**Multi-Task Federated Learning for Personalised Deep Neural Networks in Edge Computing**

摘要：联邦学习(FL)是一种新兴的方法，用于在移动设备上协作训练深度神经网络(dnn)，而不会离开设备的私人用户数据。以往的研究表明，非独立同分布(non-IID)用户数据会影响FL算法的收敛速度。此外，大多数关于FL的现有工作度量全局模型精度，但在许多情况下，例如用户内容推荐，提高单个用户模型精度(UA)才是真正的目标。为了解决这些问题，我们提出了一种多任务深度神经网络(MTFL)算法，该算法将非联邦批处理归一化(BN)层引入联邦深度神经网络。MTFL通过允许用户根据自己的数据训练个性化模型，从而有利于UA和收敛速度。MTFL与流行的迭代FL优化算法如联邦平均(fedag)兼容，并且我们的经验表明，当作为MTFL中的优化策略使用时，分布式形式的亚当优化(fedag -Adam)可以进一步提高收敛速度。使用MNIST和CIFAR10的实验表明，MTFL能够显着减少达到目标UA所需的回合数，使用现有FL优化策略时最多可减少5倍，使用fedag - adam时可进一步减少3倍。我们将MTFL与竞争的个性化FL算法进行比较，表明它能够在所有考虑的场景中实现MNIST和CIFAR10的最佳UA。最后，我们在边缘计算测试平台上使用fedag - adam对MTFL进行了评估，结果表明其收敛性和UA优势超过了其开销。

1. **背景**

多访问边缘计算(Multi-access Edge Computing, MEC)将云服务移动到网络边缘，通过内容缓存和计算卸载实现应用程序的低延迟和实时处理。再加上智能手机、物联网(IoT)设备和社交网络(SNs)收集的数据量迅速增加，MEC提供了在边缘存储和处理大量数据的机会，接近它们的来源。

用于机器学习(ML)的深度神经网络(dnn)因其广泛的潜在应用、易于部署和最先进的性能而越来越受欢迎。然而，在监督学习中训练DNN的计算成本很高，并且需要大量的训练数据，特别是随着DNN规模的增加。在MEC中使用dnn通常涉及从移动电话/物联网设备/SNs收集数据，在云中执行训练，然后在边缘部署模型。然而，对数据隐私的担忧意味着，用户越来越不愿意上传他们可能敏感的数据，这就提出了如何训练这些模型的问题。

联邦学习(FL)为边缘机器学习开辟了新的视野。在FL中，参与的客户协同训练ML模型(通常是dnn)，而不泄露他们的私人数据。McMahan等人[5]发表了使用联邦平均(FedAvg)算法对FL进行的初步调查。fedag的工作原理是在协调服务器上初始化模型，然后将该模型分发给客户端。这些客户端在他们的本地数据集上执行一轮训练，并将他们的新模型推送到服务器。在将新的聚合模型发送给客户端进行下一轮训练之前，服务器将这些模型平均在一起。我们指的是人/机构/等等。作为“用户”拥有FL的数据，而作为“客户”实际参与FL的设备。

在无法上传数据以保护客户隐私的情况下，FL是一种非常有前途的分布式ML方法。因此，FL非常适合现实场景，如分析敏感的医疗数据、移动键盘上的下一个单词预测[8]和内容推荐[9]。然而，FL提出了多种独特的挑战:

客户端通常没有独立和同分布(IID)的训练数据。每个客户端都有自己生成的数据，可能有噪声数据，也可能只有所有特征/标签的子集。这些因素都会严重阻碍FL模型的训练。

FL研究通常在集中测试集上使用全局模型精度的性能度量。然而，在许多情况下，客户的个人模型准确性才是真正的目标——激励“个性化FL”，为FL客户创建独特的模型，以提高本地性能。然而，将个性化融入FL的最佳方式仍然是一个研究不足的话题。

由于客户端数据集的非iid性质，全局FL模型的性能在某些客户端上可能比其他客户端更高。这甚至可能导致一些客户收到比他们独立训练的模型更差的模型。

本文通过提出一种多任务识别算法(MTFL)来解决上述挑战，该算法允许客户训练个性化的dnn，既提高了局部模型的准确性，又有助于进一步增强客户的隐私。与其他个性化FL算法相比，MTFL具有更低的个性化存储成本和更低的计算成本(不需要在训练循环或个性化时对客户端进行额外的SGD步骤)。

由于FL中的客户端数据集通常是非iid的，因此客户端可以被视为在针对不同任务的本地训练期间试图优化其模型。我们的MTFL方法采用通常合并到DNN架构中的批处理规范化(BN)层，并使它们对每个客户端保持私有。Mudrarkarta等人先前表明，在集中设置下，私有BN层提高了ImageNet/Places-365上联合训练的多任务学习(MTL)性能。

使用私有BN层具有双重好处，可以根据客户端的本地数据对每个模型进行个性化设置，同时有助于保护数据隐私:由于客户端模型的一些参数没有上传到服务器，因此可以从上传的模型中收集到的关于客户端数据分布的信息较少。与其他个性化深度神经网络算法相比，我们使用BN层的MTFL方法也具有存储成本优势:与竞争算法的整个个性化深度神经网络模型相比，BN层通常只包含DNN总参数的一小部分，并且在FL轮之间只需要存储这些BN参数。

