在异构结构化客户端模型中进行高效的联邦学习

刘和平，胡尧

摘要

联邦学习由于其在处理来自不同终端的数据时其隐私保护方面的优异性能，近年来受到广泛关注。然而，之前的联邦学习中普遍采用同构深度学习模型，没有考虑不同客户端中不同数据之间的差异，导致学习性能较低，通信成本较高。为此，本文提出了一种具有异构结构化客户端模型的新型联邦学习框架，用于处理不同的数据规模，并研究了其相对于具有同构模型的典型的联邦学习的优越性。此外，在客户端模型上采用奇异值分解，减少传输数据量，即通信成本。根据所上传参数和模型的异构性特点，提出了中心服务器上多模型的聚合机制。提出的框架应用于四个基准分类数据集和电磁辐射强度时间序列数据的趋势跟踪任务。实验结果表明，该方法能有效提高局部学习模型的精度，显著降低通信成本。在这篇文章的基础上，我们对其代码进行了改进，使得其通信成本进一步降低。

关键词：联邦学习; 异构结构化模型; 神经网络; 奇异值分解

1 引言

人们普遍认识到，存储在各种类型智能终端上的不同用户或设备生成的数据构成了当今大数据的主要组成部分。因此，设计强大的机器学习方法来探索来自不同终端的集成数据的内在价值已成为大数据[1][2]中的一个重要研究课题。然而，来自不同用户的数据可能非常的私密。由于担心隐私泄露，用户可能不愿意与他人共享此类数据，并且基于现有的传统数据收集的机器学习方法很难在隐私保护[3]的前提下工作。

遵循集中收集或数据最小化的原则，谷歌公司引入了一个保护隐私的机器学习框架，名为联邦学习[4]。其想法是，在每个智能终端（即客户端）上，训练本地模型，特别是深度学习模型，以从许多用户的私人数据中提取公共信息。随后，每个局部模型的参数，例如深度学习网络的权重，被上传到服务器。在中央服务器上，对这些上传的参数进行聚合优化，以提高中央和本地模型的学习性能。然后，聚合的参数被发送到参与的客户端以重新优化本地模型。显然，该框架可以从根本上保护用户的隐私，因为它不会隐式地交换或收集用户生成的数据，而只聚合本地模型参数。此外，模型参数的聚合可以实现多源信息融合，在一定程度上解决了传统方法中的冷启动问题。

已经有许多杰出的研究致力于扩展联邦学习的实用性。例如，McMahan 等人[5]提出的联邦平均算法，减少了训练批量大小或增加了本地训练的轮次，以缓解通信效率的瓶颈。在联邦平均算法的基础上，Chen 等人[6]提出了局部模型的异步学习策略和时间加权聚合方法，以提高通信效率和收敛精度。与传统的机器学习方法相比，联邦学习侧重于保护用户的隐私。Bonawitz 等人[7]提出了一种新的安全聚合协议，仅当参与客户端的数量满足特定条件（如指定数量）时，才能处理服务器上上传的参数。该策略进一步降低了服务器受到攻击而导致数据泄漏的可能性。此外，许多研究在隐私保护的前提下利用联邦学习来处理孤立的数据不足问题。Chen 等人[8]将协同过滤算法嵌入到联邦学习的框架中，以

安全地获得全局的社会偏好。所有这些工作通常基于这样的假设，即所有客户端都采用同质结构模型，这可能会导致在本地数据量不平衡的情况下的低精度和额外的计算成本。

在联邦学习的框架中，由于存在重度用户和轻度用户，在不同客户端上生成和存储的本地数据集的数量可能会显著不同[4]。因此，在每个客户端上构建相同结构的局部模型是不合理的。直观地说，在小规模数据集上训练的深度模型往往会过度拟合，而浅层模型往往无法充分学习大规模数据中的知识。为了确保联邦学习的性能，应该高度重视这一挑战，例如为具有不同数据规模的客户提供具有各种不同结构的模型。除此之外，通信效率的瓶颈也是一个不可避免的问题[9]。由于参与的客户端（如移动电话和平板电脑）通常具有有限的可用数据和不可靠、昂贵的网络连接，高通信成本使得一些带宽较小的设备无法参与参数聚合。相比之下，随着硬件和移动图形处理单元的发展，这些设备逐渐具有强大的计算能力，并可以使用复杂的深度神经网络[10]。然而，此类模型的训练更新包含大量参数矩阵，使得通信成为更严重的瓶颈。通常，通信成本的降低比计算成本的降低更值得关注。

为此，这篇报告研究了联邦学习下异构结构化局部模型的优势，并提出了一种有效的通信方法来降低通信成本。具体而言，在服务器上构建了两个异构结构模型，即深度神经网络和浅层神经网络，并将其应用于本地客户端。然后，每个客户端根据其存储数据集的数量选择本地模型。考虑到有限的带宽，我们随后提出利用奇异值分解（SVD）来获得高维模型参数的奇异值矩阵。上传奇异值而不是整个模型矩阵，以降低通信成本。基于上传参数和模型的异构特性，提出了一种服务器上的聚合方法。本报告对原论文的代码改动主要：

1）生成数据的方式，增加一个变量来记录不变数据，使得生成数据的效率提高了许多。

2）模型的布置，原本的代码是在服务器初始化一个深层模型，一个浅层模型，把深层模型和浅层模型都发给客户机，客户机同时初始化两个模型，根据自身情况来选择使用哪个模型。改进代码后，客户机把自身情况发送给服务器，让服务器决定要给客户机发送什么模型。这样客户机就只用初始化一个模型，同时也可以一定程度上减少通讯开销。而服务器一般都是算力很强大的平台，对服务器增加的操作基本上没影响。

