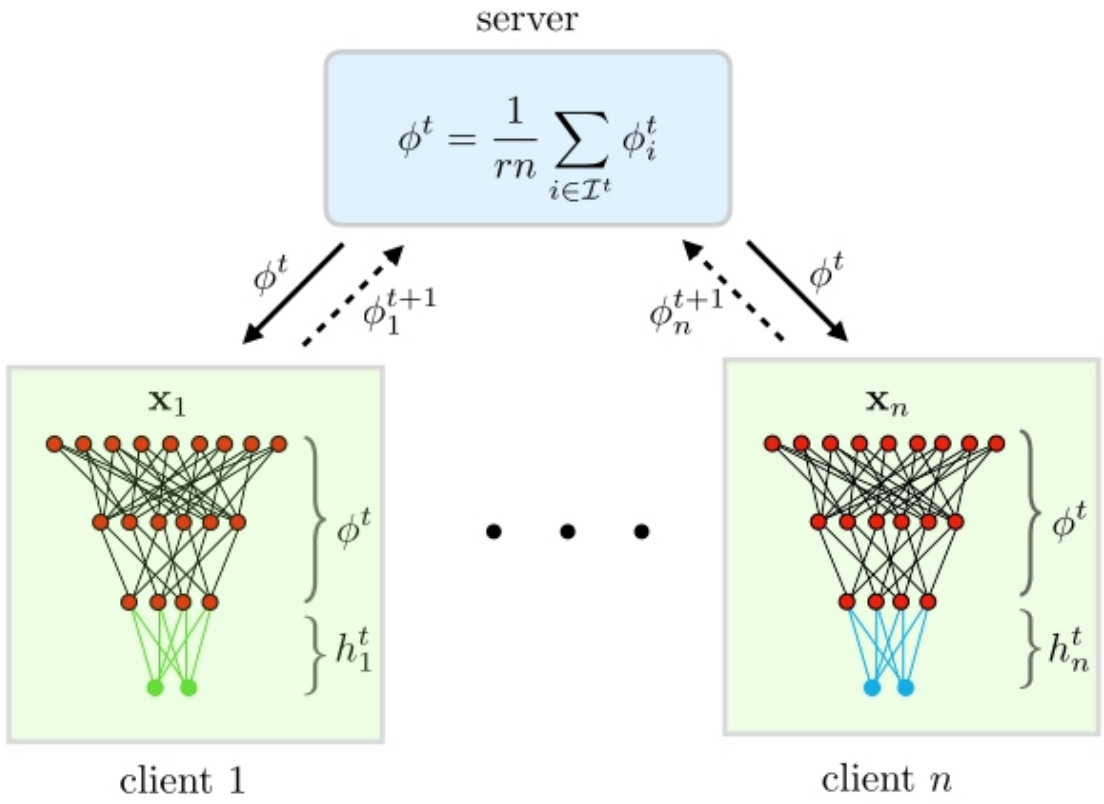

图 1. FedRep 模型结构

2 相关工作

最近的许多工作已经研究了联邦学习中的个性化，例如使用局部微调 [14, 16]、元学习 [4, 10, 11]，局部和全局模型的混合和多任务学习 [5]。在所有这些方法中，每个客户端的子问题仍然是全维的-没有学习降维的局部参数集的概念。Liang 等人 [6] 也提出了一种用于联邦学习的表示学习方法，但他们的方法试图学习许多局部表示和单个全局头部，而不是单个全局表示和许多局部头部。早些时候，Arivazhagan 等人 [8] 提出了一种学习局部头部和全局网络主体的算法，但他们的局部过程联合更新头部和主体（使用相同数量的更新），并且他们没有为他们提出的方法提供任何理论依据。与此同时，另一项工作研究了异构环境中的联邦学习 [1–3, 7, 12, 13, 15]。

3 本文方法

3.1 本文方法概述

本文根据联邦学习中数据异构的特点以及模型在全局数据集上的训练结果由于局部偏差数据集的直接观察，结合 FedRep 方法以及对比学习思想，利用参与的多个客户端共同协作训练一个共享的全局模型，同时针对于本地训练数据集的差异化特点各个客户端训练各自的本地训练头以实现个性化的联邦学习，提高模型的性能。与此同时增大本地训练模型以及全局模型表征信息的一致性进一步加速网络模型的收敛速度，最终得到一个快速收敛，对于不同程度数据异构场景性能稳定，易于推广至新加入的客户端的个性化联邦学习模型，具体模型执行流程如图 2所示，同时在此基础上将模型的总体损失改为由监督损失与对比损失的加权和。

