VerifyNet:安全、可验证的联邦学习

摘要:

联邦学习作为一种新兴的神经网络训练模型，由于其不需要收集用户原始数据就能更新参数而受到广泛关注。然而，由于攻击者可以从共享梯度中跟踪和提取参与者的隐私，所以联邦学习仍然面临各种安全和隐私威胁。在本文中，我们考虑了深度神经网络(DNNs)训练过程中的两个主要问题:(1).如何在训练过程中保护用户的隐私(即本地梯度)。(2).如何验证从服务器返回的聚合结果的完整性(或正确性)。为了解决上述问题，有人提出了几种关注安全或者隐私保护的联邦学习方法，并将其应用于不同的场景。然而，如何让客户端在训练过程中验证云服务器是否运行正常，同时保证用户的隐私，这仍然是一个有待解决的问题。在本文中，我们提出了$VerifyNet$(可验证网络)，这是第一个隐私保护和可验证的联邦学习框架。具体来说，我们首先提出了一个双重掩蔽协议，以保证在联邦学习过程中用户的本地梯度的机密性。然后，云服务器需要向每个用户提供关于其聚合结果正确性的证据。我们承诺，攻击者不可能通过伪造证据来欺骗用户，除非他能解决我们模型中采用的$NP-hard$问题。此外，$VerifyNet$还会为在训练过程中离线的用户提供支持。在实际数据上进行的大量实验也证明了我们提出的方案的实际性能。

索引术语:

隐私保护，深度学习，可验证的联邦学习，云计算。

1. 引言

深度学习在很多应用中扮演一个重要角色，如医疗预测[1][2]、自动驾驶[3][4]等。这种基于深度学习的应用已经渗透到我们社会的各个方面，并且逐渐改变人们在各个领域的习惯，比如生活、旅游和社交[5][6]。

深度学习需要大量的数据，这些数据通常来自于用户。然而，用户的数据可能很敏感或包含一些私人信息。例如，在医疗保健系统中，患者可能不愿意与第三方服务提供商(如云服务器)[7]-[9]共享自己的医疗数据。最近，由于联邦学习不需要收集用户的原始数据就能训练网络，所有用户和云服务器只需要共享本地梯度和全局参数就能一起工作，联邦学习[10][11]正逐渐受到学术界和业界的关注。但研究表明，攻击者仍然可以基于共享梯度间接获取标签[12][13]以及成员[14][15]等敏感信息。另一方面，媒体也经常报道联邦学习中的数据完整性遭到破坏[16][17]。特别是在某些非法利益的驱使下，恶意的云服务器可能会向用户返回不正确的结果。例如，一个“懒惰的”云服务器可能会用一个更简单但不准确的模型去压缩原始模型，以减少自己的计算成本，或者更糟，云服务器恶意伪造发送给用户的聚合结果。因此，保护用户隐私和数据完整性(特别是从服务器返回结果的正确性)是联邦学习训练过程中的两个基本问题。因此，设计一种安全的联合训练协议，它能够在保护用户数据隐私的同时，有效地验证服务器返回结果的正确性，这是一项迫切而有意义的任务。

为了解决上述问题，一些针对隐私保护的深度学习的研究工作已经被提出。$Shokri$等人

[18]提出了一种通过有选择地共享更新参数来保护隐私的深度学习协议，它可以在实用性和安全性之间实现平衡。$Trieu\ Phong$等人[19]将加法同态加密和梯度下降技术相结合，提出了一个安全的深度学习系统。最近，$Bonawitz$等人[11]利用秘密共享和密钥协议提出了一种实用安全的联邦学习架构，它允许用户在训练过程中离线，同时仍然保证高精确性。但是，上述解决方案都不支持验证服务器返回结果的正确性。服务器返回结果的正确性与用户本地梯度的机密性密切相关。一旦攻击者能够掌握返回给用户的数据，用户隐私受到损害的风险往往会增加。例如，在著名的白盒攻击[14][15]中，为了分析用户上传数据的统计特征，攻击者可以将精心制作的结果返回给用户，并诱导用户发布更多的敏感信息。

近年来，为了缓解训练良好的神经网络下的数据完整性问题，相继有人提出了[16][17]方案。然而，这些方案要么只支持少量的激活函数，要么需要额外的硬件辅助。据我们所知，在训练过程中，没有现有的解决方案支持神经网络的可验证性。与训练良好的神经网络相比，在训练过程中验证结果的正确性显然更加复杂，因为在预测结果的同时还要更新整个网络的参数。此外，如何在支持可验证性的同时，允许用户在训练流程中离线(由于网络不可靠、设备电池等问题)，并确保所有用户(包括离线的)的本地梯度的机密性也是一个挑战。

在本文中，我们提出了$VerifyNet$，这是第一个在神经网络训练过程中支持验证的隐私保护方法。我们首先设计了一种基于同态哈希函数和伪随机技术的可验证方法来支持每个用户的可验证性。然后，我们使用一种秘密共享技术和密钥协议结合的变种来保护用户本地梯度的隐私，并解决用户在训练过程中离线的问题。总而言之，我们的贡献可概括如下:

（1） 我们利用与伪随机技术集成的同态哈希函数作为$VerifyNet$的底层结构，这允许用户在可接受的开销下验证从服务器返回结果的正确性。

（2） 我们提出了一个双掩蔽协议，以保证在联邦学习过程中用户本地梯度的机密性。它可以承受一定数量的用户在训练过程中因某些原因离线，并且仍然对这些离线的用户提供隐私保护。

（3） 我们对我们的$VerifyNet$进行了全面的安全分析。我们承诺，即使云服务器与多个用户勾结，攻击者也不会获得任何有用的用户本地梯度信息。此外，对真实数据进行的大量实验也证明了我们的$VerifyNet$的实用性。

本文的其余部分结构如下。在第二节中，我们陈述了问题。在第三节和第四节中，我们描述了初步阶段，并给出了一个技术直觉来解释我们如何解决本文中考虑的挑战。在第五节中，我们描述了我们$VerifyNet$的技术细节。然后，我们在第六节进行安全性分析。之后，在第七节和第八节分别进行性能评估和对相关工作进行讨论。最后，第九节对本文进行总结。

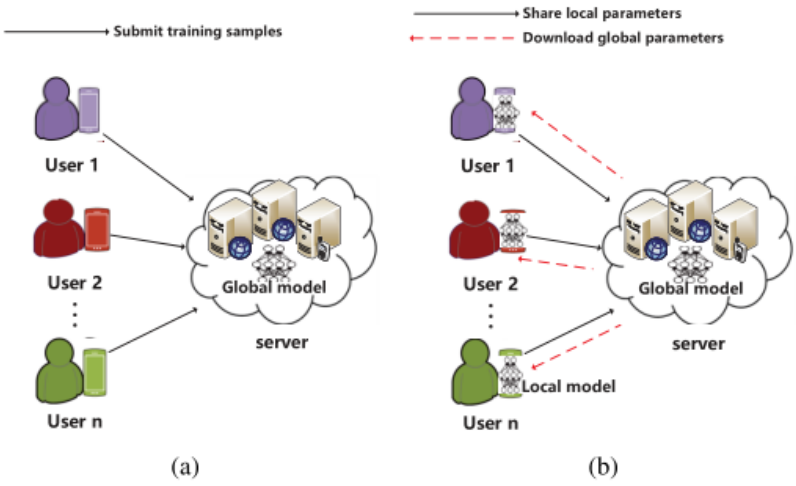
2. 问题陈述

在本节中，我们首先回顾联邦深度学习的主要概念。然后描述了系统架构、威胁模型和设计目标。

A．联邦深度学习

（1） 概述：根据训练方式的不同，深度学习可以分为以下两种类型。

a. 集中训练。如图$Fig.1(a)$所示，传统的集中训练从服务器要求用户将其本地数据(即训练样本)上传到云端开始。然后，服务器在云上初始化深度神经网络，用训练样本对其进行训练，直到得到最优模型。最后，云服务器将返回预测服务接口或返回最优参数给用户。

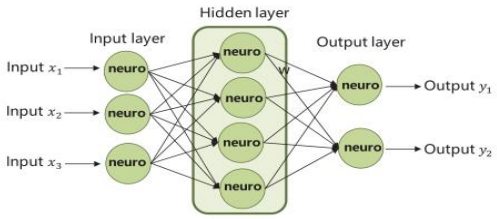


$Fig.1$ 集中训练和联邦训练的通用框架

(a)集中训练　　(b)联合训练

b. 联合训练。如上所述，用户直接将本地数据上传到服务器，可能会存在隐私泄露的风险。因此，与集中训练不同，在联合训练中(如图$Fig.1(b)$所示)，每个用户和服务器协同训练一个统一的神经网络模型。为了加速模型的收敛，每个用户与云服务器共享本地参数(即梯度)，云服务器聚合所有梯度并将结果返回给每个用户。最终，服务器和每个用户都会得到最优的网络参数。与集中训练相比，联合训练降低了用户隐私被侵犯的风险。但研究表明，攻击者仍然可以通过共享梯度间接获取敏感信息。此外，在某些非法利益的驱动下，恶意云服务器可能会向用户返回不正确的结果。因此，在本文中，我们在验证联合训练过程中从服务器返回结果的正确性的同时，重点关注用户本地梯度的隐私保护。

(2) 神经网络：作为深度学习的底层结构，神经网络可以与各种技术相结合，实现分类、预测和回归。如图$Fig.2$所示，有 3 个输入，1 个隐藏层和 2 个输出的全连接神经网络。全连接是指相邻两层之间的所有神经元都通过变量(本节称为ω)相互连接。一般来说，神经网络可以表示为函数$f\ (x,\ \omega)\ =\hat{y}$，其中$x$表示用户的输入，$\hat{y}$表示函数$f$带参数$\omega$对应的输出。