3）对模型的上传：原来的代码只对一个全连接层进行了上传，而我们选择把所有的全连接层进行上传。

2 相关工作

2.1 传统的联邦学习策略

最近关于联邦学习的研究主要包括降低通信成本、优化聚合策略和增强隐私保护。

关于通信成本的降低，Zhu 和 Jin[11]通过将总体通信成本和全局测试误差作为优化目标，提出了一种多目标联合学习算法。然后，使用多目标优化算法搜索客户端中所有神经网络的最优结构。朱航宇通过使用 SET 算法来稀疏化模型，也使用多目标优化算法搜索客户端中所有神经网络的最优结构。为了进一步降低进化优化产生的计算和通信成本，他们[12]在服务器上设置了主模型，并引入了双采样技术。具体地，主机的随机采样子模型被传输到多个随机采样的客户端以用于训练而无需初始化。Mao 等人[13]提出了一种自适应量化方法，该方法根据局部梯度的更新自适应地调整量化级别。类似地，Yan 等人[14]提出了一种动态量化随机梯度下降（SGD）方法，以探索通信成本和建模误差之间的

权衡，不同任务的实验证明了其方法的有效性。

关于中心模型的聚合优化，Yurochkin 等人[15]应用贝叶斯非参数机制来识别局部模型中与其他模型中神经元匹配的神经元子集。然后将匹配的神经元上传到服务器以形成全局模型。基于神经网络中不同深度层的不同性能，Chen 等人[5]提出了一种客户端参数的异步更新方法，以降低每一轮的通信成本。同时，他们提出了一种时间加权聚合方法，通过利用先前上传的参数来提高收敛速度，同时减少所需的通信轮次。

为了增强隐私保护，采用了差异隐私方法，通过向深度学习模型添加高斯噪声来减少查询特定数据存储库时个人信息的影响[16]。Gayer 等人将差异隐私引入到联合学习中，该技术不显示参与离线培训的客户的任何信息，并进一步保护整个私有数据集免受其他客户的攻击

除此之外，一些最新技术逐渐与联邦学习相结合。Jiang 等人将联邦学习作为模型不可知元学习的实际应用的自然来源。根据他们的声明，联邦学习的目标应该是个性化性能，即共享全局模型只是优化个性化性能的必要部分步骤。为了缓解不同节点之间的域转移，Peng 等人[17]扩展了对抗性自适应技术，以将从不同节点学习到的表示与目标节点的表示对齐。此外，还开发了一种特征解纠缠方法，以增强正向知识转移。

2.2 基于 SVD 的模型压缩方法

我们的框架旨在通过使用 SVD 分解得到的较小规模的奇异值来降低通信成本。在本小节中，我们首先回顾了一些常见的矩阵分解方法，包括三角分解[18]、QR 分解[19]和 SVD。然后，阐述了基于 SVD 的模型压缩方法。

三角因式分解将正方形矩阵 ($A_{m\times m}$) 分解为下三角矩阵 ($L_{m\times m}$) 和上三角矩阵 ($T_{m\times m}$) 的乘积。三角因式分解公式如下：

$$A_{m\times m} = L_{m\times m} \cdot T_{m\times m} \tag{1}$$

三角因式分解在求解线性方程的平方系统和计算矩阵的行列式方面发挥着关键作用[20]。作为比较，QR 因式分解将矩阵（$A_{m\times n}$）分解为正交矩阵（$Q_{m\times n}$）（即 $QQ^T = I$）和上三角矩阵（$R_{n\times n}$）的乘积。公式如下：

$$A_{m\times n} = Q_{m\times n} \cdot R_{n\times n} \tag{2}$$

QR 分解通常用于解决线性最小二乘和线性逼近问题[21]。SVD 可以获得任意矩阵 $A_{m\times n}$ 的全秩分解，如下所示：

$$A_{m\times n} = U_{m\times m} \cdot \Sigma_{m\times n} \cdot V^T_{n\times n} \tag{3}$$

其中 U 和 V 分别称为左奇异矩阵和右奇异矩阵。Σ 是对角矩阵，其中对角元素是奇异值。奇异值对应于矩阵中隐含的重要特征，重要性程度与其值呈正相关。此外，奇异值矩阵反映了左奇异向量行和右奇异向量列之间的重要性。在这些矩阵分解方法中，只有 SVD 可以通过在逆过程中调整奇异值的比例来执行矩阵压缩。压缩过程如下：

$$A_{m\times n} = U_{m\times m} \cdot \Sigma_{m\times n} \cdot V^T_{n\times n} \approx U_{m\times r} \cdot \Sigma_{r\times r} \cdot V^T_{nr\times n} = A^{'}_{m\times n} \tag{4}$$

其中 r 表示保留的奇异值，而 $A^{'}_{m\times n}$ 表示与 $A_{m\times n}$ 大小相同的压缩矩阵。与 SVD 类似，主成分分析（PCA）[22]等效于仅使用 SVD 中的右奇异矩阵，即 V^T，而不考虑原始矩阵中行之间的关系。为此，本

研究采用 SVD 压缩模型参数，以降低通信成本。

SVD 通常被用作低秩近似，通过使用秩小于原始矩阵的另一个矩阵来表示矩阵内的信息来解决大规模参数问题[23]。为了获得适当的秩的配置，Kim 等人[24]将整个网络作为一个整体，并提出了一个精度度量来衡量准确性和复杂性关系。整体精度的计算由各层精度的联合分布表示，联合分布通过考虑独立性的乘积计算。开发了一种将 PCA 能量和基于测量的方法相结合的度量，以减少低复杂度或高复杂度下单个度量的不足。类似地，Idelbayev[25]将每个层的最优秩的确定公式化为混合离散连续优化，该优化包括最小化分类损失和模型选择成本。他们证明，可以通过将未压缩网络上的 SGD 与 SVD 步骤交织来优化公式，SVD 步骤可以决定最佳秩和模型参数。在[26]中，研究人员预定义了最小有效奇异值的数量，并采用迭代“训练-压缩-再训练”方法进行 SVD。SVD 进一步与剪枝和聚类压缩相结合。采用贪婪搜索算法来搜索这些方法的组合顺序。