Algorithm 1 FedRep

Parameters: Participation rate r, step sizes α, η; number of local updates τ; number of communication rounds T.
Initialize $\phi^0, h_1^0, \ldots, h_n^0$
for $t = 1, 2, \ldots, T$ **do**
  Server receives a batch of clients $\mathcal{I}^t$ of size rn
  Server sends current representation ϕ^t to these clients
  for each client i in $\mathcal{I}^t$ **do**
    Client i initializes $h_i^t \leftarrow h_i^{t-1,\tau}$
    Client i makes τ updates to its head h_i^t:
    for $s = 1$ **to** τ **do**
      $h_i^{t,s} \leftarrow \mathtt{GRD}(f_i(h_i^{t,s}, \phi^t), h_i^{t,s}, \alpha)$
    end for
    Client i locally updates the representation as:
      $\phi_i^{t+1} \leftarrow \mathtt{GRD}(f_i(h_i^{t,\tau}, \phi^t), \phi^t, \alpha)$
    Client i sends updated representation ϕ_i^{t+1} to server
  end for
  for each client i not in $\mathcal{I}^t$, **do**
    Set $h_i^{t,\tau} \leftarrow h_i^{t-1,\tau}$
  end for
  Server computes the new representation as
    $\phi^{t+1} = \frac{1}{rn} \sum_{i \in \mathcal{I}^t} \phi_i^{t+1}$
end for

图 2. FedRep 算法流程

3.2 问题定义

具有 n 个客户端的联邦学习的一般形式是

$$\min_{(q_1,\ldots,q_n)\in\mathcal{Q}_n} \frac{1}{n}\sum_{i=1}^{n} f_i(q_i),$$

其中 f_i 和 q_i 分别是第 i 个客户端的误差函数和学习模型，Q_n 是 n 个模型的可行集的空间。我们考虑一个有监督的设置，其中第 i 个客户端的数据由分布 $(\mathbf{x}_i, y_i) \sim \mathcal{D}_i$ 生成。联邦学习

的标准方法旨在学习单个共享模型 $q = q_1 = \cdots = q_n$，该模型在客户端之间平均表现良好（McMahan 等人，2017 年; Li 等人，2018 年）。通过这种方式，客户端的目标是在选择共享模型 q 时最小化 $\frac{1}{n} f_i(q)$。然而，这种方法可能产生在异构设置中表现不佳的解决方案，其中数据分布 D_i 在客户端之间变化。事实上，在数据异质性的存在下，误差函数 f_i 将具有不同的形式，并且它们的最小化形式也不相同。因此，学习共享模型 q 可能无法提供良好解决方案。这就需要寻找更个性化的解决方案 q_i，这些解决方案可以使用客户端的数据以联合方式学习。形式上，我们考虑全局表征 q: $R_d \to R_k$ 是由　参数化的函数，将数据点映射到 k 维的较低空间，客户端特定头 $q - h_i$: $R_k \to$ Y 是由 h_i 参数化的函数，i $\in$ [n] 从低维表示空间映射到标签空间。第 i 个客户端的模型是客户端的局部参数和表示的组合: $q_i(\mathbf{x}) = (q_{h_i} \circ q_\phi)(\mathbf{x})$ 假设任何固定表示的任何客户端的最佳分类器都很容易计算，由此得到重写的全局目标函数:

$$\min_{\phi \in \Phi} \frac{1}{n} \sum_{i=1}^{n} \min_{h_i \in \mathcal{H}} f_i(h_i, \phi),$$

3.3 client 端更新

在每一轮通信时，选择一定数量的客户端 $r \in$ 0 1] 来执行客户端更新。在客户端更新中，客户端 i 进行 τ 个基于局部梯度的更新，以在给定由服务器传送的当前全局表示 ^{t} 的情况下求解其最优头部。即对于 $s = 1 \ldots \tau$，客户端 i 如下更新其头部:

$$h_i^{t,s+1} = \mathrm{GRD}(f_i(h_i^{t,s}, \phi^t), h_i^{t,s}, \alpha),$$

其中 GRD（f，h，α）是使用函数 f 关于 h 的梯度和步长　来更新变量 h 的通用符号。例如 $\mathrm{GRD}(f_i(h_i^{t,s}, \phi^t), h_i^{t,s}, \alpha)$ 可以是梯度下降、随机梯度下降（SGD）、具有动量的 SGD 等的步骤。关键是客户端 i 进行许多这样的局部更新，即 τ 值很大，以基于其本地数据找到最佳头部。客户端的损失计算包括对比损失 [9] 以及自监督损失，将两者加权求和更新梯度信息再发送给服务器进行聚合。

