《Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data》阅读报告

1. 摘要

联合学习允许多方在其组合数据上联合训练深度学习模型，而无需任何参与者将其本地数据透露给中央服务器。然而，这种形式的隐私保护协作学习是以培训期间的大量通信开销为代价的。为了解决这个问题，分布式训练文献中提出了几种压缩方法，可以将所需的通信量减少多达三个数量级。然而，这些现有方法在联合学习环境中的效用有限，因为它们要么只压缩从客户端到服务器的上游通信（不压缩下游通信），要么只在理想条件下运行良好，例如客户端数据的i.i.d.分布，这通常在联合学习中找不到。

在本文中，我们提出了稀疏三元组压缩（STC），这是一种新的压缩框架，专门为满足联邦学习环境的要求而设计。

STC扩展了现有的top-k梯度稀疏压缩技术，采用了一种新的机制来实现下游压缩以及权值更新的三元化和最优Golomb编码。

我们在四个不同学习任务上的实验表明，在常见的联邦学习场景中，STC明显优于联邦平均。这些结果支持联邦优化向高频低比特宽通信的范式转变，特别是在带宽受限的学习环境中

2. 介绍

目前有三项重大发展正在改变数据的创建和处理方式：

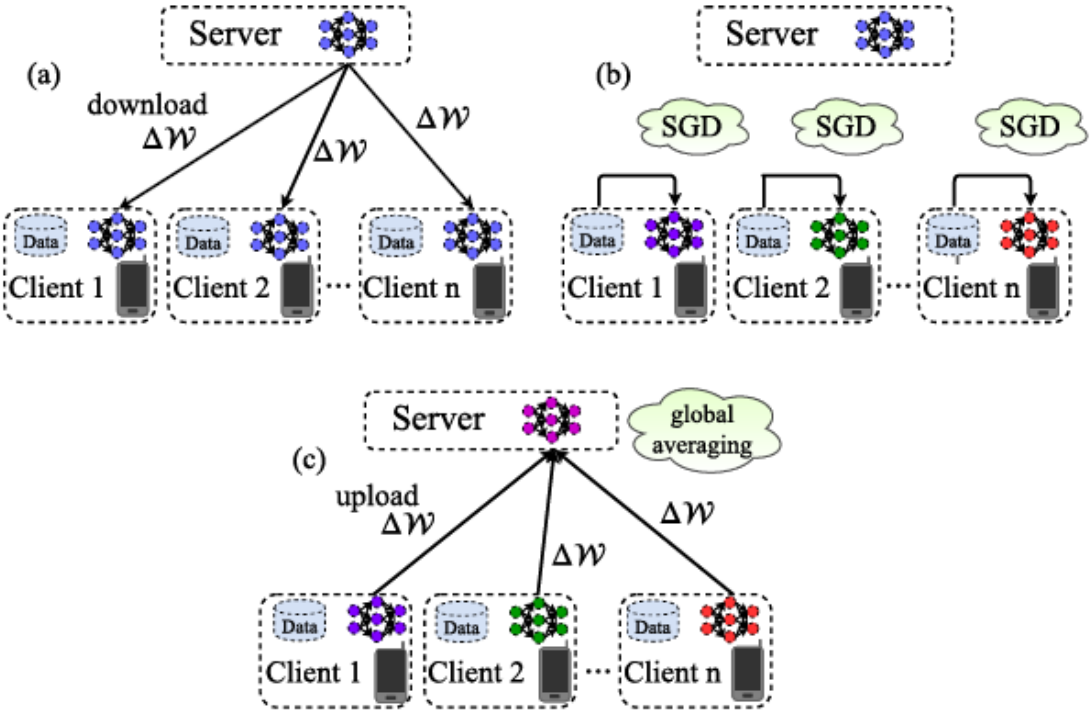
1）：随着物联网（IoT）的出现，世界上智能设备的数量在过去几年中迅速增长。其中许多设备都配备了各种传感器和日益强大的硬件，使它们能够以前所未有的规模收集和处理数据。