Zhou 等人[27]也在联邦学习中采用 SVD 以降低通信成本，但我们完全不同。在他们的工作中，所有分解的矩阵，即 U、V 和 $/Sigma$ 被上传到服务器，这增加了传输数据的数量。在服务器上，对所有上传的矩阵执行反向 SVD，这增加了服务器的工作量。相反，我们的方法只将全连接层的奇异值矩阵传输到服务器，然后通过更新的奇异值阵以及本地左/右奇异矩阵在本地执行逆 SVD。因此，可以使用较低的通信成本和本地计算消耗，根据来自其他客户端的权重来修改来自全连接的权重的关键信息。在我们的方法中，在信息更新过程中，所有矩阵始终保持满秩。因此，所提出的框架不属于[24],[26]那样的低秩近似，而是全秩通信高效模型共享策略。

3 本文方法

3.1 问题的描述

联邦学习已经成为一个很有前途的领域，专注于训练全局机器学习模型，而不隐式地访问存储在每个客户端的训练数据。在联邦学习中，每个参与者上传到服务器的信息不再是原始数据，而是学习到的局部模型参数（例如，权重）。目前盛行的范式通常将联邦学习表述为一个经验风险最小化问题，具体如下：

$$\mathcal{L}_{fl}(\Theta;\mathcal{D}) = \sum_{i=1}^{n} \mathcal{L}_i(\Theta;\mathcal{D}_i) \tag{5}$$

其中 n 是参与训练的客户数量，Θ 代表要学习的全局模型的参数集，$\mathcal{L}_i(\Theta;\mathcal{D}_i)$ 代表全局模型 Θ 在客户机 i 上存储的具有分布 $\mathcal{D}_i$ 的本地数据上的损失。让 $\mathcal{L}_{con}$ 代表传统模式下的总学习损失，其中所有数据集 $\mathcal{D} = \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_K$ 集中起来训练一个全局模型。理想情况下，$\mathcal{L}_{fl}$ 应该非常接近于 $\mathcal{L}_{con}$。给定一个非负数 δ，联邦学习的优化可以表达为：

$$|\mathcal{L}_{fl} - \mathcal{L}_{con}| < \delta \tag{6}$$

$$\Theta^* = argmin|\mathcal{L}_{fl} - \mathcal{L}_{con}| \tag{7}$$

传统联邦学习的决定性特征之一是所有参与的客户端都采用同构的本地模型，而忽略了不同客户端的本地样本之间的差异，导致学习性能低下，通信成本高。为了开发异构的联邦学习，我们需要解决的第一个问题是在服务器上设计异质结构的全局模型。随后，由于服务器上有不同结构的全局模型，

为每个客户端分配合适的全局模型是我们应该考虑的另一个挑战。此外，如何研究有效的本地模型训练方法和有效的传输策略也是我们必须解决的重要问题。在服务器接收到来自参与客户端的上传内容后，开发异质模型的聚合方法是最后需要解决的问题。本研究试图研究异质结构联邦学习的优越性，从而为上述问题提供一个实例解决方案。

3.2 方法的框架

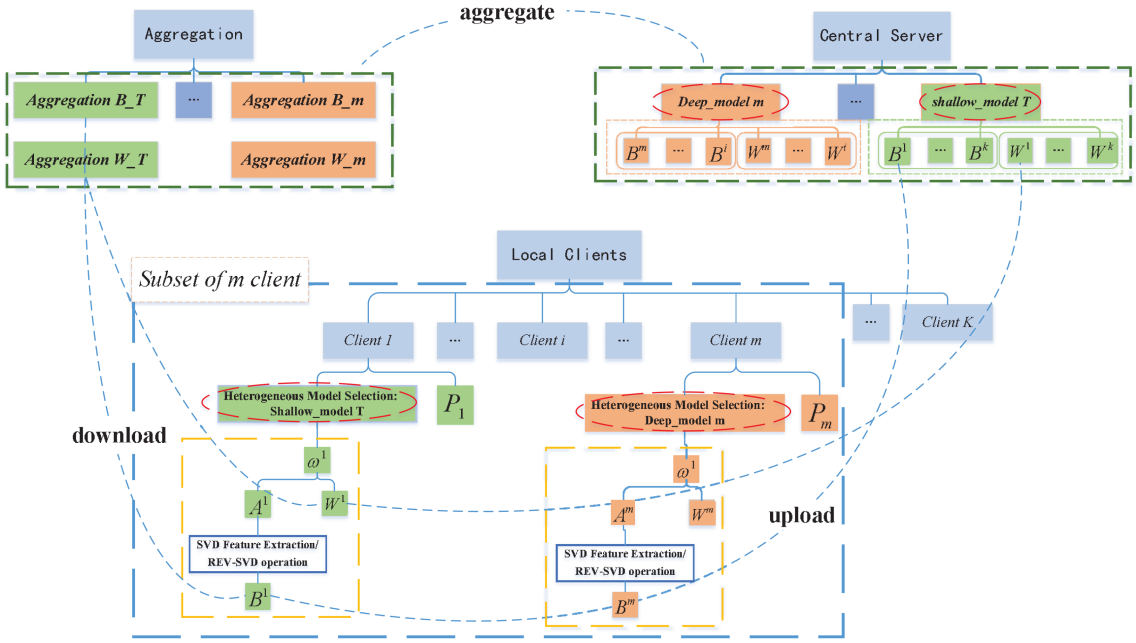

图 1: 基于 SVD 低通信方法的异构结构化联邦学习框架

图 1 显示了所提出的异质结构的联邦学习框架，该框架通过高效的通信方法得到了加强。与经典的联邦学习一样，我们提出的框架包含了本地模型训练和中心参数聚合阶段。然而，每个步骤都存在本质的区别。在本地模型训练中，每个客户端首先根据数据大小选择自己的本地模型，如红圈模块所示。换句话说，不同的本地客户端采用的学习模型可能具有不同的结构。需要指出的是，本研究旨在强调采用各种结构的中心模型的优越性，而不是提出一个具体的模型选择标准。客户端基于本地数据集训练他们的学习模型并将参数上传到中央服务器。上传的对象不是传输完整的模型参数，而是由来自特征提取层的小尺度参数（B^m）和通过 SVD 获得的全连接层的奇异值矩阵（W^m）组成。随后，如绿框所示，提出了一个多中心的聚合机制，用于在服务器上聚合异质结构的局部模型参数。所有客户都从中央服务器上下载聚合。此外，正如黄色模块一样，我们对聚合的奇异值矩阵（B_T）进行反向 SVD，以获得全连接层的更新参数。聚合的结果（W_T）直接作为特征提取模块的更新参数。最后，更新后的局部模型被认为是后续局部训练的初始化。