3.4 server 端更新

一旦本地模型头部 h_i 的本地更新完成，客户端就通过相对于当前表征进行一次基于梯度的本地更新来参与服务器更新，即计算

$$\phi_i^{t+1} \leftarrow \mathrm{GRD}(f_i(h_i^{t,\tau}, \phi^t), \phi^t, \alpha).$$

然后将 ϕ_i^{t+1} 发送到服务器，服务器对本地更新进行平均以计算下一个表征 ϕ^{t+1}。

3.5 模型分析

对于没有协作的单个客户端，样本复杂度缩放为 d 。采用了 FedRep 框架后样本复杂度缩放为 $log\ n\ + \frac{d}{n}$ 。因此，只要 $log\ n\ + \frac{d}{n} \geq d$，联邦学习将发挥作用。在实际场景中，d（数据维度）很大，因此 $e^{\,d}$ 呈指数增长; 因此协作有助于每个客户端。此外从之后进入系统的新客户端的角度来看，它具有即刻可用的模型表征，新客户端适应其任务的样本复杂度仅为 $klog\frac{1}{n}$。因此，整个系统都受益（已经学习到了全局低维表征，新客户端现在只需要学习模型头部），并且参与联邦训练的每个单独的客户端也将受益。

4 复现细节

模型的复现总体可以分成 5 个部分。首先是模型的辅助代码，主要用于处理数据集，包括数据集的加载，划分以及增加多样性的预处理过程。其次是核心代码部分的客户端的模型更新部分。客户端的本地更新采用了卷积神经网络作为主干特征提取网络，本地执行多个轮次的更新，在每一个更新过程中记录上一轮次的损失值，计算对比损失以最大化本地模型与全局模型的一致性，同时由于 FedRep 利用了全局的共享模型提取低维特征，本地数据集构建个性化的分类器，因此需要执行相对于 FedAvg 方法更多轮次的本地特征以增强个性化特征的表征能力。

模型的中央服务器聚合过程聚合各个参与的客户端发送的全局共享模型（不包含客户端的本地头部分），对接收到的共享模型参数进行加权平均。模型的训练优化部分采用了自适应权重优化，在模型训练的初期阶段将步长设置得大一些以加快训练过程。当训练的梯度信息小时，自适应缩小权重逼近全局最优解。训练优化过程结合了多种方式，包括如扰动梯度下降等方法。