$Fig.2$　　全连接神经网络

(3) 联邦学习更新：在不丧失通用性的情况下，假设每个数据记录是一个观察对$<x,\ y>$，并且整个训练集$D\ =\{<x_i,\ y_i>,\ i\ =\ 1,\ 2,\ \cdots T\}$。在训练集上定义一个损失函数为

$$L_f(D,\ \omega)=\frac{1}{|D|}\sum_{<xi,\ yi>\in D}L_f\ (xi,\ yi,\ \omega)$$

其中$L_f(x,\ y,\ \omega) = l(y,\ f(x,\ \omega))$为一个特定的损失函数$l$。在本文中，损失函数设置为$l(y,\ f(x,\ \omega)) = l(y,\ \hat{y}) = ||y,\ \hat{y}||_2$，$||\cdot||_2$是一个向量的$l_2$范数。

训练神经网络的目标是找到最优参数ω，从而使损失函数的值最小。在我们的$VerifyNet$中，我们采用随机梯度下降[20][21]来完成这个任务。具体来说，每个参数的迭代计算如下：

$$\omega^{j+1} \leftarrow \omega^j - \lambda \nabla L_f(D^j,\ \omega^j)$$

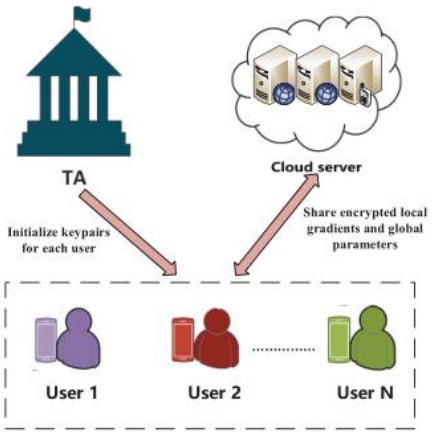
式中ω^j为第j次迭代后的参数。D^j是D的一个随机子集，而λ是学习率。在我们的联邦学习中，每个用户$n \in N$都拥有一个私有的本地数据集D_n，用预先与所有其他参与者商量好的某个神经网络训练本地数据集，其中$D = \sum_{n \in N} D_n$。具体来说，服务器在第j次迭代时选择一个随机子集$N^j \subseteq N$，然后每个用户$n \in N^j$随机选择一个子集$D_n^j \subseteq D_n$进行随机梯度下降。因此，参数更新可以重写为以下式子：

$$\omega^{j+1} \leftarrow \omega^j - \lambda \frac{\sum_{n \in N^j} p_n^j}{\sum_{n \in N^j} |D_n^j|}$$

其中$p_n^j = |D_n^j| \nabla L_f(D_n^j,\ \omega^j)$由各用户计算并共享到云服务器。然后，云服务器向所有用户返回全局参数ω^{j+1}。

B．系统架构

如图$Fig.3$所示，我们的系统模型由三个实体组成：可信权威(Trusted Authority——TA)、用户(User)和云服务器(Cloud)。



$Fig.3$　系统架构

(1)　可信权威：它的主要工作是初始化整个系统，生成公共参数，并为每个参与者分配公钥和私钥。之后，它将离线，直到出现争议才会再次上线。

(2)　用户：每个用户需要在每次迭代期间将其加密的本地梯度发送给云服务器。此外，用户还会发送一些其他加密的信息，为云服务器生成计算结果的证据做好准备。

(3)　云服务器：云服务器聚合所有在线用户上传的梯度，并将结果连同证明一起发

送给每个用户，我们要求云服务器只知道加密的梯度和最终结果。

C．威胁模型和设计目标

我们在我们的$VerifyNet$中定义了一个诚实但好奇[22]的威胁模型。具体来说，在我们的$VerifyNet$中，可信权威是值得信任的，不会与任何实体勾结。包括云服务器在内的所有其他参与方被认为是诚实而好奇的[11]，即云服务器和用户都会按照约定的协议执行程序，但也可能试图自己推断其他用户的数据隐私[12][14][23]。特别是我们允许云服务器与多个用户串通以获得最具进攻性的能力，也允许云服务器伪造证据(我们不允许串通伪造证据)并修改计算结果以欺骗用户。

我们的$VerifyNet$旨在保护用户本地梯度的机密性，同时对每个用户提供强验证性的支持，并允许用户在训练过程中离线。我们承诺，除非对手能够解决我们模型中采用的 NP-hard 问题，否则不可能实现欺骗行为。

3. 预处理

为了便于理解本文，我们引入了$VerifyNet$中使用的一些密码原语，这将使读者更容易理解我们的方法。

A．双线性配对

一个双线性配对可以表示为一个映射e: $G_1 \times G_2 \rightarrow G_I$，其中$G_1$和$G_2$都是具有相同素数阶$q$的乘法循环群。在不丧失通用性的情况下，我们假设$G_1$和$G_2$的生成器分别为$g$和$h$。通常，双线性配对$e$有以下性质:

（1） 双线性：给定随机数$a, b \in Z_q^*$，对于任意$g_1 \in G_1$和$g_2 \in G_2$，我们都能得出$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$。

（2） 可计算性：对于任意$g_1 \in G_1$和$g_2 \in G_2$，$e(g_1, g_2)$都可以有效计算。

（3） 非退化性：$e(g,h) \neq 1$，其中g和h分别是G_1和G_2的生成器。

B．同态哈希函数

通常，给定消息$x_i \in Z_q$，一个抗碰撞同态哈希函数[24][25] $HF: Z_q \rightarrow G_1 \times G_2$可表示为如下式子:

$$HF(x_i) = (A_i, B_i) = (g^{HF_{\delta, \rho}(x_i)}, h^{HF_{\delta, \rho}(x_i)})$$

其中δ和ρ都是在有限域Z_q中随机选取的秘钥。$HF_{\delta, \rho}()$是单向同态哈希函数。更准确地说，给定$HF(x_1) = (g^{HF_{\delta, \rho}(x_1)}, h^{HF_{\delta, \rho}(x_1)})$，$HF(x_2) = (g^{HF_{\delta, \rho}(x_2)}, h^{HF_{\delta, \rho}(x_2)})$，同态哈希函数有以下性质:

（1） 可加性(指数)可以表示为$HF(x_1 + x_2) \leftarrow (g^{HF_{\delta, \rho}(x_1)+HF_{\delta, \rho}(x_2)}, h^{HF_{\delta, \rho}(x_1)+HF_{\delta, \rho}(x_2)})$

（2） 乘以一个常数α可以表示为$HF(\alpha x_1) \leftarrow (g^{\alpha HF_{\delta, \rho}(x_1)}, h^{\alpha HF_{\delta, \rho}(x_1)})$。

同态哈希函数还有其他有趣的性质。有兴趣的读者可以参考[24][25]了解更多细节。

C．伪随机函数

我们在我们的$VerifyNet$中采用了 Dario Fiore 等人[25]设计的伪随机函数。通常，给定秘钥$K = (K_1, K_2)$，一个伪随机函数$PF_{K_2}: \{0,1\}^* \times \{0,1\}^* \rightarrow G_1 \times G_2$包含另外两个

伪随机函数，即$PF_{K_1}:\{0,1\}^* \rightarrow Z_q^2$和$PF_{K_2}:\{0,1\}^* \rightarrow Z_q^2$。给定输入$(I_1, I_2)$，我们能得到$PF_{K_1}(I_1) = (\gamma_{I_1}, \nu_{I_1})$和$PF_{K_2}(I_2) = (\gamma_{I_2}, \nu_{I_2})$。因此，我们得到

$$PF_K(I_1, I_2) = (\mathrm{E}, \mathrm{F}) = (g^{\gamma_{I_1}\gamma_{I_2}+\nu_{I_1}\nu_{I_2}}, h^{\gamma_{I_1}\gamma_{I_2}+\nu_{I_1}\nu_{I_2}})$$

作为验证的一部分，伪随机函数将被用来验证来自云服务器的结果的正确性。

D．秘密共享协议

在$VerifyNet$中，我们利用$Shamir$的$t-out-N$秘密共享协议[26]将秘密s分成N个单独的部分，其中N为我们模型中的用户数，t为阈值。这意味着任何大于t的共享子集都可以用来恢复秘密s。具体来说，实现这个秘密共享协议需要以下步骤:

(1) $S.share\ (s, t, U) \rightarrow \{(n, s_n)\}_{n\in U}$: 给定阈值$t \leq |U|$和秘密$s$，输出每个用户$n$的共享$s$中的$s_n$，其中$U$代表一组用户的$ID$(推测是独特的)中指定的一个有限域$F$，并且$|U| = N$。

(2) $S.recon(\{(n, s_n)\}_{n\in U}, t) \rightarrow s$: 输入一个共享子集$M$，其中$n \in M \subseteq U$,, $t \leq |M|$，输出秘密s。

E．密钥协议

我们的$VerifyNet$也采用了$Diffie-Hellman$密钥协议[27][28]来为任意两个用户创建共享密钥。具体来说，给定一个素数阶为q的群G，每个用户n的秘钥/公钥被创建为$KA.gen(G, g, q) \rightarrow (SK_n, g^{SK_n})$，其中$g$是组$G$的生成器，$SK_n$和$g^{SK_n}$分别是秘钥和公钥。然后，给定用户$m$的公钥$g^{SK_m}$，就可以生成用户$n$和用户$m$之间的共享密钥，即$KA.agree(SK_n, g^{SK_m}) \rightarrow s_{n,m}$。为了方便起见，在实际应用中$s_{n,m}$通常被设为$H((g^{SK_m})^{SK_n})$。

4. 技术直觉

如上所述，在联邦学习中，每个用户需要向云端提交其本地梯度，然后接收来自服务器的聚合结果(所有本地梯度的聚合)。然而，有三个问题需要解决。首先，我们需要保护用户的本地梯度的隐私，因为攻击者可以通过这些梯度信息间接地破坏用户的敏感信息。其次，为了防止服务器的恶意欺骗，每个用户都应该能够有效地验证服务器返回结果的正确性。第三，在现实场景中，由于网络不可靠或设备电池问题，用户无法按时上传数据到服务器是很常见的。因此，我们提出的协议应该在训练过程中支持出于某些原因而导致的用户离线。在本节中，我们将给出一个技术直觉来解释我们如何解决这三个挑战。