3.3 基于 SVD 通信的高效联邦平均算法

在传统的联邦平均算法中，参与客户端的所有模型参数都被上传到服务器进行聚合和更新。上传和下载过程中巨大的传输量需要很高的带宽，导致高昂的通信成本。由于客户端通常采用神经网络作为本地模型，我们分析了神经网络的特点，提出了一种有效的上传/下载策略来降低传输的大小。

神经网络，尤其是深度学习模型，通常由两个模块组成。第一个是包含许多层的特征提取模块，如卷积层[28]或长短期记忆（LSTM）层[29]。另一个是特定任务（如分类、回归等）的全连接层。特征提取层主要用于提取高阶特征，这对保证整个模型的性能起着至关重要的作用。全连接层的目的是将提取的特征非线性地结合起来，实现最终的任务。为了降低通信成本，同时又不造成很大的精度损失，

我们利用 SVD 来分解客户端模型密集层的参数，即高维参数。然后，在典型的联邦平均算法中不上传完整的权重矩阵，而只上传奇异值矩阵来参与聚合。特征提取模块的参数仍然完全上传，以保证特征提取模块的泛化能力。

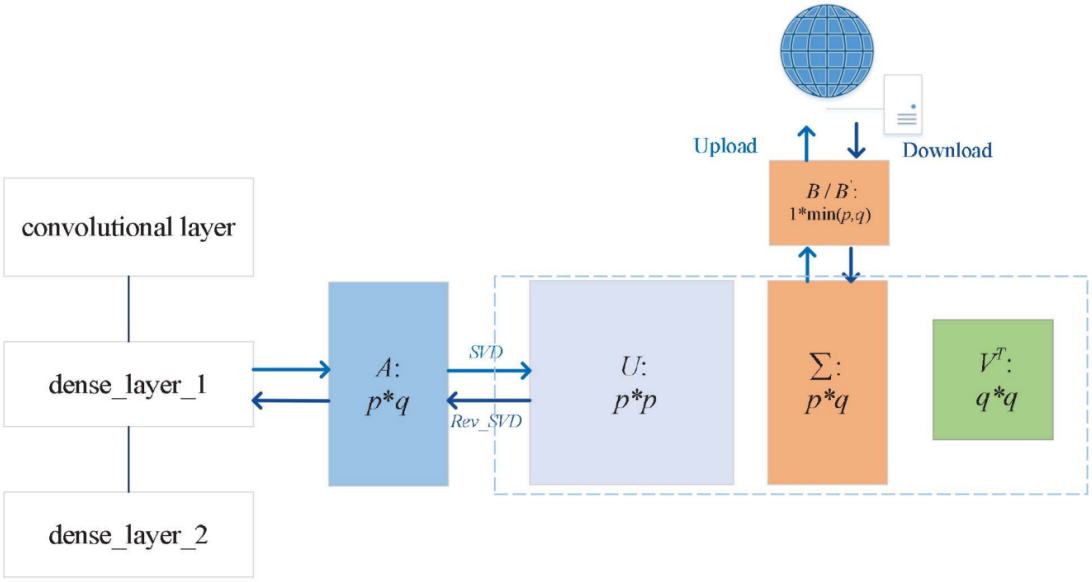

图 2: 基于 SVD 的特征提取与更新原理图

以 CNN 为例，在通信中利用 SVD 与全连接层的整个过程如图 2 所示。我们首先使用 SVD 来分解 Dense_1 层的权重矩阵 $A_{p\times q}$，也就是卷积层之后的第一个密集层。对于 $A_{p\times q}$，p 和 q 分别是输入层和隐藏层中单元的数量。p 也是特征提取输出单元的数量或提取特征的维度。在得到奇异值矩阵 $\Sigma_{p\times q}$ 以及左/右奇异值矩阵后，每个客户端将奇异值矩阵 B 的对角线元素上传到服务器进行聚合。由于奇异值对应于矩阵中隐含的重要特征，因此可以通过聚合其他客户端的信息来更新本地的重要程度。有了聚合的特征矩阵 $B^{'}$，对存储的左/右奇异矩阵进行反向 SVD 运算，以更新密集层的权重。由于特征提取模块的参数以完全上传的方式更新，以确保泛化，因此存储的左/右奇异矩阵有望保持每个局部模型的个性化。

通过 SVD 可以大大降低密集权重的总通信成本。对于第 i 个客户端模型，未用基于 SVD 的特征提取方法处理的层的权重矩阵表示为 W_i，而全连接层的权重在执行 SVD 之前和之后表示为 $A_{p_i\times q_i}$ 和 B_i。SVD 之后，上传/下载的数据量从 $p_i\times q_i$ 减少到 $\min(p_i, q_i)$，这仅仅是原来通信费用的 $1/q_i$ 或 $1/p_i$。从参与联邦学习的所有客户的角度来看，我们假设参与的客户数为 l，在传统的联邦学习中上传/下载的数据量为 $2\sum_{i=1}^{l}(|W_i|+|A_{p_i\times q_i}|)$，在对全连接层的参数进行 SVD 后，减少为 $2\sum_{i=1}^{l}(|W_i|+|B_i|)$，它们之间的差异可以计算为：

$$2\sum_{i=1}^{l}(|A_{p_i\times q_i}|-|B_i|=2\sum_{i=1}^{l}(min(p_i,q_i)\times(max(p_i,q_i)-1) \tag{8}$$