具体训练过程是完成数据预处理并将数据与模型迁移到 GPU 后，中央服务器以及本地客户端分别初始化全局共享模型，本地训练头，中央服务器将全局共享模型参数广播至各个客户端。客户端利用本地数据集进行多个轮次的本地训练以得到个性化的低维分类器，以及更新全局共享模型的参数，计算对比损失以及监督学习损失，更新权重信息并发送给中央服务器。中央服务器聚合更新的权重再与客户端进行通信，重复这个过程直到算法收敛或者达到设定的准确度阈值。

```
if __name__ == "__main__":
    total_start = time.time()

    parser = argparse.ArgumentParser()
    # general
    parser.add_argument('-go', "--goal", type=str, default="test",
                        help="The goal for this experiment")
    parser.add_argument('-dev', "--device", type=str, default="cuda",
                        choices=["cpu", "cuda"])
    parser.add_argument('-did', "--device_id", type=str, default="0")
    parser.add_argument('-data', "--dataset", type=str, default="mnist")
    parser.add_argument('-nb', "--num_classes", type=int, default=10)
    parser.add_argument('-m', "--model", type=str, default="cnn")
    parser.add_argument('-lbs', "--batch_size", type=int, default=10)
    parser.add_argument('-lr', "--local_learning_rate", type=float,
       default=0.005,
                        help="Local learning rate")
    parser.add_argument('-ld', "--learning_rate_decay", type=bool,
       default=False)
    parser.add_argument('-ldg', "--learning_rate_decay_gamma",
       type=float, default=0.99)
```

```
    parser.add_argument('-gr', "--global_rounds", type=int, default=1000)
    parser.add_argument('-ls', "--local_epochs", type=int, default=1,
                        help="Multiple update steps in one local epoch.")
    parser.add_argument('-algo', "--algorithm", type=str,
        default="FedRep")
    parser.add_argument('-jr', "--join_ratio", type=float, default=1.0,
                        help="Ratio of clients per round")
    parser.add_argument('-rjr', "--random_join_ratio", type=bool,
        default=False,
                        help="Random ratio of clients per round")
    parser.add_argument('-nc', "--num_clients", type=int, default=2,
                        help="Total number of clients")
    parser.add_argument('-pv', "--prev", type=int, default=0,
                        help="Previous Running times")
    parser.add_argument('-t', "--times", type=int, default=1,
                        help="Running times")
    parser.add_argument('-eg', "--eval_gap", type=int, default=1,
                        help="Rounds gap for evaluation")
    parser.add_argument('-dp', "--privacy", type=bool, default=False,
                        help="differential privacy")
    parser.add_argument('-dps', "--dp_sigma", type=float, default=0.0)
    parser.add_argument('-sfn', "--save_folder_name", type=str,
        default='items')
    parser.add_argument('-ab', "--auto_break", type=bool, default=False)
    parser.add_argument('-dlg', "--dlg_eval", type=bool, default=False)
    parser.add_argument('-dlgg', "--dlg_gap", type=int, default=100)
    parser.add_argument('-bnpc', "--batch_num_per_client", type=int,
        default=2)
    parser.add_argument('-nnc', "--num_new_clients", type=int, default=0)
    parser.add_argument('-fte', "--fine_tuning_epoch", type=int,
        default=0)
    # practical
    parser.add_argument('-cdr', "--client_drop_rate", type=float,
        default=0.0,
                        help="Rate for clients that train but drop out")
    parser.add_argument('-tsr', "--train_slow_rate", type=float,
        default=0.0,
                        help="The rate for slow clients when training
                            locally")
    parser.add_argument('-ssr', "--send_slow_rate", type=float,
        default=0.0,
                        help="The rate for slow clients when sending
                            global model")
```

```
52      parser.add_argument('-ts', "--time_select", type=bool, default=False,
53                          help="Whether to group and select clients at
                                each round according to time cost")
54      parser.add_argument('-tth', "--time_threthold", type=float,
           default=10000,
55                          help="The threthold for droping slow clients")
56      # pFedMe / PerAvg / FedProx / FedAMP / FedPHP
57      parser.add_argument('-bt', "--beta", type=float, default=0.0,
58                          help="Average moving parameter for pFedMe,
                                Second learning rate of Per-FedAvg, \
59                          or L1 regularization weight of FedTransfer")
60      parser.add_argument('-lam', "--lamda", type=float, default=1.0,
61                          help="Regularization weight")
62      parser.add_argument('-mu', "--mu", type=float, default=0,
63                          help="Proximal rate for FedProx")
64      parser.add_argument('-K', "--K", type=int, default=5,
65                          help="Number of personalized training steps for