A．单掩码保护用户的梯度

假设每个用户n拥有一个本地梯度x_n，$(n \in U, |U| = N)$，我们最初打算设计一个单掩码协议来保护用户梯度的隐私。具体来说，假设系统中所有用户的ID都是有序的，任意两个用户n和m都协商一个随机数$r_{n,m}$。然后，我们可以对每个用户n的本地梯度x_n进行如下加密:

$$\widehat{x_n} = x_n + \sum_{m\in U:n<m} r_{n,m} - \sum_{m\in U:n>m} r_{n,m} \quad (1)$$

因此，在每个用户提交其加密梯度$\widehat{x_n}$到服务器后，服务器可以计算聚合梯度$\sum_{n\in U} x_n$，如下:

$$Z = \sum_{n\in U} \widehat{x_n} = \sum_{n\in U} x_n \quad (2)$$

然而，这种方法有三个缺点。首先，每个用户需要与所有其他用户协商一个随机数$r_{n,m}$，这将导致二次通信开销$(O(U^2))$。其次，该协议在训练过程中不能支持用

户离线。我们注意到，即使只有一个用户没有按时上传数据，上述聚合操作也无法成功完成，因为添加在该用户梯度中的随机数是无法取消的。第三，上述协议不支持可验证。服务器返回结果的正确性与用户本地梯度的隐私密切相关。一旦攻击者能够掌控数据完整性，用户隐私受到损害的风险就会增加。因此，安全协议还应该支持服务器返回结果的可验证性。

B．支持可验证的双掩码协议

为了解决单掩码协议存在的问题，我们提出了一种双掩码协议。我们首先利用伪随机信号发生器[29]和$diffie-hellman$密钥协议[27][30]在两个用户n和m之间生成随机数$r_{n,m}$。具体来说，我们首先要求可信权威为每个用户n随机创建密钥对$(N_n^{PK},\ N_n^{SK})$。然后，我们需要云服务器广播所有公钥$N_n^{PK},\ n \in U$给所有用户。最后，利用伪随机产生器和$Diffie-Hellman$密钥协议，每两个用户n和m可以生成约定的随机数$s_{n,m} \leftarrow KA.agree(N_n^{SK},\ N_m^{PK})$。因此，每个用户的本地梯度$x_n$可以被以如下方式加密:

$$\widehat{x_n} = x_n + \sum_{m\in U:n<m} PRG(s_{n,m}) - \sum_{m\in U:n>m} PRG(s_{n,m}) \quad (3)$$

其中$PRG(s_{n,m})$是一个伪随机产生器的种子$s_{n,m}$。

接下来，我们在训练过程中采用门限秘密共享方案[26]来支持用户离线。简单地说，为了抵消离线用户梯度中加入的随机数，每个用户n利用门限秘密共享方案提前将自己的私钥N_n^{SK}共享给所有其他用户。因此，如果用户n不能按时向云端提交其数据$\widehat{x_n}$，服务器可以通过要求超过门限的用户提交用户n的秘密共享，对添加到所有其他用户$\widehat{x_m}\ (m \neq n \in U)$中的随机数(即$PRG(s_{n,m})$)进行解密。通过这种方式，可以恢复随机数$PRG(s_{n,m})$，并最终从$\widehat{x_m}$中删除。但是，仍然有一个问题。在某些时候，一些用户可能会延迟向云端上传数据，这可能会导致服务器错误地判断这些用户离线了，并要求其他在线用户上传这些用户的共享来删除随机数。然而，就在这时，这些用户成功地将他们的$\widehat{x_n}$上传到云端。因此，由于服务器有足够的用户秘密共享，它可以通过删除所有随机数$s_{n,m}$得到x_n。为了解决这个问题，我们在每个$\widehat{x_n}$中添加一个新的随机噪声β_n。β_n也将通过门限秘密共享方案共享给所有用户。因此，每个用户的局部梯度x_n被以如下方式加密:

$$\widehat{x_n} = x_n + PRG(\beta_n) + \sum_{m\in U:n<m} PRG(s_{n,m}) - \sum_{m\in U:n>m} PRG(s_{n,m}) \quad (4)$$

这样一来，一旦需要解密去得到聚合的结果$\sum x_n$，云服务器只能从所有在线用户得到β_n，和所有离线用户的N_n^{SK}，因为这些信息足够进行解密操作。

双掩码协议主要是为了在训练过程中保护用户的数据隐私，并在训练过程中支持用户因某些原因离线。但在可验证性方面考虑不足，例如没有设计具体的可验证机制。因此，双掩码协议不支持验证从服务器返回的聚合结果的正确性。我们希望设计一个可验证的解决方案，它还要与我们的双掩码协议高度兼容，并允许每个用户轻松地验证从服务器返回的结果的正确性，而不需要可信的第三方的参与。为了解决这一挑战，我们利用了与伪随机技术集成的同态哈希函数作为可验证方法的底层结构，它允许用户在可接受的开销下验证云服务器执行的正确性。具体验证流程见第五节。

5. 提出的方案

在本节中，我们将介绍$VerifyNet$的技术细节。从更高的层次来看，$VerifyNet$的目的是解决联合训练过程中存在的三个问题。一是保护训练流程中用户的本地梯度的隐私。第二，为了防止服务器的恶意欺骗，我们的$VerifyNet$支持每个用户有效地验证服务器返回结果的

正确性。第三，在训练过程中，$VerifyNet$也支持用户离线。$Fig.4$是我们$VerifyNet$的详细描述，$VerifyNet$由 5 次迭代完成上述任务。具体来说，可信权威首先初始化整个系统，并生成$VerifyNet$中所需的所有公钥和私钥。然后，每个用户n加密其本地梯度x_n并提交给云服务器。在从所有在线用户接收到足够的消息后，云服务器聚合所有在线用户的梯度，并将结果连同证据一起返回给每个用户。最后，每个用户通过验证证据来决定接受或拒绝计算结果，并返回到第 0 轮开始新的迭代。

如第 4 轮所示，具体验证过程如下：

正确性的验证：

$$
\begin{aligned}
(A,\ B) &= (\prod_{n=1}^{n=|U_3|} A_n,\ \prod_{n=1}^{n=|U_3|} B_n) \\
&= (g^{\sum_{n\in U_3} HF_{\delta,\rho}(x_n)},\ h^{\sum_{n\in U_3} HF_{\delta,\rho}(x_n)}) \\
&= (g^{HF_{\delta,\rho}(\sum_{n\in U_3} x_n)},\ h^{HF_{\delta,\rho}(\sum_{n\in U_3} x_n)}) \\
&= (A',\ B')
\end{aligned}
$$

$$
\begin{aligned}
e(A,\ h) &= e(g^{HF_{\delta,\rho}(\sigma)},\ h) = e(\mathrm{g},\ h^{HF_{\delta,\rho}(\sigma)}) \\
&= e(g,\ B)
\end{aligned}
$$

$$
\begin{aligned}
e(L,\ h) &= e(g^{\sum_{n\in U_3} \gamma_n\gamma + v_n v - HF_{\delta,\rho}(x_n)},\ \mathrm{h})^{1/d} \\
&= e(\mathrm{g},\ h^{\sum_{n\in U_3} \gamma_n\gamma + v_n v - HF_{\delta,\rho}(x_n)})^{1/d} \\
&= e(g,\ Q)
\end{aligned}
$$

$$
\begin{aligned}
&e(A,\ h)\bullet\ e(\mathrm{L},\ \mathrm{h})^{d} \\
&= e(g^{HF_{\delta,\rho}(\sigma)},\ h)\bullet\ e(g^{\sum_{n\in U_3} \gamma_n\gamma + v_n v - HF_{\delta,\rho}(x_n)},\ \mathrm{h}) \\
&= e\big(g,\ h\big)^{\sum_{n\in U_3} \gamma_n\gamma + v_n v} \\
&= \Phi
\end{aligned}
$$

如果上述方程不成立，则拒绝聚合结果。否则，接受结果并移动到第 0 轮。用户和云服务器迭代运行 0 到 4 轮，直到整个神经网络配置满足预先设置的约束。

$VerifyNet$的实现过程

1. 第 0 轮(初始化)

TA:

1.为每个用户$n(n \in U, |U| = N)$产生公钥/私钥如$\{(\delta,\rho),(N_n^{PK},\ N_n^{SK}),(P_n^{PK},P_n^{SK}),K=\ (K_1,K_2)\}$,其中$(\delta,\rho)$和$K=(K_1,K_2)$分别是同态哈希函数和伪随机函数中的密钥。$(N_n^{PK},\ N_n^{SK}),(P_n^{PK},P_n^{SK})$将被用于加密用户$n$的本地梯度$x_n$。

用户n:

1.通过一条安全通道给云服务器发送公钥$(N_n^{PK},\ P_n^{PK})$。

服务器端:

1.从至少t个用户$(U_1 \subseteq U)$接收消息，其中t是用在我们模型中$Shamir$的$t-out-of-N$协议的门限。否则，中止并重新开始。

2.广播$\{m,N_m^{PK},P_m^{PK},\tau=sum\}_{m\in U_1}$给每个用户$\in U_1$,其中$\tau=sum$表示用于计算的静态标签。

2. 第 1 轮(密钥共享)

用户n:

1.从云服务器接收$\{m,N_m^{PK},P_m^{PK},\tau=sum\}_{m\in U_1}$。检查是否$|U_1|\geq t$和所有密钥对$(N_m^{PK},\ P_m^{PK})$是否是唯一的。如果不是，中止并重新开始。