MTFL在典型的迭代FL框架之上添加了个性化。fedag和其他流行的算法就是这种迭代优化框架的实例。

1. **相关工作**

由于这项工作解决了现有FL算法的几个挑战，我们概述了FL的三个子主题的相关工作:考虑个性化的工作，处理实际和部署挑战的工作，以及旨在提高收敛速度和全局模型性能的工作。

2.1个性化联邦学习

一些作者考虑了“个性化”FL模型的方法，以便为非用户数据集定制模型性能。

元学习的目标是训练一个易于用少量样本进行微调的模型。Fallah等人提出了基于模型不可知论元学习(Model Agnostic Meta-Learning, MAML)的perfedag算法，该算法在客户端损失函数中增加了一阶自适应项，因此它们可以一步调整到客户端数据集。Jiang等人强调了fedag与一阶MAML更新之间的联系，并提出了一种三阶段训练算法来提高个性化。

其他作者建议在FL中结合本地和全球模型进行培训，以提高个性化。Hanzely和Richtárick增加了一个可学习的参数，允许客户控制局部和全局模型混合的程度。Dinh等人为每个用户保留了一个全局模型和一个个人模型，在他们的个人模型上执行SGD，然后在外循环中更新他们的全局模型副本。Huang等人[19]在每个客户端上保留了一个本地模型，并在客户端损失函数中添加了一个近似值项，以使这些模型在跨筒仓FL设置中接近“个性化”云模型。

Smith等人提出了MOCHA，它执行Federated MTL，将FL表述为模型权重矩阵和关系矩阵的函数。他们的算法考虑了客户端的异构硬件，这意味着MOCHA不能直接与我们的MTFL方案进行比较。最近，Dinh等人将MOCHA和其他算法推广到FedU框架中，包括提出了一个去中心化的版本。

我们的工作提出了一种多任务学习方法来实现外语个性化(MTFL)。我们随后证明，与现有的个性化FL算法相比，我们的方法具有显著的收敛速度、个性化性能、隐私和存储成本优势。

2.2边缘计算中的联邦学习

FL在网络边缘执行分布式计算。一些作者考虑了在这种环境下FL的系统设计和通信成本。Jiang等人提出了一种FL系统，通过选择梯度值最大的模型权重来减少客户端上传的数据总量。他们还考虑了实现细节，如异步或循环客户端更新。Bonawitz等人提出了fedag系统设计，指定了客户端/服务器角色、故障处理和安全性。他们还为拥有超过1000万客户的系统部署提供分析。为了解决FL中客户端数据集的非iid性质，Duan等人提出了Astrea框架:增强客户端数据集以帮助减少局部类失衡，并在全局聚合方法中引入中介。

一些作者也研究了无线连接的FL客户机的影响。Yang等研究了无线FL场景下的不同调度策略。他们的分析表明，在低信噪比(SINR)下，简单的FL方案表现良好，但随着SINR的增加，需要更智能的选择客户端的方法。Ahn等人提出了一种带有无线边缘设备的FL混合联邦蒸馏方案，包括使用空中计算和压缩方法。结果表明，该方案在高噪声无线场景下具有更好的性能。

其他作者提出了在边缘计算中考虑FL客户端计算、网络和通信资源的方案。Wang等人在智能手机上进行了实验，认为fedag的计算时间(而不是通信时间)是现实世界FL的最重要瓶颈，并提出了适应这种计算异质性的算法。Nishio和Yonetani设计了一个系统，该系统在启动一轮FL之前收集客户端的计算和无线资源信息，从而减少了达到fedag目标精度所需的实时时间。

这些先前的工作已经提出了FL系统的实现方案。然而，他们没有考虑FL中的MTL，这是我们对MTFL算法的主要贡献。

2.3联邦学习性能

开创性的FedAvg算法通过将初始模型发送给参与的客户端来协作训练模型，每个客户端使用他们的本地数据对模型执行SGD。这些新模型被发送到服务器进行平均，并开始新一轮。在提高fedag的收敛速度方面取得了一些进展。Leroy等人在更新全局时使用了Adam自适应优化