                                pFedMe")
66      parser.add_argument('-lrp', "--p_learning_rate", type=float,
           default=0.01,
67                          help="personalized learning rate to caculate
                                theta aproximately using K steps")
68      # FedFomo
69      parser.add_argument('-M', "--M", type=int, default=5,
70                          help="Server only sends M client models to one
                                client at each round")
71      # FedMTL
72      parser.add_argument('-itk', "--itk", type=int, default=4000,
73                          help="The iterations for solving quadratic
                                subproblems")
74      # FedAMP
75      parser.add_argument('-alk', "--alphaK", type=float, default=1.0,
76                          help="lambda/sqrt(GLOABL-ITRATION) according to
                                the paper")
77      parser.add_argument('-sg', "--sigma", type=float, default=1.0)
78      # APFL
79      parser.add_argument('-al', "--alpha", type=float, default=1.0)
80      # Ditto / FedRep
81      parser.add_argument('-pls', "--plocal_steps", type=int, default=1)
82      # MOON
83      parser.add_argument('-tau', "--tau", type=float, default=1.0)
84      # FedBABU
85      parser.add_argument('-fts', "--fine_tuning_steps", type=int,
```

```
        default=10)
    # APPLE
    parser.add_argument('-dlr', "--dr_learning_rate", type=float,
        default=0.0)
    parser.add_argument('-L', "--L", type=float, default=1.0)
    # FedGen
    parser.add_argument('-nd', "--noise_dim", type=int, default=512)
    parser.add_argument('-glr', "--generator_learning_rate", type=float,
        default=0.005)
    parser.add_argument('-hd', "--hidden_dim", type=int, default=512)
    parser.add_argument('-se', "--server_epochs", type=int, default=1000)
    parser.add_argument('-lf', "--localize_feature_extractor",
        type=bool, default=False)
    # SCAFFOLD
    parser.add_argument('-slr', "--server_learning_rate", type=float,
        default=1.0)
    # FedALA
    parser.add_argument('-et', "--eta", type=float, default=1.0)
    parser.add_argument('-s', "--rand_percent", type=int, default=80)
    parser.add_argument('-p', "--layer_idx", type=int, default=2,
                        help="More fine-graind than its original paper.")
    # FedKD
    parser.add_argument('-mlr', "--mentee_learning_rate", type=float,
        default=0.005)
    parser.add_argument('-Ts', "--T_start", type=float, default=0.95)
    parser.add_argument('-Te', "--T_end", type=float, default=0.98)
    # GPFL
    parser.add_argument('-lamr', "--lamda_reg", type=float, default=0.0)


    args = parser.parse_args()

    os.environ["CUDA_VISIBLE_DEVICES"] = args.device_id

    if args.device == "cuda" and not torch.cuda.is_available():
        print("\ncuda is not avaiable.\n")
        args.device = "cpu"
    print("Algorithm: {}".format(args.algorithm))
    print("Local batch size: {}".format(args.batch_size))
    print("Local steps: {}".format(args.local_epochs))
    print("Local learing rate: {}".format(args.local_learning_rate))
    print("Local learing rate decay:
        {}".format(args.learning_rate_decay))
```

```
    if args.learning_rate_decay:
        print("Local learing rate decay gamma:
            {}".format(args.learning_rate_decay_gamma))
    print("Total number of clients: {}".format(args.num_clients))
    print("Clients join in each round: {}".format(args.join_ratio))
    print("Clients randomly join: {}".format(args.random_join_ratio))
    print("Client drop rate: {}".format(args.client_drop_rate))
    print("Client select regarding time: {}".format(args.time_select))
    if args.time_select:
        print("Time threthold: {}".format(args.time_threthold))
    print("Running times: {}".format(args.times))
    print("Dataset: {}".format(args.dataset))
    print("Number of classes: {}".format(args.num_classes))
    print("Backbone: {}".format(args.model))
    print("Using device: {}".format(args.device))
    print("Using DP: {}".format(args.privacy))
    if args.privacy:
        print("Sigma for DP: {}".format(args.dp_sigma))
    print("Auto break: {}".format(args.auto_break))
    if not args.auto_break:
        print("Global rounds: {}".format(args.global_rounds))
    if args.device == "cuda":
        print("Cuda device id:
            {}".format(os.environ["CUDA_VISIBLE_DEVICES"]))
    print("DLG attack: {}".format(args.dlg_eval))
    if args.dlg_eval:
        print("DLG attack round gap: {}".format(args.dlg_gap))
    print("Total number of new clients: {}".format(args.num_new_clients))
    print("Fine tuning epoches on new clients:
       {}".format(args.fine_tuning_epoch))
    print("=" * 50)
```