2.选择一个随机数β_n。生成β_n作为$\{(m,\beta_{n,m})\}_{m\in U_1}\leftarrow S.share(\beta_n,t,U_1)$，其中$\beta_{n,m}$是用户$n$对用户$m$的共享。

3.产生N_n^{PK}作为$\{(m,N_{n,m}^{SK})\}_{m\in U_1}\leftarrow S.share(N_n^{SK},t,U_1)$，其中$N_{n,m}^{SK}$是用户$n$对用户$m$的共享。

4.计算$P_{n,m}\leftarrow AE.enc\left(KA.agree(P_n^{SK},P_m^{PK}),n||m||N_{n,m}^{SK}||\beta_{n,m}\right)_{m\in U_1}$，其中$AE.enc()$表示使用密钥$KA.agree(P_n^{SK},P_m^{PK})$的一个系统加密。

5.发送$\{P_{n,m}\}_{m\in U_1}$给云服务器。

服务器端:

1.从至少t个用户$(U_2\subseteq U_1)$接收消息。否则，中止并重新开始。

2.广播$\{P_{m,n}\}_{m\in U_2}$给每个用户$\in U_2$。

3. 第 2 轮(掩码输入)

用户n:

1.从云服务器接收$\{P_{m,n}\}_{m\in U_2}$。检查是否$U_2\subseteq U_1$和$|U_2|\geq t$。如果不是，中止并重新开始。

2.和每个用户$m\in U_2$计算共享密钥如$s_{n,m}\leftarrow KA.agree(N_s^{SK},N_m^{PK})$

3.加密本地梯度如$\widetilde{x_n}\ =\ x_n\ +\ PRG(\beta_n)\ +\sum_{m\in U_2:n<m}PRG(s_{n,m})\ -\ \sum_{m\in U_2:n>m}PRG(s_{m,n})$

4.为了验证未来从服务器返回结果的正确性，一些信息被如下计算:

计算$HF(x_n)=(A_n,\ B_n)=\ (g^{HF_{\delta,\ \rho^{(x_n)}}},\ h^{HF_{\delta,\ \rho^{(x_n)}}})$

计算$PF_{K_1}(n)=\ (\gamma_n,\ \nu_n);\ PF_{K_2}(\tau)=\ (\gamma,\ v)$

计算$PF_K(n,\ \tau)=(E_n,\ F_n)=(g^{\gamma_n\gamma+v_nv},\ h^{\gamma_n\gamma+v_nv})$

5.计算$L_n\ =\ (E_n\bullet A_n^{-1})^{1/d}=(g^{\gamma_n\gamma+v_nv-HF_{\delta,\rho(x_n)}})^{1/d}$,其中$d$是正整数

6.计算$Q_n\ =\ (F_n\bullet B_n^{-1})^{1/d}=(h^{\gamma_n\gamma+v_nv-HF_{\delta,\rho(x_n)}})^{1/d}$,其中$d$是正整数

7.发送$\sigma_n=(\widetilde{x_n},A_n,B_n,L_n,Q_n,\Omega_n=1)$给服务器

服务器端:

1.从至少t个用户$(U_3\subseteq U_2)$接收消息。否则，中止并重新开始。

2.广播U_3给每个用户$\in U_2$。

4. 第 3 轮(解码)

用户n:

1.检查是否$U_3\subseteq U_2$和$|U_3|\geq t$。如果不是，中止并重新开始。

2.解密每个$P_{n,m},m\in U_2\backslash\{n\}$作为$n||m||N_{n,m}^{SK}||\beta_{n,m}\leftarrow AE.dec\left(KA.agree(P_n^{SK},P_m^{PK}),P_{n,m}\right)$

3.发送$\{(N_{n,m}^{SK})|m\in U_2\backslash U_3\}$和$\{(\beta_{n,m})|m\in U_3\}$给云服务器，其中$U_2\backslash U_3$表示在第 1 轮中发送数据给服务器，但是在第 2 轮上传数据给服务器之前就离线的用户。

服务器端:

1.从至少t个用户($U_4 \subseteq U_3$)接收消息。否则，中止并重新开始。

2.计算$N_n^{SK} \leftarrow S.recon(\{N_{n,m}^{SK}\}_{m\in U_4}, t)$

3.计算$\beta_n \leftarrow S.recon(\{\beta_{n,m}\}_{m\in U_4}, t)$

4.计算$PRG(s_{n,m}) \leftarrow PRG(KA.agree(\{N_n^{SK}, N_m^{PK}\}_{n\in U_2\setminus U_3, m\in U_3}))$

5.计算$PRG(\beta_n)_{n\in U_3}$

6.为所有用户$\in U_3$计算聚合梯度如:

$\sum_{n\in U_3} x_n = \sum_{n\in U_3} \overline{x_n} - \sum_{n\in U_3} PRG(\beta_n) - \sum_{n\in U_3, m\in U_2\setminus U_3: n<m} PRG(s_{n,m}) + \sum_{n\in U_3, m\in U_2\setminus U_3: n>m} PRG(s_{m,n})$

7.计算聚合梯度的证据$\{A, B, L, Q, \Omega\}$如下:

$A = \prod_{n=1}^{n=|U_3|} A_n;\ B = \prod_{n=1}^{n=|U_3|} B_n;\ L = \prod_{n=1}^{n=|U_3|} L_n;\ Q = \prod_{n=1}^{n=|U_3|} Q_n;\ \Omega = \prod_{n=1}^{n=|U_3|} \Omega_n$

8.广播C_{result}={σ=$\sum_{n\in U_3} x_n, A, B, L, Q, \Omega$}给每个用户$\in U_4$

5. 第 4 轮(验证)

用户n:

1.知道$PF_{K_1}(n) = (\gamma_n,\ v_n)$和$PF_{K_2}(\tau) = (\gamma,\ v)$，计算 φ=$\sum_{n\in U_3}(\gamma_n\gamma + v_n v)$和 Φ=$e(g,h)^{\varphi}$

2.验证$(A, B) = (A',\ B')$,$e(A,h) = e(g,B); e(L,h) = e(g,Q), \Phi = e(A,h) \bullet e(L,h)^d$这些等式是否成立

3.如果以上任何等式不成立，就拒绝聚合结果。否则，接收结果并转向第 0 轮

$Fig.4$ $VerifyNetd$的细节描述

6. 安全性分析

在本节中，我们首先简要描述验证的正确性。然后，分析$VerifyNet$如何保证每个用户本地梯度的机密性。其他安全指标超出了本文的范围。

A．验证正确性

如第五节所示，在从云服务器收到$\{\sigma,\ A,\ B,\ L,\ Q,\ \Omega\}$后，每个用户先检查是否$\Phi = e(A,\ h) \bullet e(\mathrm{L},\ \mathrm{h})^d$。基于$l-BDHI$假设[25]仅当$\sum_{n\in U_3}\gamma_n\gamma + v_n v$包含在$L$中(在$g$的指数中)，才能实现$\Phi = e(A,\ h) \bullet e(\mathrm{L},\ \mathrm{h})^d$。如果这样，则每个用户可以推断出$L = \prod_{n=1}^{n=|U_3|} L_n = g^{\sum_{n\in U_3}\gamma_n\gamma + v_n v - HF_{\delta,\rho}(\sigma)}$，并且知道$A = g^{HF_{\delta,\rho}(\sigma)}$。然后，基于$DDH$假设[32]，每个用户进一步检查是否$e(A,h) = e(g,B)$和$e(L,h) = e(g,Q)$都成立。如果都成立，那么每个用户都会相信云服务器正确计算了B和Q。直到这里，每个用户都验证了(A,B)是正确计算的。最后，如果$HF(\sigma) = (A',\ B') = (A,B)$成立，每个用户都确信云服务器确实返回了正确的聚合结果$\sigma = \sum_{n\in U_3} x_n$。

这里我们省略了详细的证明，因为利用$l-BDHI$[25]和DDH假设[32]可以很容易地证明。

B．诚实但是好奇的安全性

在本节中，我们要证明我们的$VerifyNet$在诚实但好奇的设置下是安全的。在我们的威胁模型中，云服务器可以与任何$t-1$个用户串通去获得最具攻击性的能力，但它们除了聚合的结果之外仍然不知道诚实用户的本地梯度。如上所述，每个用户

的本地梯度x_n被加密为:

$$\widetilde{x_n} = x_n + PRG(\beta_n) + \sum_{m\in U_2:n<m} PRG(s_{n,m}) - \sum_{m\in U_2:n>m} PRG(s_{m,n})$$

此外，每个x_n还用于同态哈希函数[24][25]中生成部分验证信息。由于同态哈希函数已被证明是安全的[25]，这里我们主要讨论$\widetilde{x_n}$可以实现的隐私保护级别。在正式介绍完整的证明过程之前，我们先介绍一些以后会用到的符号。

我们知道用户可能会在训练流程的某个时刻离线。我们用$U_i \subseteq U$表示在第$i-1$轮时顺利上传数据到云服务器的用户。因此，我们有$U \supseteq U_1 \supseteq U_2 \supseteq U_3 \supseteq U_4$。标识$U_i \setminus U_{i+1}$是用来代表那些在$i-1$轮发送数据到服务器，但是在第$i$轮上传数据之前掉线的用户。如之前所示，假定每个用户n持有一个本地梯度x_n(n∈U)，我们采取$x_{U'} = \{x_n\}_{n\in U'}$表示一个本地梯度的子集$U'$，其中$U' \subseteq U$

在我们的$VerifyNet$中，一部分的视图被定义为其内部状态(包含其输入和随机性)以及从其他部分接收到的所有消息。需要注意的是，当这部分在某个时刻退出执行时，该部分将立即停止接收消息。

简单来说，我们使用S表示云服务器。给定一个子集$W \subseteq U \cup \{S\}$部分,各方的共同视图W可以表示为一个随机变量$REAL_W^{U,t,k}(x_U, U_1, U_2, U_3, U_4)$，其中 t和k在我们的协议中分别表示阈值和安全参数。