1. **采用方法**

**Multi-Task Federated Learning (MTFL)**

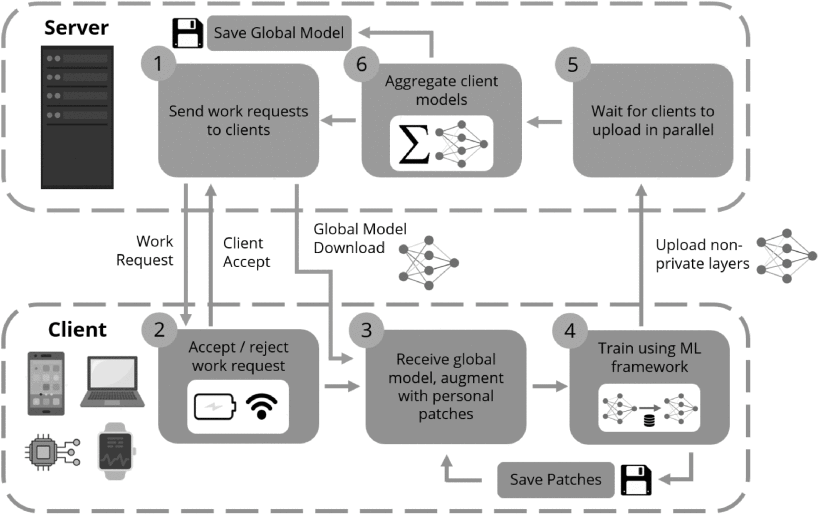


图1显示了MTFL算法如何在边缘计算环境中运行的高级概述。MTFL算法基于客户机-服务器框架，但轮由服务器发起，如图1所示。首先，服务器从其数据库中选择所有已知客户端或所有已知客户端的一个子集，并请求它们参与FL轮(步骤1)，并向它们发送一个Work Request消息。客户端将根据用户偏好接受工作请求(例如，用户可以将其设备设置为仅在充电并连接WiFi时参与FL)。然后，所有接受的客户端都向服务器发送一个Accept消息(步骤2)。服务器将全局模型(以及任何相关的优化参数)发送给所有接受的客户端，这些客户端使用私有补丁(步骤3)增强其全局模型的副本。然后，客户端使用自己的数据执行本地训练，创建不同的模型。客户端将新模型中的补丁层保存在本地，并将其非私有模型参数上传到服务器(步骤4)。

服务器等待客户端完成训练并上传他们的模型(步骤5)。它可以等待最大时间限制，也可以等待给定比例的客户端在继续上传，这取决于服务器的首选项。在此之后，服务器将聚合所有接收到的模型以生成保存在服务器上的单个全局模型(步骤6)，然后再开始新一轮。

因此，MTFL将绝大部分计算任务转移给客户端设备，由客户端设备执行实际的模型训练。它比fedag和其他个性化fl算法更能保护用户的数据隐私:不仅用户数据不会被上传，而且他们本地模型的关键部分也不会被上传。该框架还通过其循环时间/上传客户端分数限制来考虑客户端掉队者。此外，MTFL利用补丁层来提高个体用户非id数据集上的局部模型性能，使MTFL更加个性化。

3.1用户模型精度和MTFL

在许多FL作品中，例如FedAvg的原始论文[5]，作者使用一个中央IID测试集来衡量FL的性能。根据FL场景，这个度量可能是可取的，也可能不是可取的。如果目的是创建一个在IID数据上具有良好性能的单一模型，那么这种方法将是合适的。然而，在许多FL场景中，希望创建一个在单个用户设备上具有良好性能的模型。例如，Google在其GBoard下一个单词预测软件中使用了fedag[8]。目标是提高单个用户的预测分数。由于用户通常没有非iid数据，单个全局模型可能对某些用户显示良好的性能，而对其他用户显示较差的性能。

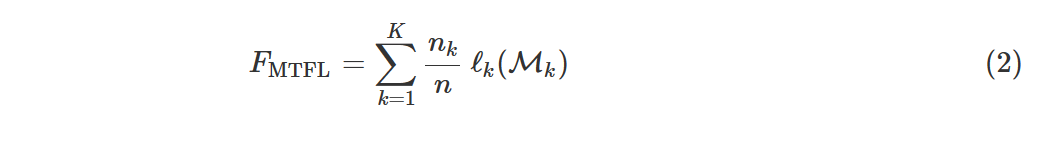
我们建议使用平均用户模型精度(UA)作为FL性能的替代度量。UA是使用本地测试集的客户机上的精度。每个客户端的测试集应该从与其训练数据相似的分布中提取。在本文中，我们对分类问题进行了实验，但是UA可以针对不同的度量(例如，错误，召回)进行更改。

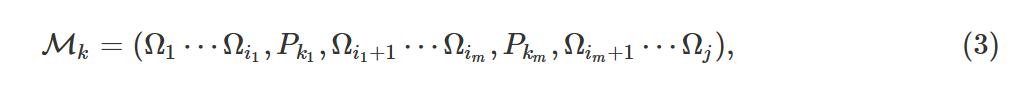
在FL中，用户数据通常是非iid的，因此可以认为用户具有不同但相关的学习任务。FL方案有可能实现良好的全局模型精度，但UA很差，因为聚合模型在一些客户端的数据集上可能表现不佳(特别是如果它们有少量的局部样本，那么在fedag平均步骤中加权较少)。我们提出了MTFL算法，允许客户端建立不同的模型，同时仍然受益于FL，以提高平均UA。Mudrakarta等人之前已经表明，向dnn添加小的每个任务的“补丁”层可以提高它们在MTL场景中的性能。因此，补丁是为客户培训个性化模型的好选择。

在FL中，目标是最小化以下目标函数:

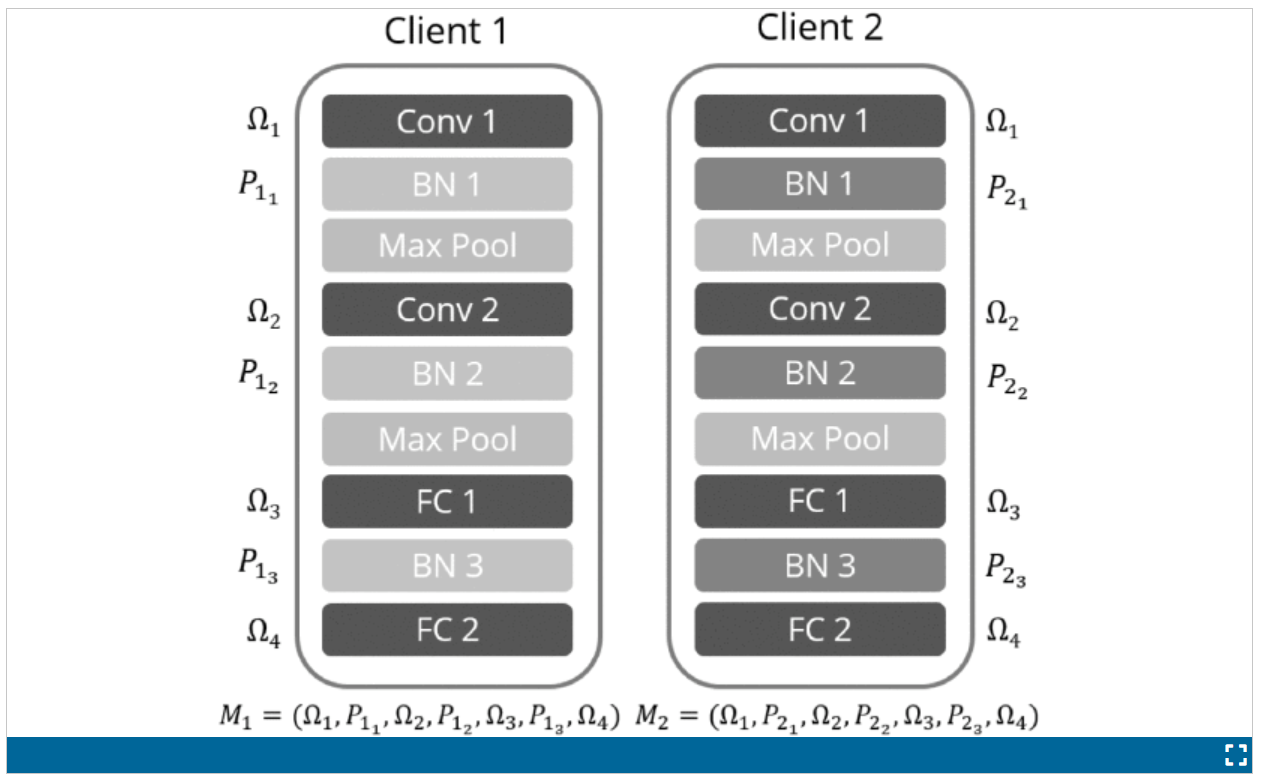


其中，K为客户端总数，nk为客户端K上的样本数，n为所有客户端的样本总数，K为客户端K上的损失函数，Ω为全局模型参数集。在FL模型中添加唯一的客户端补丁将MTFL的目标函数更改为





其中Mk是客户k上的补丁模型，由联邦模型参数Ω1⋯Ωj (j是联邦层的总数)和补丁参数Pk1⋯Pkm (m是局部补丁的总数，{i}是补丁参数的索引集)组成，这是客户k唯一的。图2显示了MTFL中使用的DNN模型的示例组成。



MTFL是一种将MTL整合到FL中的通用算法。MTFL中可以使用不同的优化策略(包括3.3节中描述的fedag - adam)。

Algorithm 1. MTFL

1: Initialise global model Ω and global optimiser values V

2: while termination criteria not met do

3: Select round clients, Sr⊂S, |Sr|=C⋅|S|

4: for each client sk∈Sr in parallel do

5:Download global parameters Mk←Ω

6: Download optimiser values Vk←V

7: for i∈ patchIdxs do ⊳ Apply local patches

8: Mk,i←Pk,i, Vk,i←Wk,i

9: end for

10: for batch b drawn from local data Dk do

11: Mk,Vk←LocalUpdate(Mk,Vk,b)

12: end for

13: for i∈ patchIdxs do ⊳ Save local patches

14: Pk,i←Mk,i,Wk,i←Vk,i

15: end for

16: for each i∉ patchIdxs do

17: Upload Mk,i,Vk,i to server

18: end for

19: end for

20: for i∉ nonPatchIndexes do

21: Ωi←GlobalModelUpdate(Ωi,{Mk,i}k∈Sr)

22:Vi←GlobalOptimUpdate(Vi,{Vk,i}k∈Sr)

23:end for

24: end while

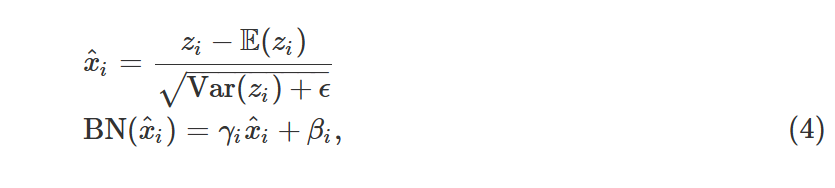
如算法1所示，MTFL运行几轮通信，直到满足给定的终止标准(如目标UA)(第2行)。在每轮通信中，从所有客户端S(第3行)中选择客户端的子集Sr参与。这些客户端下载全局模型Ω，这是一个模型参数的元组，如果使用全局优化器V(第5-6行)。然后客户端用他们的私有补丁层更新他们的全局模型和优化器的副本(第7-9行)，其中' patchIdxs '变量包含DNN中补丁层放置的索引。客户端在本地数据上使用他们现在个性化的全局模型副本和优化器执行训练(第10行)。根据在MTFL中使用的FL优化策略的选择，LocalUpdate函数表示模型的本地训练。对于fedag, LocalUpdate只是minibatch-SGD。我们将在3.3节进一步讨论这一点，以及提出的fedag - adam优化策略。本地训练完成后，保存更新后的本地补丁(第11-13行)，将非补丁层和优化器值上传到服务器(第14-16行)。