客户端参与越多，所提出的特征上传算法越有效。此外，该算法仅传输关键特征，将左右奇异矩阵保留在客户端，进一步降低了隐私泄露的可能性，增强了系统的保密性。

3.4 具有异构结构化客户端模型的联邦学习聚合策略

如图 1 所示，所提出的联邦学习框架与传统的框架有很大的不同，传统的同构模型聚合方法不再适用。在本节中，我们提出了一种针对异构结构化模型的多中心模型聚合机制。

对于第 i 个客户模型，需要聚合的参数包括两部分，即 W_i 和 B_i，它们代表了模型的不同内容，应分别进行相应的聚合。首先，根据客户采用的模型的结构对其进行分类，将具有相同结构的客户模型归为同一类。然后，我们进一步将通信轮次分为特定轮次和常规轮次。在前一种情况下，特定任务层的完整参数被上传，这有望获得一个共享的中心模型，避免局部最优。在后一种通信回合中，奇异值矩阵被上传以解决通信瓶颈问题。在服务器上，上传的集群模型 W_i 和 B_i 分别根据任何聚合优化（例如平均加权）进行聚合。显然，聚合模型的数量是由模型的类型决定的，这与传统的只使用一个聚合模型是不同的。此外，对于不同种类的客户端模型，聚集机制也可以不同。聚合后，得到的权重和奇异值被下载到客户端进行更新。

从个性化推荐的角度来看，根据用户产生的数据集的大小对用户进行分类是有意义的。拥有小数据集的客户可以被认为是处于早期认知阶段的用户，他们一般对物品的知识较少，偏好波动较大。这些客户被聚集起来，期望通过交换波动的认知来广泛地扩展他们对未知事物的认知，从而逐渐建立新的偏好。作为比较，包含大量数据集的客户对应于具有稳定认知和个人偏好的用户。他们很少需要在信息交流过程中通过来自他人的波动性认知来发展新的偏好。更重要的是，从处于早期认知阶段的客户那里获得的认知甚至可能成为一种干扰。

4 复现细节

4.1 与已有代码对比

此次复现参考的代码是论文的作者直接分享的。原有代码略显冗余，我把代码按自己的逻辑梳理、重写了一遍。在此之上，主要改变了代码的三处：生成数据的方式、对模型的布置和对模型的上传。

生成数据的方式：在给各个客户端生成非独立同分布的数据时，原来代码在生成数据上主要有两个缺点：1. 生成数据的分布方式固定；2. 生成数据的速度慢。对于第一点，原有的代码直接将数据的分布固定，如图三所示：

```
DATA_CLASS_TRAIN_SIZE_P = {
    '5_1':[{'class_distr': [0.0, 0.0, 0.11977929278346476, 0.0, 0.0, 0.0, 0.0, 0.07575154934094151, 0.8044691578755937, 0.0],
      'train_size': 10616},
     {'class_distr': [0.5591156977548751, 0.0, 0.0, 0.0, 0.44088430224512487, 0.0, 0.0, 0.0, 0.0, 0.0],
      'train_size': 2155},
     {'class_distr': [0.0, 0.0, 0.0, 0.6312042801049751, 0.0, 0.0, 0.0, 0.0, 0.0, 0.3687957198502496],
      'train_size': 7029},
     {'class_distr': [0.0, 0.097515645626172 07, 0.5206530594081663, 0.0, 0.0, 0.0, 0.0, 0.3818312949656616, 0.0, 0.0],
      'train_size': 2980},
     {'class_distr': [0.0, 0.0, 0.0, 0.0, 0.0, 0.12757414835970785, 0.0, 0.0, 0.1439434303278 59, 0.7284824213124331],
      'train_size': 5220}],
    '5_2':[{'class_distr': [0.6372845942367841, 0.0, 0.0, 0.0, 0.362715405763215 87, 0.0, 0.0, 0.0, 0.0, 0.0],
      'train_size': 6452},#14452
     {'class_distr': [0.0, 0.0, 0.0, 0.53064217193498, 0.16898513273319787, 0.0, 0.3003726953318221, 0.0, 0.0, 0.0],
      'train_size': 6348},
     {'class_distr': [0.0, 0.0, 0.0, 0.8985034297094225, 0.0, 0.0, 0.0, 0.0, 0.0, 0.10149657029057745],
      'train_size': 4178},#4178},
     {'class_distr': [0.0, 0.21221019620266024, 0.3183871174617743, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.4694026863355654],
      'train_size': 4603},#4603},
     {'class_distr': [0.0, 0.0, 0.0, 0.2916701496981477, 0.0, 0.0, 0.7083298503018522, 0.0, 0.0, 0.0],
      'train_size': 6079}],
    '5_3':[{'class_distr': [0.49480815252999644, 0.0, 0.0, 0.5051918474700036, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      'train_size': 14243},
     {'class_distr': [0.0, 0.41554020631608524, 0.0, 0.0, 0.5183292930751032, 0.0, 0.06613050060881166, 0.0, 0.0, 0.0],
      'train_size': 5584},
     {'class_distr': [0.6696557958577942, 0.0, 0.19497885461011902, 0.1353653495320869, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
      'train_size': 8481},
     {'class_distr': [0.0, 0.22942872601850678, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.7705712739814932, 0.0],
      'train_size': 4583},
     {'class_distr': [0.0, 0.0, 0.0, 0.599964712249932, 0.0, 0.0, 0.4000352877500681, 0.0, 0.0, 0.0],
      'train_size': 4085}],
    '5_4':[{'class_distr': [0.21198145753809364, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.7880185424619064, 0.0],
      'train_size': 14457},
     {'class_distr': [0.0, 0.0, 0.0, 0.0, 0.534305050565261, 0.0, 0.07087705175130654, 0.3948178976834325, 0.0, 0.0],
      'train_size': 5312},
     {'class_distr': [0.32112867405167445, 0.0, 0.43817611960844055, 0.0, 0.0, 0.2406952063398851, 0.0, 0.0, 0.0, 0.0],
      'train_size': 5323},
     {'class_distr': [0.10016095487403315, 0.0, 0.0, 0.0, 0.0, 0.5581735790857962, 0.3416654660401708, 0.0, 0.0, 0.0],
```