2）：在并行发展中，深度学习彻底改变了从数据资源中提取信息的方式，在计算机视觉、自然语言处理或语音识别等领域取得了突破性的成功。随着数据量的不断增长，深度学习的规模越来越大，其近年来取得的惊人成功至少可以部分归因于为培训提供了非常大的数据集。因此，利用物联网设备提供的丰富数据进行培训和改进深度学习模型具有巨大潜力。

3）：数据隐私已成为许多用户日益关注的问题。最近发生的多起数据泄漏和滥用案件表明，数据的集中处理对最终用户的隐私具有很高的风险。

我们面临以下困境：如果这些数据不能存储在集中的位置，我们如何利用数百万物联网设备的丰富组合数据来训练深度学习模型？

联邦学习解决了这个问题，因为它允许多方在其组合数据上联合训练深度学习模型，而无需任何参与者将其数据透露给中央服务器。联邦学习三个步骤：在第一步中，所有参与的客户端从服务器下载最新的主模型W。接下来，客户端使用随机梯度下降（SGD）基于其局部训练数据改进下载的模型。最后，所有参与的客户机都将其本地改进的模型W_i上传回服务器，在那里收集并聚合这些模型以形成新的主模型。这些步骤重复，直到满足某个收敛准则。



联合学习中的一个主要问题是发送模型更新所产生的巨大通信开销。当天真地遵循前面描述的协议时，每个参与的客户机都必须在每次培训迭代期间传达完整的模型更新。每一次这样的更新都与经过训练的模型大小相同，对于具有数百万个参数的现代体系结构，其大小可能在千兆字节范围内。在大数据集上几十万次的训练迭代过程中，每个客户机的总通信量可以轻松增长到1 PB以上。因此，如果**通信带宽有限或通信成本高昂，联合学习可能会变得毫无成效，甚至完全不可行。**

每个客户端在培训期间必须上传和下载的bits总量如下所示：

$$\mathrm{b}^{\mathrm{up/down}} \in \mathcal{O}(\underbrace{N_{\mathrm{iter}} \times f}_{\#\ \mathrm{updates}} \times \underbrace{|\mathcal{W}| \times (H(\Delta \mathcal{W}^{\mathrm{up/down}}) + \eta)}_{\mathrm{update\ size}})$$

N_{iter}是每个客户端执行的训练迭代总数，f是通信频率，$|W|$是模型的大小，$H(\triangle W^{up/down})$是上传和下载期间交换的权重更新的熵，η是编码的低效率

如果我们假设模型的大小和训练迭代次数是固定的（例如，因为我们希望在给定的任务上达到一定的精度），那么我们就有三种选择来减少通信：

1）我们可以减少通信频率f；

2） 通过有损压缩方案降低权重更新$H(\triangle W^{up/down})$的熵；

3）使用更有效的编码来传达权重更新，从而降低η。

3. 联邦学习环境的挑战

1）不平衡和非独立同分布数据：由于各个客户机上的培训数据是由客户机根据其本地环境和使用模式自行收集的，因此本地数据集的大小和分布在不同客户机之间通常会有很大差异。

2）大量客户：联合学习环境可能由数百万参与者组成。此外，由于协作学习模型的质量由所有客户的组合可用数据决定，协作学习环境将自然增长。

3）参数服务器：一旦客户端数量增长超过某个阈值，权重更新的直接通信就变得不可行，因为通信和聚合更新的工作负载都随客户端数量线性增长。因此，在联邦学习中，不可避免地要通过中间参数服务器进行通信。这将每个客户端的通信量和通信轮次减少为向服务器上载一个本地权重更新和从服务器下载一个聚合更新，并将聚合工作负载从客户端移开。然而，通过参数服务器进行通信给通信效率的分布式培训带来了另一个挑战，因为现在需要压缩服务器的上传和服务器的下载，以减少通信时间和能耗。