4.1 与已有开源代码对比

本文代码实现上原始算法的区别主要在于改进了损失函数，将原来的真值与预测值之间的二次损失改为对比损失。对比损失的含义在于最大化本地模型与全局模型的一致性，因此客户端本地更新的方式有所不同。训练过程中，本地客户端需要记录上一轮次的本地模型损失，在每个迭代周期计算现有的模型损失。为了最大化本地模型与全局模型的一致性从而更好的利用全局低维表征获得更好的训练模型，需要利用中央服务器发送的全局模型损失值。对比损失需要计算本地模型与全局模型的差异性并减小这一差异，计算与上一轮次损失的差异并使之增大。最终模型总体损失由对比损失以及自监督学习两部分加权构成。最后利用损失的梯度信息更新模型权重，将新的模型权重发送给中央服务器进行聚合。

```python
import copy
import torch
import torch.nn as nn
import numpy as np
import time
import torch.nn.functional as F
from flcore.clients.clientbase import Client


class clientMOON(Client):
    def __init__(self, args, id, train_samples, test_samples, **kwargs):
        super().__init__(args, id, train_samples, test_samples, **kwargs)

        self.tau = args.tau
        self.mu = args.mu

        self.global_model = None
        self.old_model = copy.deepcopy(self.model)

    def train(self):
        trainloader = self.load_train_data()
        start_time = time.time()

        # self.model.to(self.device)
        self.model.train()

        max_local_epochs = self.local_epochs
        if self.train_slow:
            max_local_epochs = np.random.randint(1, max_local_epochs //
               2)

        for step in range(max_local_epochs):
            for i, (x, y) in enumerate(trainloader):
                if type(x) == type([]):
                    x[0] = x[0].to(self.device)
                else:
                    x = x.to(self.device)
                y = y.to(self.device)
                if self.train_slow:
                    time.sleep(0.1 * np.abs(np.random.rand()))
                rep = self.model.base(x)
                output = self.model.head(rep)
                loss = self.loss(output, y)
```

```
43
44                  rep_old = self.old_model.base(x).detach()
45                  rep_global = self.global_model.base(x).detach()
46                  loss_con = -
                        torch.log(torch.exp(F.cosine_similarity(rep,
                        rep_global) / self.tau) /
                        (torch.exp(F.cosine_similarity(rep, rep_global) /
                        self.tau) + torch.exp(F.cosine_similarity(rep,
                        rep_old) / self.tau)))
47                  loss += self.mu * torch.mean(loss_con)
48
49                  self.optimizer.zero_grad()
50                  loss.backward()
51                  self.optimizer.step()
52
53          # self.model.cpu()
54          self.old_model = copy.deepcopy(self.model)
55
56          if self.learning_rate_decay:
57              self.learning_rate_scheduler.step()
58
59          self.train_time_cost['num_rounds'] += 1
60          self.train_time_cost['total_cost'] += time.time() - start_time
61
62
63      def set_parameters(self, model):
64          for new_param, old_param in zip(model.parameters(),
                self.model.parameters()):
65              old_param.data = new_param.data.clone()
66
67          self.global_model = model
68
69      def train_metrics(self):
70          trainloader = self.load_train_data()
71          # self.model = self.load_model('model')
72          # self.model.to(self.device)
73          self.model.eval()
74
75          train_num = 0
76          losses = 0
77          with torch.no_grad():
78              for x, y in trainloader:
79                  if type(x) == type([]):
```

```
                    x[0] = x[0].to(self.device)
                else:
                    x = x.to(self.device)
                y = y.to(self.device)
                rep = self.model.base(x)
                output = self.model.head(rep)
                loss = self.loss(output, y)

                rep_old = self.old_model.base(x).detach()
                rep_global = self.global_model.base(x).detach()
                loss_con = -
                   torch.log(torch.exp(F.cosine_similarity(rep,
                   rep_global) / self.tau) /
                   (torch.exp(F.cosine_similarity(rep, rep_global) /
                   self.tau) + torch.exp(F.cosine_similarity(rep,
                   rep_old) / self.tau)))
                loss += self.mu * torch.mean(loss_con)
                train_num += y.shape[0]
                losses += loss.item() * y.shape[0]

        # self.model.cpu()
        # self.save_model(self.model, 'model')

        return losses, train_num
```

4.2 创新点

基于 FedRep 是利用全局共享模型学习地位表征，本地独有数据集训练本地数据头的方式解决联邦学习中的数据异构问题。受到对比学习思想的启发，对比学习从模型层面重新看待数据异构问题，基于模型从整体数据集中学到的特征优于倾斜数据集学到的特征这一直观现象，最大化全局模型与本地模型表征的一致性。

MOON 的关键思想是利用模型表示之间的相似性来校正个体方的局部训练，即在模型层次上进行对比学习。MOON 是一个简单的联邦学习框架，具体做法是通过对比损失函数减小本地模型和全局模型的距离，增大本地模型和前一轮次的本地模型的距离，MOON 具体过程如图 3所示。MOON 可以基于 FedAvg 框架实现，不同之处在于本地训练阶段。FedAvg 的本地训练是在本地执行多次更新，再将更新参数上传至中央服务器聚合参数。而对比学习中的本地训练阶段需要计算监督学习损失，上一轮次的损失，全局损失。由此计算对比损失 l_{con}。从对比损失狗狗是计算可以得出，对比损失旨在减小本地模型与全局模型差异，增大本地模型间训练不同轮次的差异性。总体损失由两部分组成：模型对比损失以及自监督学习损失，将两部分加权相加即为模型的总体损失。

$$l_{con} = -log\frac{exp(sim(z, z_{glob}/\tau)}{exp(sim(z, z_{glob}/\tau) + exp(sim(z, z_{prev}/\tau)}$$

$$l = l_{sup}(w_i^t; (x, y)) + ul_{con}(w_i^t; w_i^{t-1}; w^t; x)$$

PartyLocalTraining(i, w^t):
$w_i^t \leftarrow w^t$
for epoch $i = 1, 2, ..., E$ **do**
  for each batch $\mathbf{b} = \{x, y\}$ of $\mathcal{D}^i$ **do**
    $\ell_{sup} \leftarrow CrossEntropyLoss(F_{w_i^t}(x), y)$
    $z \leftarrow R_{w_i^t}(x)$
    $z_{glob} \leftarrow R_{w^t}(x)$
    $z_{prev} \leftarrow R_{w_i^{t-1}}(x)$
    $\ell_{con} \leftarrow -\log \frac{\exp(\text{sim}(z, z_{glob})/\tau)}{\exp(\text{sim}(z, z_{glob})/\tau) + \exp(\text{sim}(z, z_{prev})/\tau)}$
    $\ell \leftarrow \ell_{sup} + \mu\ell_{con}$
    $w_i^t \leftarrow w_i^t - \eta\nabla\ell$
return w_i^t to server