在一轮结束时，服务器根据GlobalModelUpdate和GlobalOptimUpdate函数生成一个新的全局模型和优化器(第18-20行)。这些函数同样依赖于所使用的FL优化策略，并在3.3节中进一步讨论。例如，fedag使用GlobalModelUpdate的客户端模型的加权平均值。更新后的全球模型标志着这一轮的结束，新一轮的开始。

MTFL的每轮总计算复杂度以|Sr|为尺度，其中|Sr|为每轮参与的客户端数量。每个客户机执行的计算与客户机总数无关。由于客户机并行执行本地计算，MTFL(与fedag一样)具有显著的可伸缩性。可扩展性在FL中很重要，因为实际部署中预计会有大量低功耗客户端。全局模型和优化器更新(算法1中的第20-23行)取决于所使用的优化策略。对于fedag和fedag - adam, GlobalModelUpdate本质上是map-reduce(局部训练后的平均)-也是O(|Sr|)。对于FedAdam, GlobalOptimUpdate中map-reduce之后的Adam步骤不依赖于客户端的数量(仅依赖于DNN架构)。

有许多研究在点对点(p2p)环境下研究FL的工作，我们在本文中没有考虑。简单的p2p FL算法包括将所有客户端模型发送给所有参与的对等端进行分散聚合。将MTFL扩展到这些方案是微不足道的:对等体只需发送/聚合非私有层。更复杂的p2p FL算法可能需要更复杂的方法来整合私有层——这是我们为未来的工作留下的一个有趣的方向。

Mudrakarta等人表明，批处理归一化(Batch normalization, BN)层可以作为集中设置下MTL的模型补丁。我们稍后表明，BN层在MTFL中作为补丁工作得很好，考虑到它们在参数数量方面非常轻量。BN层由



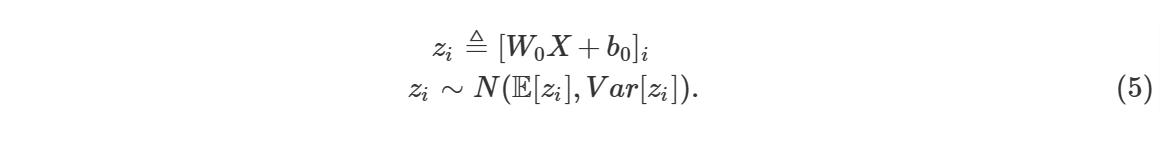
其中E(zi)和Var(zi)是神经元在一个小批量中的激活(zi，后非线性)的均值和方差，γi和βi是在训练过程中学习到的参数。BN层在训练期间跟踪E(zi)和Var(zi)的加权移动平均值:μi和σ2i，用于推理时间。在第4节中，我们研究了保留统计量μ，σ和/或可训练参数γ，β作为私有补丁层的一部分的好处。

我们选择在MTFL中使用BN层进行个性化。这种选择的原因是双重的:1)它们表现出出色的个性化性能，2)BN参数的存储成本非常小(小于测试模型架构的总模型尺寸的1%)。Mudrakarta等人也研究了深度卷积补丁在集中式多任务学习中的使用。原则上，在MTFL期间，任何模型层都可以保持私有，然而，在保持私有的参数数量与全局模型收敛能力之间存在固有的权衡。

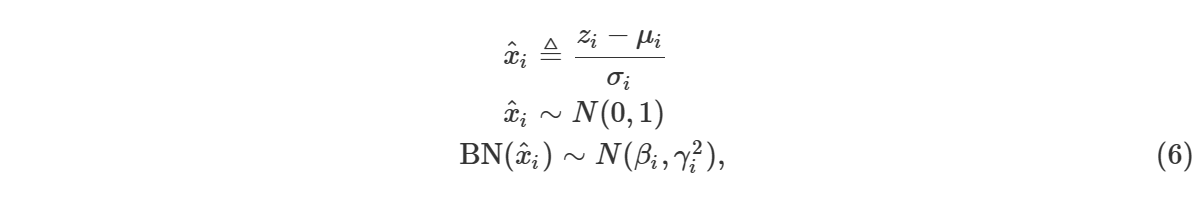
* 1. BN补丁对推理的影响

为了理解BN-patch层对UA的影响，我们考虑了在FL聚合步骤之前和之后客户端本地测试集上内部DNN激活的变化。

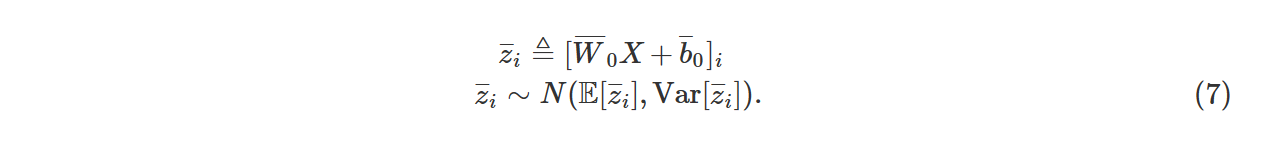
如图3a所示，在迭代FL中，UA通常在聚合步骤之后下降。这是因为模型已经在局部训练集上进行了几个epoch的调整，并且突然将其模型权值替换为联邦权值，而联邦权值不太可能比预聚合模型具有更好的测试性能。这个想法在[32]中得到了进一步的检验，并在后面的实验部分中得到了展示。考虑一个简单的深度神经网络，它由密集层组成，然后是BN，然后是非线性。通过应用权重和偏置(W0, b0)，客户测试集(X)上的第一层神经元激活向量可以建模为正态分布，BN依赖于其工作