4）部分参与：在针对物联网的一般联合学习设置中，通常不能保证所有客户端都参与每一轮通信。设备可能会失去连接、电池耗尽或因其他原因为协作培训做出贡献。

5）电池和内存有限：移动和嵌入式设备通常未连接到电网。

基于上述联邦学习环境的特征，我们得出结论，用于联邦学习的通信效率高的分布式训练算法需要满足以下要求：

1）它应该压缩上传和下载通信。

2）它应该对非独立同分布、小批量和不平衡数据具有鲁棒性。

3）它应该对大量客户和部分客户参与保持稳健。

4. 贡献

我们表明，能够压缩上传和下载通信的方法对非独立同分布数据分布非常敏感，而对此类数据更稳健的方法不会压缩下游。然后，我们将继续为联邦学习构建一个新的高效通信协议，解决这些问题并满足所有要求（R1）–（R3）。我们对我们的方法进行了收敛性分析，并在四种不同的神经网络结构和数据集上进行了大量的实验，结果表明稀疏三值压缩（STC）协议优于现有的压缩方案，需要较少的梯度计算和通信比特来收敛到给定的目标精度。这些结果也扩展到了独立同分布数据制度。

5. 相关工作

我们可以将现有的大量传播效率分布式深度学习研究机构分为三个不同的小组：

1）通信延迟方法降低了通信频率f

2）稀疏化方法通过将更改限制在参数的一小部分来减少更新的熵$H(\triangle W)$。提出稀疏化方法主要是为了加速数据中心的并行训练。它们在更具挑战性的联邦学习环境中的收敛特性尚未得到研究。稀疏化方法（以其现有形式）主要压缩上游通信，因为来自不同客户端的更新的稀疏模式通常会有所不同。如果参与客户端的数量大于反向稀疏率（在联合学习中很容易出现这种情况），则下游更新甚至不会被压缩。

3）密集量化方法通过将所有更新限制为一组减少的值来减少权重更新的熵

W表示神经网络的全部参数，而W表示W内参数的一个特定张量，w表示网络的一个标量参数。神经网络参数之间的算术运算应被初步理解。

6. 现有压缩方法的限制

高效分布式深度学习的相关工作几乎只考虑客户之间的独立同分布数据，即他们假设局部梯度相对于完整批次梯度的无偏性，根据

$$\mathbb{E}_{x\sim p_i}[\nabla_{\mathcal{W}} l(x,\mathcal{W})] = \nabla_{\mathcal{W}} R(\mathcal{W}) \quad \forall i = 1,..,n$$

其中，p_i是第i个客户的数据分布，R（W）是组合培训数据的经验风险函数。在联邦学习环境中，我们通常只能希望平均值无偏

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{x^i\sim p_i}[\nabla_{\mathcal{W}} l(x^i,\mathcal{W})]=\nabla_{\mathcal{W}} R(\mathcal{W})$$

单个客户端梯度将偏向本地数据集：

$$\mathbb{E}_{x\sim p_i}[\nabla_{\mathcal{W}} l(x,\mathcal{W})]=\nabla_{\mathcal{W}} R_i(\mathcal{W})\neq\nabla_{\mathcal{W}} R(\mathcal{W})\quad \forall i=1,..,n.$$

7. 稀疏三元压缩（STC）

Top-k稀疏化在具有non-i.i.d客户端数据的分布式学习环境中显示出最有前途的性能。我们将以这一观察结果为出发点，为联邦学习构建一个有效的通信协议。为了达成这个协议，我们将解决三个公开的问题，这些问题阻止top-k稀疏化直接应用于联邦学习：

1）我们将通过对权值更新进行量化和最佳无损编码，进一步提高我们方法的效率。

2）我们将把下游压缩合并到该方法中，以实现从服务器到客户端的高效通信。

3）我们将实现一种缓存机制，以便在部分客户端参与的情况下保持客户端同步。

当稀疏化与非零元素的量化相结合时，可以实现更高的压缩增益。

Algorithm 1 STC

input: flattened tensor $T \in \mathbb{R}^n$, sparsity p
output: sparse ternary tensor $T^* \in \{-\mu, 0, \mu\}^n$
$k \leftarrow \max(np, 1)$
$v \leftarrow \text{top}_k(|T|)$
$\text{mask} \leftarrow (|T| \geq v) \in \{0, 1\}^n$
$T^{masked} \leftarrow \text{mask} \odot T$
$\mu \leftarrow \frac{1}{k}\sum_{i=1}^{n} |T_i^{masked}|$
return $T^* \leftarrow \mu \times \text{sign}(T^{masked})$