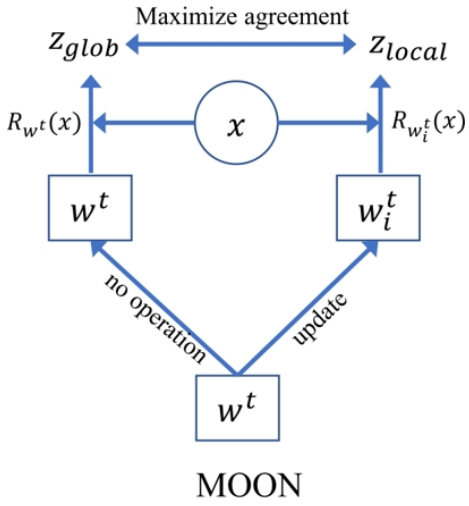

图 3. 对比学习

5 实验结果分析

本文在 MNIST 数据集上进行了实验，共十个类别，特征提取网络为卷积神经网络，通信轮次设为 1000，本地训练轮次设为 5，每个通信轮参与率设为 1。每次通信轮控制台打印当前训练轮次，平均训练损失，平均测试准确率，平均测试 AUC 值，标准测试误差信息。

实验结果如图 4和图 5所示。从实验的训练误差结果图可以得出，随着通信轮次的增加，FedRep 算法收敛速度显著由于梯度下降的方法。客户端数量由 10 增加到 100 以及 200 的过程中，算法的收敛性以及稳定性均取得了显著提升。在图中的实验设置下 FedRep 的性能最优。

FedRep 框架结合对比学习的实验结果如图 5所示。在少于 50 轮次，算法快速收敛，并达到了 98% 左右的测试准确度。由此可见，改进的算法一指数级的速度收敛到最优表征，对于数据集的不同异构程度算法性能稳健。实验过程中观察到模型对于新加入的客户端非常友好，能够快速将全局共享模型迁移到新客户端，使得新客户端关注于训练本地数据集的独特的数据头。