图 3: 原有代码的数据分布

而我们通过一个函数，如图四，在这个项目开始时，会自动随机的生成每个客户端的数据的分布，且和原来的代码设定的分布效果一样，即每个客户端之间是非独立同分布的：

```python
def gen_non_iid_weights():
    global SET_VALUE
    if DATA_TYPE == "MNIST":
        (x_train, y_train), (x_test, y_test) = mnist.load_data()
    elif DATA_TYPE == "CIFAR":
        (x_train, y_train), (x_test, y_test) = cifar10.load_data()
    else:
        (x_train, y_train), (x_test, y_test) = cifar100.load_data()

    x = np.concatenate([x_train,x_test]).astype('float32')
    y = np.concatenate([y_train,y_test])

    classes_num = np.unique(y)
    classes = np.array(range(classes_num.size))
    classes_client_num = int(classes_num.shape[0] / 4)
    num_classes_this_client = random.randint(classes_client_num, classes_client_num+1)
    classes_this_client = random.sample(classes.tolist(), num_classes_this_client)

    min_train = 4000
    mid_train = 8000
    SET_VALUE = mid_train
    max_train = 12000

    weights_aa = []
    for i in range(MIN_NUM_WORKERS_P):
        w = np.array([random.random() for _ in range(num_classes_this_client)])
        weights = np.array([0.] * classes.shape[0])
        for j in range(len(classes_this_client)):
            weights[classes_this_client[j]] = w[j]
        weights /= np.sum(weights)
        if i < MIN_NUM_WORKERS_P/2:
            train_size = random.randint(min_train, mid_train)
            # print(i, "          ", train_size)
        else:
            train_size = random.randint(mid_train, max_train)

        weights_aa.append({'train_size':train_size,'class_distr':weights.tolist()})
    return weights_aa
```

图 4: 对于数据分布改进的代码

对于第二个小点，我们发现原来的代码造成数据生成慢的原因是有一个变量会重复的计算。如图五中的 candidates_index 这个变量。

```python
def sample_single_non_iid(self, x, y, weight=None):
    # first pick class, then pick a datapoint at random
    chosen_class = np.random.choice(self.classes, p=weight)
    #print('sample_single_non_iid_y.shape',y.shape)
    candidates_idx = np.array([i for i in range(y.shape[0]) if y[i] == chosen_class])
    idx = np.random.choice(candidates_idx)
    return self.post_process(x[idx], y[idx])
```

图 5: 原有代码的数据生成

对于这个变量，我们只需把他设置为在初始化整个类时计算一次即可，因为这个变量对于同一群数据得到的结果也是一样的。经过我们的改进之后，产生一个 10000 个样本的数据集从原来的 192.5231010913849s 降低到 1.7408041954040527s

模型的布置: 原有的代码中，每次服务器向客户端发送模型时都要把深模型和浅模型一起发送了，而且每个客户端都要维护深模型和浅模型，但是这是很不合理的，因为每个客户端其实只需要一个模型。于是，我们把客户端的信息在其启动的时候就发送给服务器，让服务器判断要给客户端发送什么类型的模型，这样客户端的操作就和在传统的联邦学习中一样的了。如图六，为原有的代码服务器发送给客户端的信息，图七为改进了之后的代码服务器发送给客户端的信息。

```
for rid in self.ready_client_sids:
    emit('stop_and_eval', {
        'round_number': self.current_round,
        'ready_client_sids': rid,
        'deep_model_id': self.deep_model_id,
        'shallow_model_id': self.shallow_model_id,
        'deep_current_weights': obj_to_pickle_string(self.global_model.deep_current_weights),
        'shallow_current_weights': obj_to_pickle_string(self.global_model.shallow_current_weights),
        'current_weights': obj_to_pickle_string(self.global_model.current_weights),
        'weights_format': 'pickle'
    }, room=rid)
```

图 6: 原有的代码服务器发送给客户端的信息

```
for rid in self.shallow_rid:
    emit('request_update', {
        'ready_client_sids': rid,
        'round_number': self.current_round,
        'current_weights': obj_to_pickle_string(self.global_model.shallow_current_weights),
        'weights_format': 'pickle',
        'run_validation': self.current_round % FLServer.ROUNDS_BETWEEN_VALIDATIONS == 0,
    }, room=rid)
for rid in self.deep_rid:
    emit('request_update', {
        'ready_client_sids': rid,
        'round_number': self.current_round,
        'current_weights': obj_to_pickle_string(self.global_model.deep_current_weights),
        'weights_format': 'pickle',
        'run_validation': self.current_round % FLServer.ROUNDS_BETWEEN_VALIDATIONS == 0,
    }, room=rid)
```

图 7: 改进的代码服务器发送给客户端的信息

改进的代码中服务器会对相应的客户端发送不同的模型，而且客户端内部只用维护一个模型就行，这极大地减轻了客户端的压力（如果客户端维护两个模型的话，每次客户端与服务器交流前都要判断自己的模型是哪种类型的），减少了代码的冗余程度。

对模型的上传：原有的代码仅对一层全连接层使用 SVD 模型压缩，我们把所有的全连接层都使用 SVD 模型压缩，进一步加快通信效率。

4.2 实验环境搭建

使用的电脑 CPU 为 Inter(R) Core(TM) i7-7700, 无 GPU，使用的操作系统为 Windows10，选择的编程语言是 Python，运用的 IDE 是 Pycharm。服务器和客户端之间的交流使用 SocketIO 来实现，文中用到的模型和模型的训练等都是使用 TensorFlow 来实现的。

4.3 创新点

提出了一种异构结构模型的联邦学习框架，和传统的联邦学习只有一个全局模型不同，其有多种结构的全局模型。提出看一种基于奇异值分解的高效通信方法，而且还在一定程度上降低了隐私泄露的可能性。