这个三元化步骤降低更新的熵通过：

$$H_{\text{sparse}} = -p\log_2(p) - (1-p)\log_2(p) + 32p$$

$$H_{\text{STC}} = -p\log_2(p) - (1-p)\log_2(p) + p$$

与常规疏化比较时，在稀疏率为p = 0.01的情况下，三元化获得的额外压缩Hsparse/HSTC = 4.414。为了通过纯粹的稀疏化获得相同的压缩增益，必须增加近似相同的稀疏率因子。

k-Contraction：

对于参数0 < k≤d, k压缩是一个满足压缩性质的算子comp: $R^d \to R^d$

$$\mathbb{E}\|x-\text{comp}(x)\|^2 \leq \left(1-\frac{k}{d}\right)\|x\|^2 \quad \forall x\in\mathbb{R}^d.$$

STC_k在算法1中定义是$\tilde{k} - contraction$

$$0 < \tilde{k} = \frac{\|\mathrm{top}_k(x)\|_1^2}{k\|x\|_2^2} d \leq d.$$

更新规则:

$$\mathcal{W}^{(t+1)} := \mathcal{W}^{(t)} - \mathrm{STC}_k\left(\mathcal{A}^{(t)} + \eta \Delta \mathcal{W}_{i_t}^{(t)}\right)$$
$$\mathcal{A}^{(t+1)} := \mathcal{A}^{(t)} + \Delta \mathcal{W}_{i_t}^{(t+1)} - \mathrm{STC}_k\left(\Delta \mathcal{W}_{i_t}{}^{(t+1)}\right)$$

收敛的根据:

$$\mathbb{E}[f(\overline{\mathcal{W}_T})] - f^* \leq \mathcal{O}\left(\frac{G^2}{\mu T}\right) + \mathcal{O}\left(\frac{\frac{d^2}{\tilde{k}^2} G^2 \frac{L}{\mu}}{\mu T^2}\right) + \mathcal{O}\left(\frac{\frac{d^3}{\tilde{k}^3} G^2}{\mu T^3}\right).$$

延伸至下游压缩，对于局部权值更新$W_i^{(t)}$, STC的更新规则可以写成:

$$\Delta \mathcal{W}^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\mathrm{STC}_k\left(\Delta \mathcal{W}_i^{(t+1)} + A_i^{(t)}\right)}_{\Delta \tilde{\mathcal{W}}_i{}^{(t+1)}}$$
$$A_i^{(t+1)} = A_i^{(t)} + \Delta \mathcal{W}_i^{(t+1)} - \Delta \tilde{\mathcal{W}}_i^{(t+1)}$$

在服务器端应用与客户端相同的压缩机制来压缩下游通信。这将更新规则修改为:

$$\Delta \tilde{\mathcal{W}}^{(t+1)} = \mathrm{STC}_k\left(\frac{1}{n} \sum_{i=1}^{n} \underbrace{\mathrm{STC}_k\left(\Delta \mathcal{W}_i^{(t+1)} + A_i^{(t)}\right)}_{\Delta \tilde{\mathcal{W}}_i{}^{(t+1)}} + A^{(t)}\right)$$

有客户端和服务器端残留更新:

$$A_i^{(t+1)} = A_i^{(t)} + \Delta \mathcal{W}_i^{(t+1)} - \Delta \tilde{\mathcal{W}}_i{}^{(t+1)}$$
$$A^{(t+1)} = A^{(t)} + \Delta \mathcal{W}^{(t+1)} - \Delta \tilde{\mathcal{W}}^{(t+1)}.$$

用于部分客户端参与的权重更新缓存，为了解决同步问题并减少客户端的工作负载，我们建议在服务器上使用缓存机制。假设最后一轮τ通信产生了更新:

$$\{\Delta \tilde{\mathcal{W}}^{(t)} | t = T-1, \ldots, T-\tau\}$$

服务器可以缓存这些更新的所有部分和，直到某一点:

$$\{P^{(s)} = \sum_{t=1}^{s} \Delta \tilde{\mathcal{W}}^{(T-t)} | s = 1, .., \tau\}$$