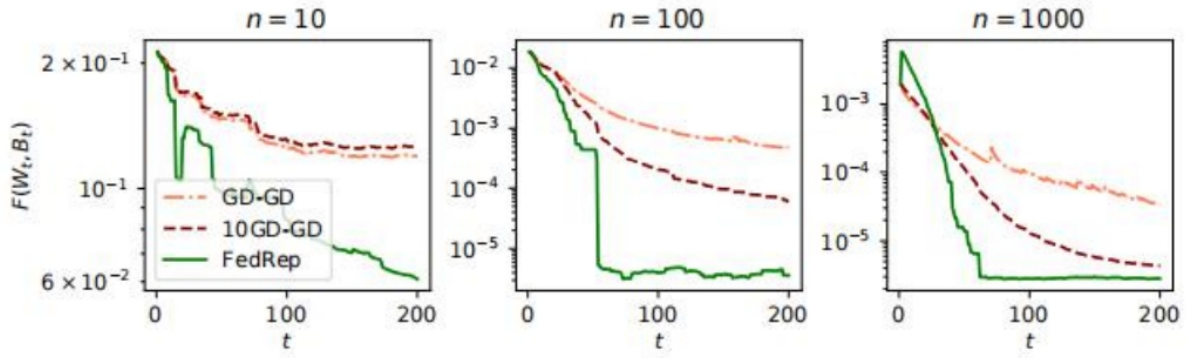

图 4. 训练误差

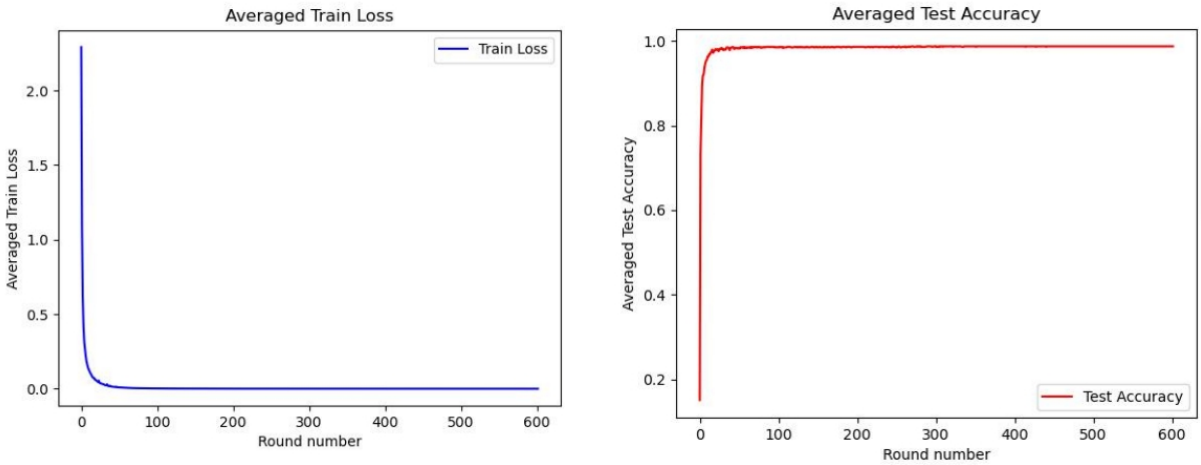

图 5. 实验结果

6 总结与展望

本文复现了一种新的表征学习框架和联邦学习的算法，针对联邦学习数据异构的特点学习共享的全局表征模型以及针对本地数据标签差异性训练的本地特征头，实现了利用全局共享低维特征提高分类精度的同时实现了个性化联邦学习。非 IID 是联邦学习有效性的关键挑战。为了提高联合深度学习模型在非 IID 数据集上的性能，受对比学习思想的启发，结合 FedRep 模型以及对比学习（MOON）思想，在 FedRep 的基础上加入对比损失。MOON 通过增大客户端上本轮训练与上一轮次的差距，减小客户端与全局模型差距的模型层次的对比学习从而进一步提高模型的性能。

FedRep 框架是一个通用的、简单的框架，可以轻松地应用于广泛的联邦学习问题，例如从线性回归到图像分类，以及情感分析等问题。可以在各种个性化的联邦学习基线上显著提高本地客户端的准确性。现有的工作表明，微调全局联邦学习方法，也往往表现得非常好。我将在今后的工作中进一步研究这一现象。在 FedRep 框架进一步扩展，以提高相对于微调方法的性能。

参考文献

[1] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818*, 2019.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[3] Fei Chen, Mi Luo, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning with fast convergence and efficient communication. *arXiv preprint arXiv:1802.07876*, 2018.

[4] Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning: A meta-learning approach. *arXiv preprint arXiv:2002.07948*, 2020.

[5] Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. Federated learning with compression: Unified analysis and sharp guarantees. In *International Conference on Artificial Intelligence and Statistics*, pages 2350–2358. PMLR, 2021.

[6] Filip Hanzely and Peter Richtárik. Federated learning of a mixture of global and local models. *arXiv preprint arXiv:2002.05516*, 2020.

[7] Xinmeng Huang, Ping Li, and Xiaoyun Li. Stochastic controlled averaging for federated learning with communication compression. *arXiv preprint arXiv:2308.08165*, 2023.

[8] Yihan Jiang, Jakub Konečnỳ, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning. *arXiv preprint arXiv:1909.12488*, 2019.

[9] Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10713–10722, 2021.

[10] Paul Pu Liang, Terrance Liu, Liu Ziyin, Nicholas B Allen, Randy P Auerbach, David Brent, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. *arXiv preprint arXiv:2001.01523*, 2020.

[11] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *arXiv preprint arXiv:2002.10619*, 2020.

[12] Dohyung Park, Anastasios Kyrillidis, Constantine Caramanis, and Sujay Sanghavi. Finding low-rank solutions via nonconvex matrix factorization, efficiently and provably. *SIAM Journal on Imaging Sciences*, 11(4):2165–2204, 2018.

[13] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. *Advances in neural information processing systems*, 30, 2017.

[14] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems*, 33:7611–7623, 2020.

[15] Kangkang Wang, Rajiv Mathews, Chloé Kiddon, Hubert Eichner, Françoise Beaufays, and Daniel Ramage. Federated evaluation of on-device personalization. *arXiv preprint arXiv:1910.10252*, 2019.

[16] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. Salvaging federated learning by local adaptation. *arXiv preprint arXiv:2002.04758*, 2020.