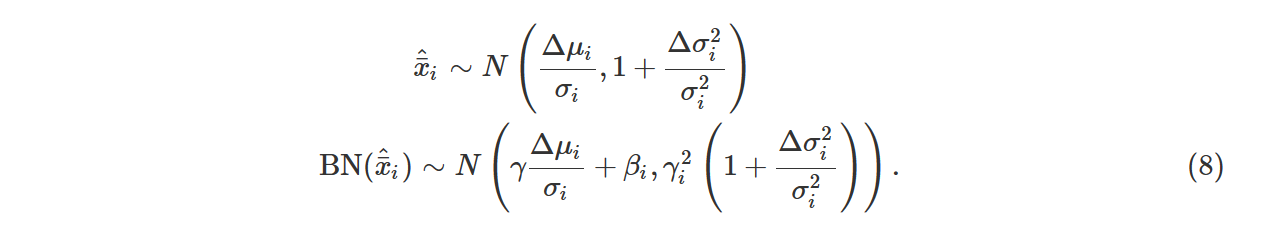
在局部训练过程中，客户端的模型已经适应了本地数据集，用于推理的bn层统计数据(μ， σ2)已经从层激活中更新。假设局部训练后(聚集前)μi≈E[zi]， σ2i≈Var[zi]，则bn层(忽略λ)计算



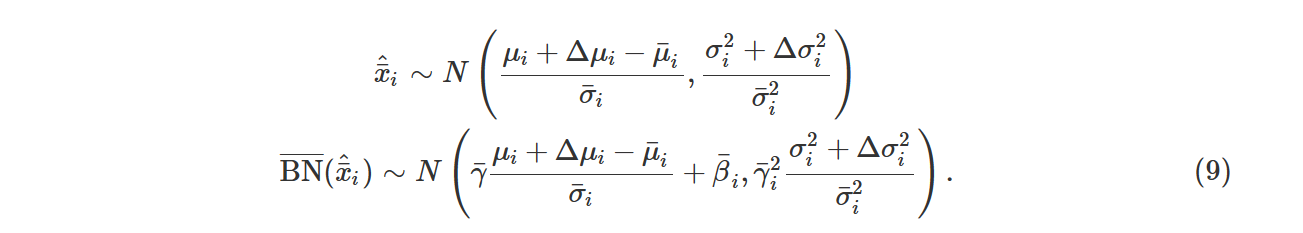
其中，βi， γ2i为学习到的BN参数。如果客户端正在参与FL或MTFL，则在下载具有联邦值W0,b0的全局模型后更新模型参数W0, b0。第一层的激活是



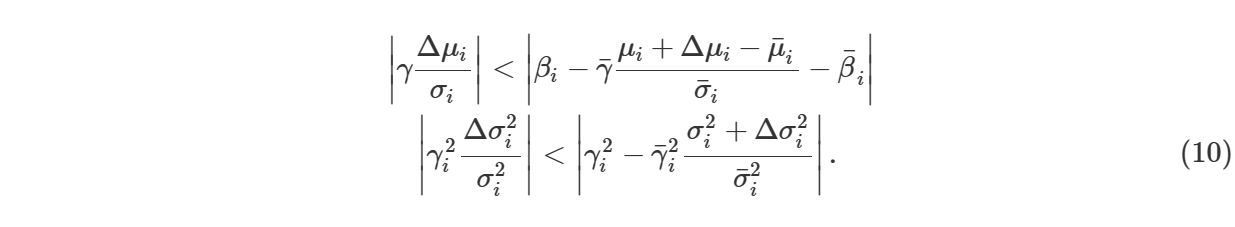
定义聚集前和聚集后激活之间的均值和方差差异，Δμi=E[zi]−E[zi]和Δσ2i=Var[zi]−Var[zi]， BN-patch层作为MTFL(聚集后保持μ， σ， β， γ)的一部分的输出为



如果BN层不是patch层(即客户端参与FL，其联合BN值为μ¯，σ¯，β¯，γ¯)，则BN层的输出为



我们假设在MTFL中使用BN-patch层限制神经元激活更接近于聚集步骤之前的状态，与作为FL的一部分的非patch BN层相比，使用MTFL在聚集前后的均值和方差的差异小于使用FL时



假设上述不等式成立，很容易看出当使用BN-patch而不是联合BN层时，在第一层之后通过网络传播的值是如何更接近预聚合值的。如果在整个网络中添加bn -patch，则中间DNN值将被定期“约束”，使其更接近预聚合值，最终导致网络输出更接近预聚合输出。