接下来我们将提出两个定理。第一个定理表明，任何少于t个用户的串通(不包括云服务器)都无法获得除聚合结果以外的其他用户的私有信息。

定理 1(防御来自多个用户的联合攻击):对于所有$t,k,W \subseteq U, x_U, |U| \geq t$的$U$和$U_4 \subseteq U_3 \subseteq U_2 \subseteq U_1 \subseteq$U,有一个 PPT 模拟器 SIM 模拟的输出和$REAL_W^{U,t,k}$的输出没有区别。

$$REAL_W^{U,t,k}(x_U, U_1, U_2, U_3, U_4) \equiv SIM_W^{U,t,k}(x_W, U_1, U_2, U_3, U_4)$$

证明:由于我们排除了云服务器的参与，集合W中各方的联合视图不依赖于集合W以外的其他用户的输入。因此，模拟器可以通过运行协议，使用诚实但好奇的用户的真实输入，生成完美的模拟，而不是将诚实用户的输入替换为虚假数据(如随机生成的数字)。我们强调，在W中模拟的用户视图与真实视图的输出是相同的。更具体地说，在第二轮中，模拟器通过使用随机数(例如 0)而不是使用真正的梯度，为所有诚实的用户(不在W中的)生成掩蔽输入$\widetilde{x_n}$。另外，我们注意到在解掩码这一轮中，服务器只发送了所有在线用户ID的列表，而不是具体的$\widetilde{x_n}$的真实值，这意味着诚实而好奇的用户无法识别云服务器返回的计算结果是否基于诚实用户的真实梯度。因此，在W中的用户的模拟视图与真实视图$SIM_W^{U,t,k}$的输出是难以区分的。

定理 2(防御云服务器和多个用户的联合攻击):对于所有$t,k,x_U,W \subseteq U \cup \{S\}, |W \setminus \{S\}| < t, |U| \geq t$的 U 和$U_4 \subseteq U_3 \subseteq U_2 \subseteq U_1 \subseteq$U，有一个 PPT 模拟器 SIM 模拟的输出和$REAL_W^{U,t,k}$的输出没有区别。

$$REAL_W^{U,t,k}(x_U, U_1, U_2, U_3, U_4) \approx SIM_W^{U,t,k}(x_W, U_1, U_2, U_3, U_4)$$

其中，

$$\xi = \begin{cases} \sum_{n\in U_3\setminus W} x_n & \text{if } |U_3| \geq t \\ \perp & otherwise \end{cases}$$

证明:我们使用一个标准的混合论证来证明定理 2。主要思想是模拟器SIM对我们的协议执行一系列修改，这最终使模拟视图$SIM_W^{U,t,k}$与真实视图$REAL_W^{U,t,k}$难以区分。在我们的混合参数中，hyb_i，$\{i = 1, \cdots 9\}$表示对原始协议的安全修改，这确保修改操作与原始操作一样。

hyb_1　在这个修改中，模拟器改变了所有诚实用户n的行为，其中 $n \in \{U_2 \backslash W\}$。即对于每个用户$n$，选择一个均匀随机数$v_{n,m}$来替换在同一集合中的用户$n$和$m$之间的共享密钥$KA.agree\ (P_n^{SK},\ P_m^{PK})$，并执行加解密功能的函数。例如，每个诚实用户使用$v_{n,m}$来生成第1轮提到的密文$P_{n,m}$，而不是使用$KA.agree\ (P_n^{SK},\ P_m^{PK})$。$DDH$假设[32]确保该混合协议与真实协议完全一样。

hyb_2　在这个修改中，模拟器替换所有加密数据(例如加密共享β_n和N_n^{SK})，这些加密数据是诚实用户(在集合$\{U_2 \backslash W\}$中)使用随机数(例如，具有合适的长度的0)的加密共享发送给其他用户的。然而，所有诚实用户仍然在解掩码轮次中向云服务器返回正确的共享。因为我们只是改变了密文的内容，所以对称认证加密协议[11][31]的属性可以保证该混合协议与真实协议之间一样。

hyb_3　在这个修改中，我们首先定义一个子集，如下所示。

$$U^* = \begin{cases} U_2 \backslash W \quad if\ \xi = \perp \\ \\ U_2 \backslash U_3 \backslash W \quad otherwise \end{cases}$$

然后，在密钥共享轮次中，对于集合U^*中的所有诚实用户n，模拟器将所有β_n的共享替换为随机值(例如，具有适当长度的0)。很明显，对手不会得到额外的β_n，可能因为诚实的用户不会透露他们的β_n份额 $(|U_3| \geq t,\ U^* = U_2 \backslash U_3 \backslash W)$，也可能因为所有的诚实用户都离线了$(|U_3| < t,\ U^* = U_2 \backslash \mathrm{W},$ 其中$\xi = \perp)$。$Shamir$的秘密共享方案保证了即使拥有任何$t-1$当前秘密的份额都不能恢复这个秘密,这意味着即使是诚实但好奇的用户拥有β_n的$|W| < t$的份额,他们仍然无法判断这些份额是否来自真实β_n的诚实用户提交的。

hyb_4　在这个修改中，模拟器不是为U^*中的所有用户生成$PRG(\beta_n)$，而是使用大小适当的均匀随机数r_n来替换它。很容易理解，模拟器只是替代了PRG的输出，而PRG就是前面提到的伪随机发生器[29]。因此，伪随机生成器[11]的安全性确保了该混合协议与真实协议难以区分。

hyb_5　在这个修改中，对于集合U^*中的每个用户n，模拟器生成如下的屏蔽输入:

$$\widehat{x_n} = r_n + \sum_{m \in U_2: n<m} PRG(s_{n,m}) - \sum_{m \in U_2: n>m} PRG(s_{m,n})$$

而不是使用

$$\widehat{x_n} = x_n + PRG(\beta_n) + \sum_{m \in U_2: n<m} PRG(s_{n,m}) - \sum_{m \in U_2: n>m} PRG(s_{m,n})$$

由于$PRG(\beta_n)$在之前的混合中已经用均匀随机数改变了，我们知道$x_n + r_n$也是一个随机值，很容易推导出r_n和$x_n + PRG(\beta_n)$的分布是一样的。

需要注意的是，如果$\xi = \perp$，则模拟器已经完成了模拟(如hyb_5所描述)，因为SIM在不知道$x_n (n \in W)$的情况下，成功地模拟了$REAL$，因此

对于接下来的所有模拟，我们假设$\xi \neq \perp$。

hyb_6　在这个修改中，对于每个用户$n \in U_3\backslash W$，模拟器将N_n^{SK}替换为随机值(例如，具有适当长度的 0)。与hyb_3类似，$Shamir$的秘密共享协议的安全性保证了这种混合协议与真实协议一样。

hyb_7　在这个修改中，给定用户$m' \in U_3\backslash W$，对于所有其他用户$n \in U_3\backslash W$，模拟器一致地选择一个随机数来替换用户n和m'之间的共享密钥(例如$s_{n,m'} = KA.agreee\{N_n^{SK}, N_m^{PK}\}$，这个随机数将被用作用户$n$和$m$的$PRG$的种子。

具体来说，为每个用户$n \in U_3\backslash W\backslash\{m'\}$选择一个随机值$s'_{n,m'}$。不是发送

$\widehat{x_n} = x_n + PRG(\beta_n) + \sum_{m\in U_2:n<m} PRG(s_{n,m}) - \sum_{m\in U_2:n>m} PRG(s_{m,n})$

而是 SIM 提交

$\widehat{x_n} = x_n + r_n + \sum_{m\in U_2\backslash\{m'\}:n<m} PRG(s_{n,m}) - \sum_{m\in U_2\backslash\{m'\}:n>m} PRG(s_{m,n}) + \triangle_{n,m'} \bullet PRG(s'_{n,m'})$

其中如果$n < m'$则$\triangle_{n,m'} = 1$。否则，$\triangle_{n,m'} = -1$

相应地，我们能得到

$$\widehat{x_{m'}} = x_{m'} + r_{m'} + \sum_{m\in U_2} \triangle_{n,m'} \bullet PRG(s'_{n,m'})$$

类似地，DDH假设[32]确保这个混合协议与真实协议难以区分。

hyb_7　在这个修改中，相较于在之前的混合中挑选同样的用户m'和所有其他用户$n \in U_3\backslash W$，模拟器也均匀地选择一个随机数$r_{n,m'}$以取代$PRG(s'_{n,m'})$的计算。与hyb_4类似，很容易理解，模拟器只是替代了PRG的输出。因此，伪随机生成器[11]的安全性确保了该混合协议与真实协议难以区分。

hyb_6　在这个修改中，对于集合$U_3\backslash W$中的每个用户n，模拟器提交

$$\widehat{x_n} = R_n + r_n + \sum_{m\in U_2\backslash U_3\backslash W} \triangle_{n,m} \bullet r_{n,m}$$

而不是发送

$\widehat{x_n} = x_n + PRG(\beta_n) + \sum_{m\in U_2:n<m} PRG(s_{n,m}) - \sum_{m\in U_2:n>m} PRG(s_{m,n})$

其中$\{R_n\}_{n\in U_3\backslash W}$为模拟器选取的随机值且受$\sum_{n\in U_3\backslash W} R_n = \sum_{n\in U_3\backslash W} x_n = \xi$的影响。因此，由于$SIM$对于所有参与方$n \in w$，在不知道$x_n$的情况下，成功地模拟了$REAL$，因此模拟器已经完成了模拟。基于$hyb_1$到$hyb_9$，我们可以推断，协议修改之后的输出与真实的输出分布相同。完成证明。