5 实验结果分析

实验使用的数据集是 MNIST，初始化五个客户端，把 MINST 数据非独立同分布的分给每个客户端，每个客户端随机分到的标签类型只有两到三种。随机选取两个客户端分配较多的数据，另外三个分配交少的数据。在服务器建立两个全局模型，一个是深模型，有四层卷积层，两层全连接层；一个是浅模型，有两层卷积层和两层全连接层。随着初始化，服务器会给数据量较多的客户端分配深模型，

给数据量较少的客户端分配浅模型。然后开始训练，直到客户端和服务器之间的交流次数达到 400 次。结果如下图所示：

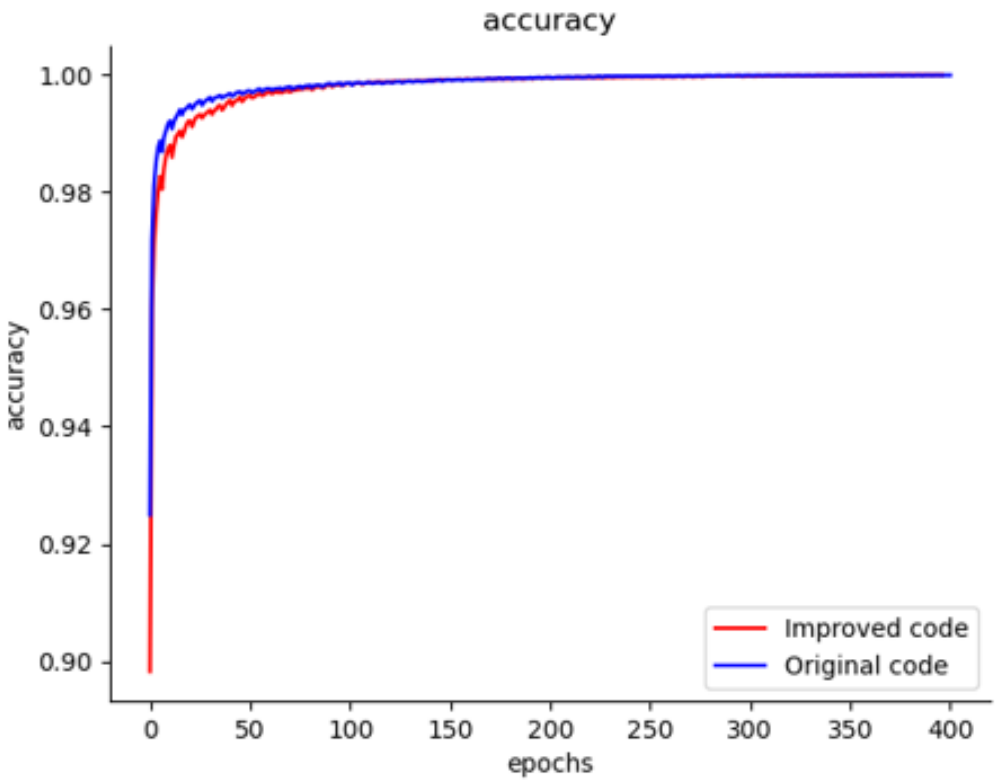

图 8: 代码的结果 acc 曲线

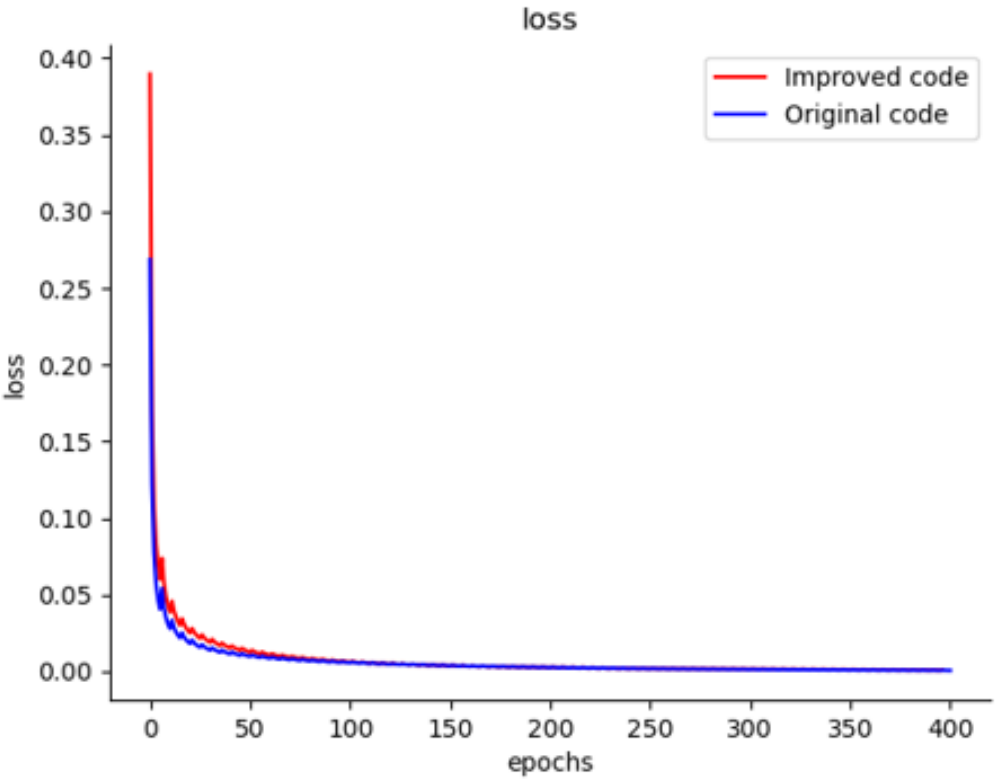

图 9: 代码的结果 loss 曲线

可以看出，改进之后的代码在收敛速度上更慢了，究其原因是我们选择使用奇异值分解压缩的层数增加了，即每次交换的数据量变少了，收敛所需要达到的通信轮次自然也更多了。我们对代码的改