3.3 MTFL内部的联邦优化

如算法1所示，MTFL为每个客户机应用私有补丁层，并在LocalUpdate期间与联邦(非私有)层一起训练它们。在每一轮结束时，服务器聚合来自客户机的上传的联邦层(以及使用的任何分布式优化器值)，使用GlobalModelUpdate函数生成一个新的全局模型。如果使用分布式自适应优化，则还将调用GlobalOptimUpdate函数。表1详细介绍了不同的FL训练算法，以其实现这些函数为特征。

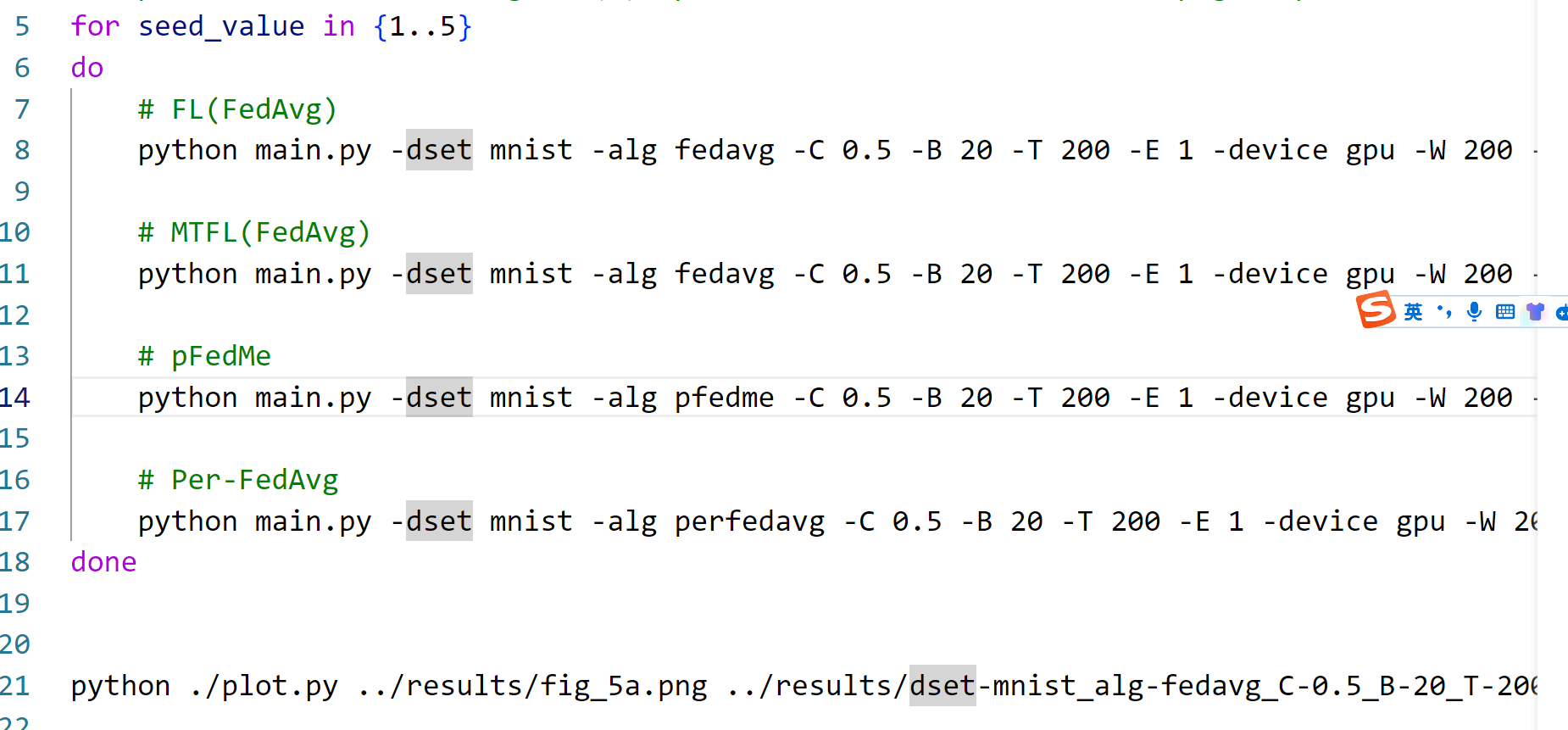
在fedag中，LocalUpdate只是minibatch-SGD，而GlobalModelUpdate生成新的全局模型，作为上传的客户端模型的加权(根据本地样本的数量)平均值。fedag使用没有自适应优化的SGD，因此算法1中的变量V是一个空值元组，GlobalOptimUpdate不执行任何功能。对于FedAdam，客户端也在LocalUpdate期间执行SGD。但是，在GlobalModelUpdate期间，服务器将获取之前的全局模型与平均上传的客户端模型之间的差异(Δr)。服务器将Δr视为“伪梯度”，并使用存储在服务器上的一组第一和第二矩值来使用类似于adam的更新步骤更新全局模型。客户端在FedAdam中不使用分布式自适应优化，因此V也是一个空值元组，GlobalOptimUpdate不执行任何功能。