7. 性能评估

我们招募了 600 台移动设备来评估我们$VerifyNet$的性能，其中大多数智能设备都配备了$4GB$内存，并配备了$Android$ 6.0系统。每个移动设备离线运行相同的卷积神经网络，获取所有参数的本地梯度。所有的原始数据都是从$MNIST$数据库$(http://yann.lecun.com/exdb/mnist/)$中选取的，该数据库有 60,000 个样本的训练集，10,000 个样本的测试集。此外，“云”是用联想服务器模拟的，该服务器采用$Intel(R)\ Xeon(R)\ E5-2620\ 2.10GHZ\ CPU, 16GB\ RAM, 256SSD, 1TB$机械硬盘的配置，运行在$Ubuntu$ 18.04操作系统上。具体来说，我们采用基于椭圆曲线的密钥协商协议来实现两个用户之间的密钥分配，和标准的$Shamir$的$t-out-of-N$秘密共享协议[26]来生成秘密的共享。另外，我们在计数器模式使用AES和 128 位密钥的$AES-GCM$分别实现认证加密和伪随机发生器。

A. 功能性

如$TABLE.I$所示，我们比较了最新的$PPML$[11]，$PPDL$[19]和$SafetyNets$[16]的功能性，因为这些方案的主要工作与我们的类似。具体来说，我们知道$PPML$和$PPDL$在执行过程中都保证了数据隐私的机密性，但是它们的模型不支持可验证性。此外，$PPDL$也未能解决用户离线问题。另一方面，$SafetyNets$主要从可验证的角度进行设计，因此在此协议中没有考虑到数据隐私泄露和用户在训练过程中离线的问题。与这些方案相比，$VerifyNet$支持每个用户对云服务器返回的结果进行验证，同时保证了用户本地梯度的机密性。此外，与$PPML$类似，$VerifyNet$在整个训练过程的任何子过程中允许用户离线。

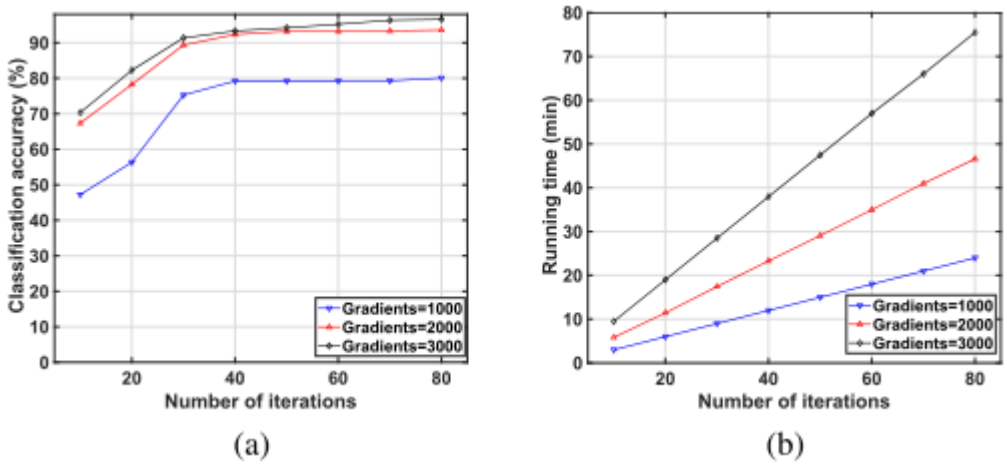
	PPML	PPDL	SafetyNets	VerifyNet
Data Privacy	✓	✓	×	✓
Robustness to Failures	✓	×	×	✓
Verifiability	×	×	✓	✓

$TABLE.I$ $SafetyNets$的详细描述

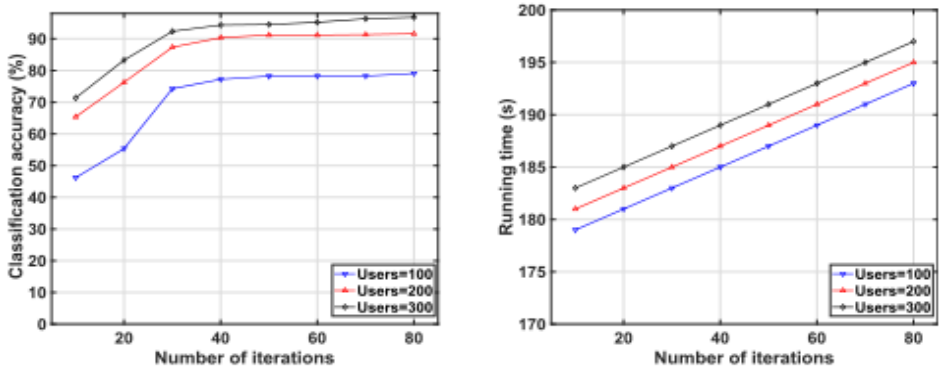
B. 分类精度

在本节中，我们选择$MNIST$数据库中的数据来测试我们$VerifyNet$的分类准确性。实验在卷积神经网络[16][19]上进行，该网络由两层卷积层和两层全连接层组成，每层 128 个神经元。当然，在联邦学习中，模型输出的准确性与两个因素密切相关，即参与训练的用户数量和每个用户拥有的本地梯度的大小。一般来说，模型输出的准确性与参与训练的梯度/用户数量成正比，也与系统产生的计算和通信开销成正比。为了分析这些因素之间的关系，我们记录了$VerifyNet$在每个用户不同数量/梯度下的分类精度和运行时间。这里我们使用符号$|U|$和$|G|$分别表示我们实验中每个用户的用户数和梯度。

$Fig.5$显示每个用户不同梯度数量下的分类精度和运行时间，其中一个迭代意味着一个参数更新(即从第 0 轮到第 4 轮)已经完成。为了简单起见，这里我们只考虑没有用户离线的情况。显然，梯度数量的增加有利于模型输出精度的提高，但也带来了更多的计算开销(如$Fig.5(b)$所示)。但是，通过比较不同梯度的分类精度(分别看梯度为 2000 和梯度为 3000)，训练中涉及的梯度的数量并不是越多越好，因为当梯度的数量增加到一定数量时，模型的精度会收敛。因此，在实际应用中，我们可以根据经验选择适当数量的梯度，以避免不必要的开销。$Fig.6$显示不同用户数下的分类准确率和运行时间。同样，增加系统中用户的数量有利于提高模型的分类精度，但也需要额外的计算开销。请注意，为了简单起见，我们不记录在不同数量的用户/梯度下在系统中传输的数据总量。然而，由于$VerifyNet$是一个交互协议，从理论上讲，随着用户/梯度的增加，我们的方案将不可避免地产生一定的通信开销。



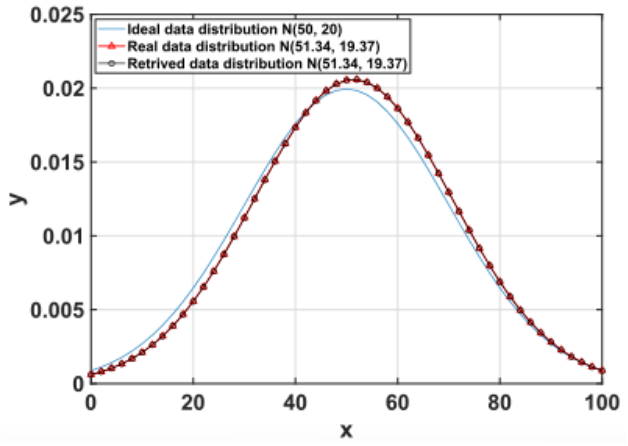
$Fig.5$ (a)没有离线，$|U| = 100$，每个用户不同梯度数量的分类精度
(b)没有离线，$|U| = 100$，每个用户不同梯度数量的运行时间



$Fig.6$ (a)没有离线，$|G| = 1000$，不同用户数量的分类精度
(b)没有离线，$|G| = 1000$，不同用户数量的运行时间

C. 验证的准确性

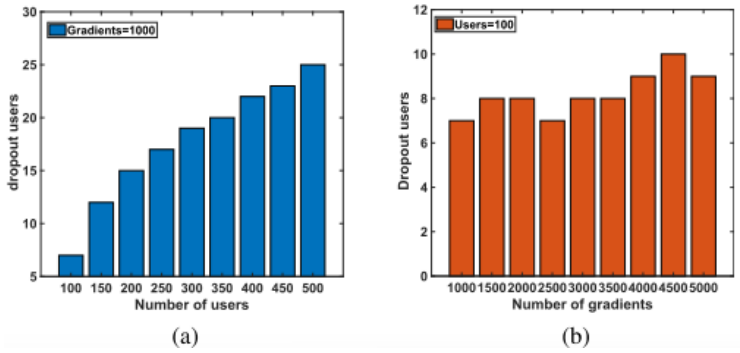
如上所述，为了防止服务器的恶意欺骗，$VerifyNet$支持每个用户验证服务器返回结果的正确性。具体来说，云服务器需要向每个用户提供关于其聚合结果正确性的证据，每个用户可以通过检查证据拒绝或接受结果。为了验证的准确性，我们模拟 200 个用户上传加密的本地数据到服务器，所有数据都是从正态分布$N(50,20)$中随机选取的。为了简单起见，这里我们也只考虑没有用户离线的情况。由于随机选取的数据为离散点，其实际分布($N(51.34,19.37)$，$Fig.7$中的红线)与原来的理想分布略有不同。然后，我们要求云服务器为每个用户计算聚合的结果以及相应的证据。如果验证通过，用于生成聚合结果上传的数据分布应与真实数据一致。在此基础上，我们进一步使用聚合的结果去计算上传数据的均值和方差。如$Fig.7$所示，我们可以发现检索到的数据分布与真实的数据分布正好重叠，这也证实了通过验证证据后从服务器返回的结果是正确的。



$Fig.7$ 验证的精确性

D. 用户离线的概率

我们的$VerifyNet$允许用户在训练过程中离线，因为在训练流程中，由于用户设备电池问题、硬件质量问题等原因导致的用户离线是非常常见的。为了评估这种现象的普遍性，我们记录了整个系统中每个用户在不同的用户数量/梯度下的离线用户数。具体来说，我们要求所有用户在指定范围时间内多次向服务器上传数据，并记录重复实验中离线用户的平均数。如$Fig.8$所示，我们发现，随着用户/梯度数量的增加，一定数量的用户离线是不可避免的，并且随着用户数量的增加，这种现象更加明显。但是从$Fig.8$可以看出，用户离线的比例相对于总用户数量并不显著。因此，这也为使用秘密共享协议管理用户离线问题提供了依据。