与全局模型一起:

$$\mathcal{W}^{(T)} = \mathcal{W}^{(T-\tau-1)} + \sum_{t=1}^{\tau} \Delta\tilde{\mathcal{W}}^{(T-t)}.$$

然后，每个想要参与下一轮通信的客户端都必须首先通过下载P（s）或W（T）与服务器同步，这取决于它跳过了多少次前一轮通信。对于一般稀疏更新，熵的界可以实现为：

$$H(P^{(\tau)}) \leq \tau H(P^{(1)}) = \tau H(\Delta\tilde{\mathcal{W}}^{(T-1)})$$

这意味着下载的大小将随着客户端跳过培训的轮数线性增长。跳过的平均轮数等于反向参与分数1 \/η。这通常是可以容忍的，因为下行链路通常比上行链路更便宜，带宽也远高于上行链路。本质上，所有只传递参数更新而不是完整模型的压缩方法都会遇到同样的问题。signSGD也是如此，尽管在这里，下游更新的大小仅随延迟时间的对数增长，根据

$$H(P_{signSGD}^{(\tau)}) \leq \log_2(2\tau + 1).$$

Lossless Encoding

Algorithm 2 Efficient Federated Learning With Parameter Server Via STC

input: initial parameters $\mathcal{W}$
output: improved parameters $\mathcal{W}$
init: all clients C_i, $i = 1, .., [\text{Number of Clients}]$ are initialized with the same parameters $\mathcal{W}_i \leftarrow \mathcal{W}$. Every Client holds a different data set D_i, with $|\{y : (x, y) \in D_i\}| = [\text{Classes per Client}]$ of size $|D_i| = \varphi_i |\cup_j D_j|$. The residuals are initialized to zero $\Delta\mathcal{W}, \mathcal{R}_i, \mathcal{R} \leftarrow 0$.
for $t = 1, .., T$ **do**
  for $i \in I_t \subseteq \{1, .., [\text{Number of Clients}]\}$ ***in parallel*** **do**
    Client C_i does:
    · $\text{msg} \leftarrow \text{download}_{S \rightarrow C_i}(\text{msg})$
    · $\Delta\mathcal{W} \leftarrow \text{decode}(\text{msg})$
    · $\mathcal{W}_i \leftarrow \mathcal{W}_i + \Delta\mathcal{W}$
    · $\Delta\mathcal{W}_i \leftarrow \mathcal{R}_i + \text{SGD}(\mathcal{W}_i, D_i, b) - \mathcal{W}_i$
    · $\Delta\tilde{\mathcal{W}}_i \leftarrow \text{STC}_{p_{up}}(\Delta\mathcal{W}_i)$
    · $\mathcal{R}_i \leftarrow \Delta\mathcal{W}_i - \Delta\tilde{\mathcal{W}}_i$
    · $\text{msg}_i \leftarrow \text{encode}(\Delta\tilde{\mathcal{W}}_i)$
    · $\text{upload}_{C_i \rightarrow S}(\text{msg}_i)$
  end
  Server S does:
  · $\text{gather}_{C_i \rightarrow S}(\Delta\tilde{\mathcal{W}}_i),\ i \in I_t$
  · $\Delta\mathcal{W} \leftarrow \mathcal{R} + \frac{1}{|I_t|} \sum_{i \in I_t} \Delta\tilde{\mathcal{W}}_i$
  · $\Delta\tilde{\mathcal{W}} \leftarrow \text{STC}_{p_{down}}(\Delta\mathcal{W})$
  · $\mathcal{R} \leftarrow \Delta\mathcal{W} - \Delta\tilde{\mathcal{W}}$
  · $\mathcal{W} \leftarrow \mathcal{W} + \Delta\tilde{\mathcal{W}}$
  · $\text{msg} \leftarrow \text{encode}(\Delta\tilde{\mathcal{W}})$
  · $\text{broadcast}_{S \rightarrow C_i}(\text{msg}),\ i = 1, .., M$
end
return $\mathcal{W}$