进主要在加快通信效率上，在实际的联邦学习应用上意义重大，但是本次试验，无论是客户端还是服务器都是在同一台设备的不同进程间完成的，故在此不能很好的表示其意义。

6 总结与展望

随着联邦学习这一领域的研究越来越深入，异构问题渐渐的暴露出来。本文很好的考虑到了模型异构的问题，而且还提出了解决这个问题的方法。但是，这也有两个主要问题：第一点，只考虑了模型异构，不考虑数据异构的问题；第二点，文中提出的方法只是简单的把两个异构的模型分开来处理，两个模型之间互不交流，就如同在是两个互不相关的任务一样。现如今面对模型异构的解决方法有知识蒸馏的方法，在模型之间只传递知识，而不是模型。这样即使模型异构，模型之间也可以进行交流和传递知识。随着联邦学习的逐步发展，其范式的缺点也会逐步被解决，可以得心应手的面对大数据时代下的数据孤岛困境，可以被广泛的运用于我们的生活中。

参考文献

[1] ONG A, YS; Gupta. AIR(5): Five Pillars of Artificial Intelligence Research[J]. IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, 2019, 3: 411-415.

[2] Y.Shi, C.-T.Lin, Y.-C.Chang, et al. Consensus learning for distributed fuzzy neural network in Big Data environment[J]. IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, 2021, 5: 29-41.

[3] M.Asim, Y.Wang, K.Wang, et al. A review on computational intelligence techniques in cloud and edge computing[J]. IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, 2020, 4: 742-763.

[4] H.B.McMahan, E.Moore, D.Ramage, et al. Communication-efficient learning of deep networks from decentralized data[J]. Proceedings of the 20 th International Conference on Artificial Intelligence and Statistics, 2017, 54: 1273-1282.

[5] J. KONCY B, D.Ramage. Federated optimization: Distributed optimization beyond the datacenter[J]. Optimization for Machine Learning, 2016, [Online].

[6] Y.Chen, X.Sun, Y.Jin. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation[J]. IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, 2020, 31: 4229-4238.

[7] Et AL. K B. Practical secure aggregation for privacy-preserving machine learning[J]. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017: 1175-1191.

[8] Y.Chen, X.Sun, Y.Hu. Federated learning assisted interactive EDA with dual probabilistic models for personalized search[J]. Part of the Lecture Notes in Computer Science book series, 2019, 11655: 374-383.

[9] Et AL. P. Advances and open problems in federated learning[J]. Foundations and Trends in Machine

Learning, 2021, 14: 1-210.

[10] Y.Deng. Deep learning on mobile devices: A review[J]. SPIE Defense, Commercial Sensing, 2019, 10993: 52-66.

[11] H.Zhu, Y.Jin. Multi-objective evolutionary federated learning[J]. IEEE Transactions on Neural Networks and Learning Systems, 2020, 31: 1310-1322.

[12] H.Zhu, Y.Jin. Real-time federated evolutionary neural architecture search[J]. IEEE Transactions on Evolutionary Computation, 2022, 26: 364-378.

[13] Et AL Y. Communication efficient federated learning with adaptive quantization[J]. ACM Transactions on Intelligent Systems and Technology, 2022, 13: 1-26.

[14] G.Yan, S.-L.Huang, T.Lan, et al. DQSGD: Dynamic quantized stochastic gradient descent for communication efficient distributed learning[J]. IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems, 2021: 136-144.

[15] M.Yurochkin, M.Agarwal, S.Ghosh, et al. Bayesian nonparametric federated learning of neural networks [J]. Proceedings of the 36th International Conference on Machine Learning, 2019, 97: 7252-7261.

[16] Et AL. M A. Deep learning with differential privacy[J]. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016: 308-318.

[17] X.Peng, Z.Huang, Y.Zhu, et al. Federated adversarial domain adaptation[J]. International Conference on Learning Representations, 2020: 1-19.

[18] J.R.Bunch, J.E.Hopcroft. Triangular factorization and inversion by fast matrix multiplication[J]. American Mathematical Society, 1974, 28: 231-236.

[19] E.Anderson, Z.Bai, J.Dongarra. Generalized QR factorization and its applications[J]. Linear Algebra and its Applications, 1992, 162: 243-271.

[20] O.Schenk, K.Gärtner. Solving unsymmetric sparse systems of linear equations with pardiso[J]. Future Generation Computer Systems, 2004, 20: 475-487.

[21] J.Ye, Q.Li. A two-stage linear discriminant analysis via QR-decomposition[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005, 27: 929-941.

[22] S.Wold, K.Esbensen, P.Geladi. Principal component analysis[J]. Chemometrics and Intelligent Laboratory Systems, 1987, 2: 37-52.

[23] N.K.Kumar, J.Schneider. Literature survey on low rank approximation of matrices[J]. Linear Multilinear Algebra, 2017, 65: 2212-2244.

[24] H.Kim, M.U.K.Khan, C.-M.Kyung. Efficient neural network compression[J]. Conference on Computer Vision and Pattern Recognition, 2019: 12569-12577.

[25] Y.Idelbayev, M.A.Carreira-Perpinán. Low-rank compression of neural nets: Learning the rank of each layer[J]. Conference on Computer Vision and Pattern Recognition, 2020: 8049-8059.

[26] K.Goetschalckx, B.Moons, P.Wambacq, et al. Efficiently combining SVD, pruning, clustering and retraining for enhanced neural network compression[J]. Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning, 2018: 1-6.

[27] H.Zhou, J.Cheng, X.Wang, et al. Low rank communication for federated learning[J]. International Conference on Database Systems for Advanced Applications, 2020: 1-16.

[28] K.Greff, R.K.Srivastava, J.Koutník, et al. LSTM: A search space odyssey[J]. IEEE Transactions on Neural Networks and Learning Systems, 2017, 28: 2222-2232.

[29] F.Gers, J.Schmidhuber, F.Cummins. Learning to forget: Continual prediction with LSTM[J]. 1999 Ninth International Conference on Artificial Neural Networks ICANN 99, 1999: 850-855.