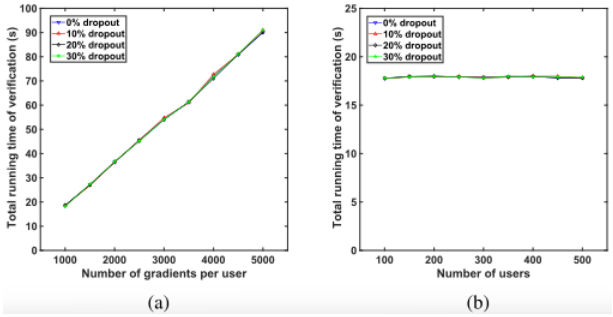
$Fig.8$ 离线用户。$(a)|G| = 1000$，用户数量不同
$(b)|G| = 100$，每个用户梯度数量不同

E. 客户端的性能分析

我们从计算开销和通信开销两方面分析了客户端的性能，在不同离线用户比例下对$VerifyNet$进行了测试。

(1) 计算开销：$Fig.9$显示每个用户在验证过程中的运行时间。显然，用户的运行时间随着梯度数量的增加而线性增加，但随着用户数量的增加而保持不变。主要原因之一是验证开销只是与每个用户拥有的梯度数量有关，因为每个用户n需要为新增梯度x_n生成 (A_n, B_n, L_n, Q_n)。$Fig.10$显示了验证的计算开销和总开销的对比。为了简单起见，我们在实验中考虑没有用户离线和有30%的用户离线的情况。我们可以看到，每个用户的主要计算成本来自于验证过程，而与用户数量或梯度无关。此外，我们的$VerifyNet$在计算开销方面保持了良好的性能。例如，在我们的系统中，当用户数为 500，梯度总数为 500000 时，每个用户只需要大约 17 秒就

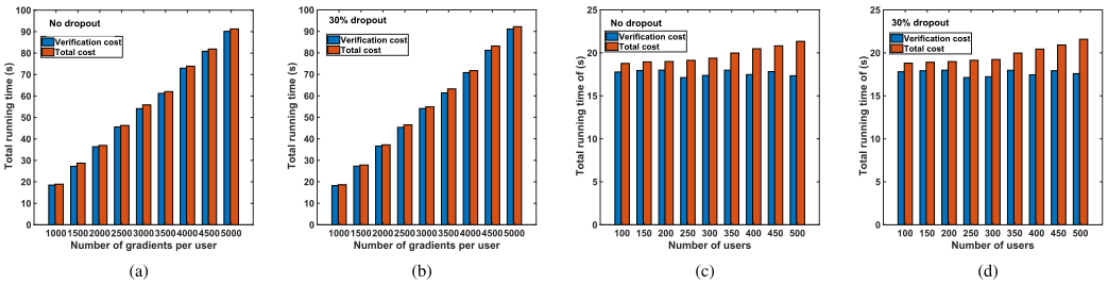
可以完成一次参数更新迭代。



$Fig.9$ 每个用户的总共运行时间(验证过程)

$(a)|U| = 100$，每个用户有不同的梯度数量

$(b)|G| = 1000$，用户数量不同



$Fig.10$ 每个用户的验证计算开销和总开销的比较

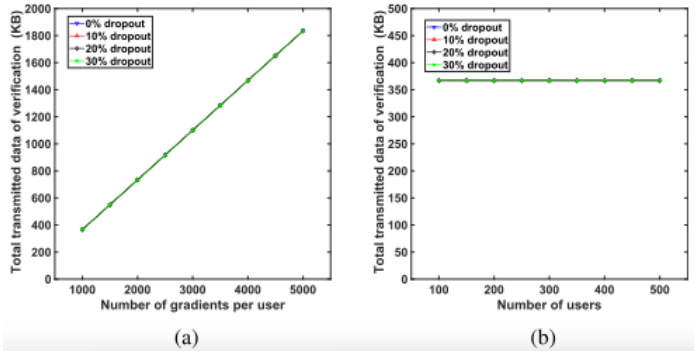
(a)没有离线，$|U| = 100$，每个用户有不同的梯度数量

$(b)30\%$离线，$|U| = 100$，每个用户有不同的梯度数量

(c)没有离线，$|G| = 1000$，用户数量不同

$(d)30\%$离线，$|G| = 100$，用户数量不同

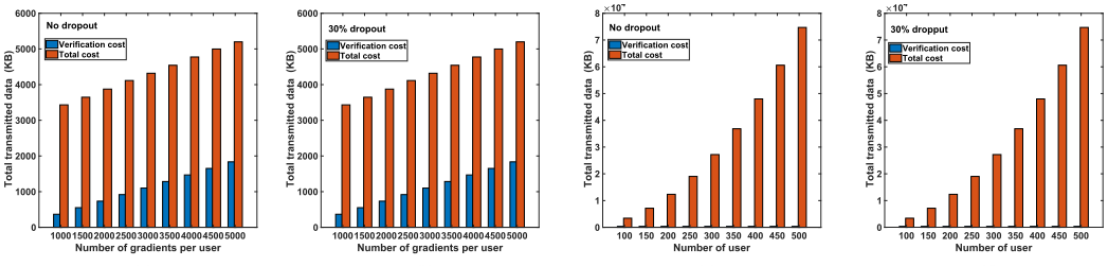
(2) 通信开销：$Fig.11$显示每个用户在验证过程中传输的数据总数。与$Fig.9$相似，用户的传输数据总量也随着梯度的增加而线性增加，但随着用户数量的增加而保持不变。$Fig.12$为验证通信开销与每个用户总开销的对比。我们可以看到，验证过程中产生的开销占总开销的比例并不明显，甚至可以随着用户数量的增加而忽略不计。此外，实验表明我们的$VerifyNet$在通信开销方面仍然保持着良好的性能。例如，在我们的系统中，当用户数为 500，梯度总数为 500000 时，每个用户只需要大约 70MB 就可以完成一次参数更新迭代。



$Fig.11$ 每个用户传输的数据总数(验证过程)

$(a)|U| = 100$，每个用户有不同数量的梯度

$(b)|G| = 1000$，用户的数量不同



$Fig.12$ 每个用户的验证通信开销和总开销的比较

(a)没有离线，$|U| = 100$，每个用户有不同数量的梯度

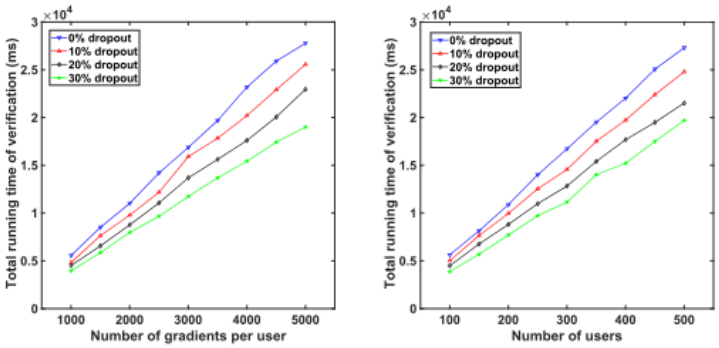
(b)30%离线，$|U| = 100$，每个用户有不同数量的梯度

(c)没有离线，$|G| = 1000$，用户的数量不同

(d)30%离线，$|G| = 1000$，用户的数量不同

F. 服务器性能分析

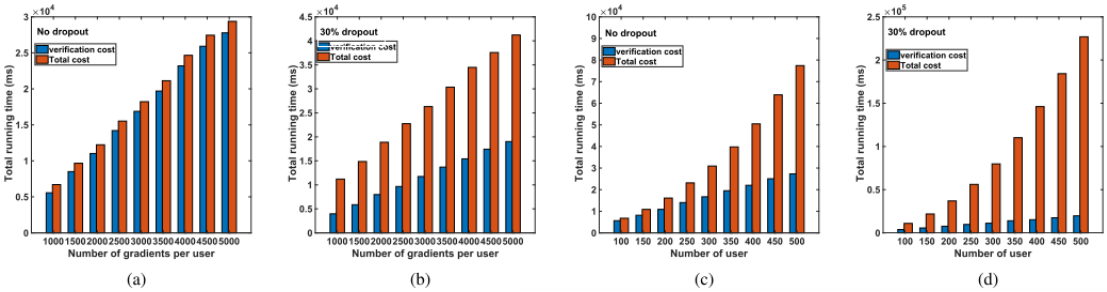
（1） 计算开销：$Fig.13$显示云服务器的验证过程运行时间。我们可以看到，服务器的运行时间随着梯度或用户数量的增加而线性增加。主要原因是，随着梯度或用户数量的增加，云服务器需要为每个新增的梯度和用户生成聚合结果的证据。$Fig.14$显示了云服务器的验证计算开销与总开销的比较。通过对比$Fig.14(c)$和$Fig.14(d)$，我们可以很明显地发现用户离线比例在很大程度上决定了云服务器整体成本的走向。例如，当没有用户离线时，云服务器只需要大约 75000 ms来完成一次参数更新的迭代，但如果30%的用户离线，则需要 220000 ms。



$Fig.13$ 云服务器的总运行时间(验证过程)