我们建议使用自适应优化(即Adam)作为分布式优化策略。我们称这种策略为fedag - adam。在fedag -Adam中，客户端共享一个全局的Adam第一和第二矩集，存储在算法1的V变量中。客户端为他们的补丁层(Wk)存储私有优化器的值，因为我们发现当为补丁保留私有优化器的值时性能会更好。在LocalUpdate期间，客户端执行Adam SGD，联邦模型层和Adam值由客户端在一轮结束时上传。为了生成一个新的全局模型，服务器在GlobalModelUpdate中计算客户端模型的平均值，并在GlobalOptimUpdate中计算Adam矩的平均值。因此，与fedag或FedAdam相比，fedag - adam每轮的通信成本是fedag或FedAdam的3倍。然而，在许多FL场景中，主要关注的是减少模型收敛所需的通信轮数。我们随后证明fedag - adam显著提高了FL和MTFL的收敛速度。

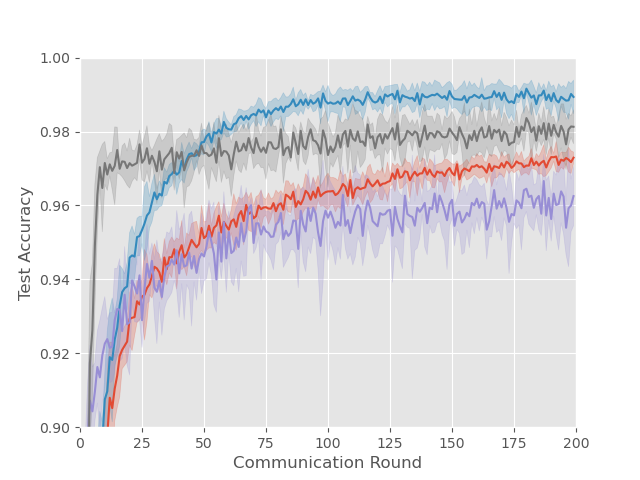
对于本文的其余部分，我们将引用迭代FL方案，该方案不保留任何私有模型补丁作为FL，其优化策略在括号中，例如FL(fedag)。如果客户端保留私有模型补丁，我们将该方案称为MTFL，再次使用括号中的优化策略，例如MTFL(fedag)。

1. 本地复现实验结果

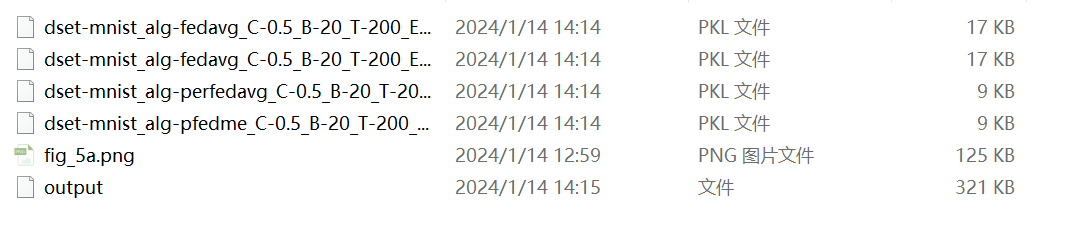
根据code/run中的代码，



我们一步步将其运行，最终即可以绘制出图像：



同时也能得到其他的文件



这样可以看出结果

Setup-代码将MTFL(FedAvg)的个性化性能与其他两种最先进的个性化FL算法进行比较:Per-FedAvg和pFedMe，以及FL(FedAvg)(其中没有模型层是私有的)。我们调整了每个算法的超参数，以实现200轮通信内的最大平均UA。我们用私有(γ，β)呈现MTFL(fedag)，而不是MTFL(fedag - adam)，因为我们只希望比较个性化算法，而不是自适应优化策略的好处。我们还将算法的局部计算量固定为大致不变:我们对MTFL(FedAvg)和FL(FedAvg)执行E=1 epoch的局部训练。对于MNIST，使用批量大小为20，这相当于W=200和W=400分别进行15步和8步的本地SGD。对于CIFAR10，这分别相当于W=200和W=400时本地SGD的13步和7步。per - fedag对局部迭代使用K值，因此我们将FL和MTFL的步数固定为相同的。pFedMe uses有两个内部循环，我们将外部循环的数量R设置为与per - fedag中的K相同的值，并将pFedMe的内部循环数量固定为1。这种设置导致为每个算法执行相同数量的本地步骤，但是，per - fedag和pFedMe的每个本地步骤的成本远远高于FL(fedag)和MTFL(fedag)。还要注意，MTFL(fedag)和FL(fedag)只有一个超参数η要调优，而per - fedag和pFedMe都有两个超参数。这使得对per - fedag和pFedMe进行超参数搜索的成本大大增加。

result-由实验得到的图显示，在所有测试场景中，与其他方案相比，MTFL(fedag)能够实现更高的UA。对于MNIST，在W=200的情况下，per - fedag和pFedMe能够达到比FL(fedag)更高的UA，但实际上在W=400的情况下更慢。然而，在CIFAR10实验中，所有个性化FL方案都能够比FL(fedag)更快地获得良好的UA。这可能是因为CIFAR10的任务比MNIST要困难得多。有趣的是，per - fedag在这个任务上似乎很快就过拟合了。同样值得注意的是，MTFL(fedag)能够击败per - fedag和pFedMe，同时也有一个更少的超参数需要调整，并且计算成本更低。MTFL还提供了保持一些模型参数私密性的额外好处(pFedMe和per - fedag都上传整个模型)。