$(a)|U| = 100$，每个用户有不同数量的梯度

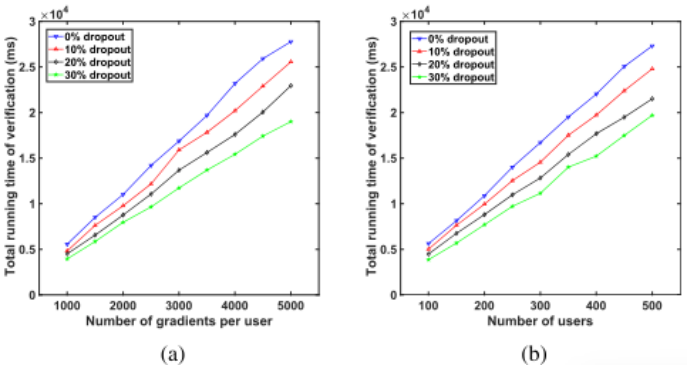
$(b)|G| = 1000$，用户数量不同



$Fig.14$ 云服务器验证计算开销和总开销的比较

(a)没有离线，$|U| = 100$，每个用户有不同数量的梯度
(b)30%离线，$|U| = 100$，每个用户有不同数量的梯度
(c)没有离线，$|G| = 1000$，用户数量不同
(d)30%离线，$|G| = 1000$，用户数量不同

(2) 通信开销：$Fig.15$显示云服务器验证过程中传输的总数据。我们可以看到，随着用户或梯度数量增加，云服务器的通信开销也呈线性增长。$TABLE.II$和$TABLE.III$分别显示每一轮的计算和通信开销，其中红色字体表示验证过程中的开销。对于每个用户，无论是计算开销还是通信开销都主要来自于掩码输入和验证，因为每个用户n都需要对每个梯度生成σ_n和验证证据，并将加密后的结果发送到云服务器。对于云服务器来说，在掩码输入轮接收到所有消息后，需要聚合所有用户的加密梯度，并在解码轮中恢复所有在线用户的秘密，这会导致很大的计算/通信开销。



$Fig.15$ 云服务器传输数据总数(验证过程)
$(a)|U| = 100$，每个用户有不同数量的梯度
$(b)|G| = 1000$，用户数量不同

	Dropout	Key Sharing	Masked Input	Unmasking	Verification	Total
Client	0%	2508(ms)	(1311 + 11919) (ms)	4(ms)	5941(ms)	21683 (ms)
Client	10%	2510(ms)	(1263 + 11942)(ms)	4(ms)	5830 (ms)	21549 (ms)
Client	20%	2492(ms)	(1250 + 11892)(ms)	4(ms)	5942 (ms)	21580 (ms)
Client	30%	2480(ms)	(1245 + 11971) (ms)	4(ms)	5942 (ms)	21642 (ms)
Server	0%	0(ms)	0 (ms)	(50136 + 27311)(ms)	0 (ms)	77477 (ms)
Server	10%	0(ms)	0 (ms)	(112379 + 24799)(ms)	0 (ms)	137178 (ms)
Server	20%	0(ms)	0 (ms)	(163551 + 21519)(ms)	0 (ms)	185070 (ms)
Server	30%	0(ms)	0 (ms)	(207222 + 19705)(ms)	0 (ms)	226927 (ms)

$TABLE.II$ 每轮计算开销

	Dropout	Key Sharing	Masked Input	Unmasking	Verification	Total
Client	0%	274(KB)	(74002 + 184) (KB)	65(KB)	(4 + 184)(KB)	73 (MB)
Client	10%	273(KB)	(73998 + 184) (KB)	65(KB)	(4 + 184)(KB)	73 (KB)
Client	20%	274(KB)	(73999 + 184)(KB)	65(KB)	(4 + 184) (KB)	73 (MB)
Client	30%	274(KB)	(73998 + 184)(KB)	65(KB)	(4 + 184) (KB)	73 (MB)
Server	0%	72(MB)	(111 + 90) (MB)	(32 + 0.18)(MB)	0 (MB)	305.18 (MB)
Server	10%	72(MB)	(110 + 81) (MB)	(29 + 0.18)(MB)	0 (MB)	292.18 (MB)
Server	20%	72(MB)	(102 + 72) (MB)	(25 + 0.18)(MB)	0 (MB)	271.18 (MB)
Server	30%	72(MB)	(98 + 63) (MB)	(22 + 0.18)(MB)	0 (MB)	255.18 (MB)

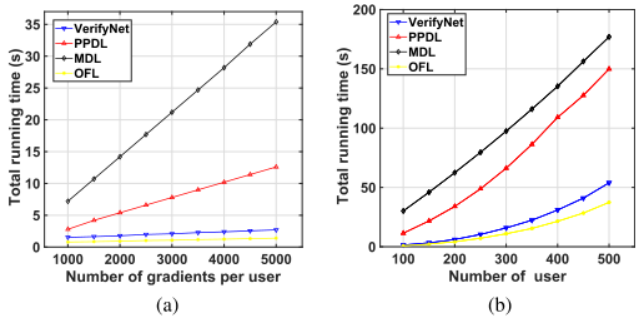
$TABLE.III$ 每轮通信开销

G. 与现有方法比较的性能分析

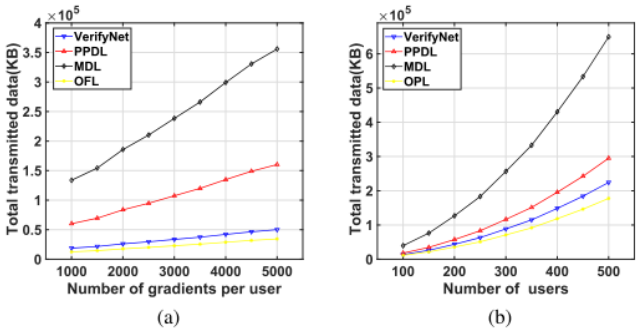
在本节中，我们通过与最先进的方法比较来分析$VerifyNet$的成本，这些方法有MDL[33]、$PPDL$[19]、$SafetyNets$[16]和原始联邦学习模型OFL[10]，其中OFL是

在明文环境中执行联邦学习的原始模型。这里我们使用OFL来描述联邦学习在明文和密文中的性能差异。MDL[33]和$PPDL$[19]与我们的$VerifyNet$中考虑的场景是一致的，它们的目标也是通过隐私保护技术来保护用户本地梯度的隐私。但是，他们没有考虑服务器返回结果的可验证性问题。相反，$SafetyNets$[16]的目标是验证从云返回结果的正确性，这是在不可信云上深度神经网络的可验证性执行的第一种方法。它可以将某些类型的深度神经网络转换成算术电路，然后通过与服务器的多次交互来验证返回结果的正确性。

1. 加密过程的性能：如$Fig.17$和$Fig.18$所示，我们记录了在不同用户/梯度数量下$VerifyNet$、MDL、$PPDL$和OFL中每个用户的运行时间和总传输数据。由于在MDL和$PPDL$中没有考虑可验证的计算，验证所需的计算和通信开销也被排除在我们的$VerifyNet$之外。另外，这里我们只考虑没有用户离线的情况，因为MDL和$PPDL$不支持用户在训练过程中离线。我们可以看到$VerifyNet$的成本明显小于MDL和$PPDL$，而并不比原来的解决方案OFL大多少。这主要是因为与MDL和$PPDL$中使用的技术相比，我们的双掩码协议具有更高的效率。特别是$ElGamal$ 密码系统[34]被使用在MDL中去加密用户的本地梯度，同时保证加密域上的乘法同态。然而，由于加密每个梯度涉及多个指数运算，$ElGamal$不适合由大规模数据驱动的联邦学习。$PPDL$利用了基于LWE的同态加密[35]，其速度比$ElGamal$密码系统快。然而，它的计算/通信开销也随着每个用户的用户/梯度数量的增加而显著增加。与MDL和$PPDL$相比，我们设计了一个双掩码协议来加密用户的本地梯度。由于我们在训练过程中没有考虑用户离线的情况，每个用户n只需要计算一次N_n^{PK}即可。因此，加密操作相当于在每个梯度中加入多个随机值，大大减少了加密过程中的计算和通信开销。



$Fig.17$ 运行时间
$(a)|U| = 100$，每个用户有不同的梯度数量
$(b)|G| = 1000$，用户数量不同

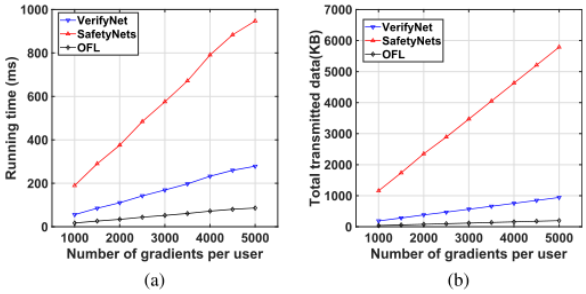


$Fig.18$ 总的传输数据

$(a)|U| = 100$，每个用户有不同的梯度数量

$(b)|G| = 1000$，用户数量不同

2. 验证过程的性能：通过运行$SafetyNets$[16]和OFL[10]进行比较，我们对$VerifyNet$在验证过程中的性能进行了评价。$SafetyNets$实现的场景与我们的$VerifyNet$不同，$VerifyNet$的目的是验证训练过程中服务器返回结果的正确性，只考虑整个系统只有一个用户。为了比较相同实验环境下的开销，我们设置$|U| = 1$，并利用$SafetyNets$的可验证技术来完成我们$VerifyNet$的相同任务。另外，这里我们只考虑没有用户离线的情况。然后记录每个用户不同梯度数的三种方案的运行时间和总传输数据。从$Fig.19$可以看出，$VerifyNet$和$SafetyNets$的成本与原始模型OFL相比很明显。然而，$VerifyNet$的性能明显优于$SafetyNets$。其中一个原因是$SafetyNets$的技术局限性，另一个原因是我们提出的协议中利用了同态哈希和伪随机函数的组合。具体来说，$SafetyNets$使用交互式证明系统[36]检查云服务器返回的计算结果的正确性。它需要与服务器进行多次交互和计算才能完成验证任务，并且在计算和通信开销[37]方面存在缺陷。然而，我们利用与伪随机技术集成的同态哈希函数作为$VerifyNet$的底层结构，以高效地处理数据而著称。因此，我们的$VerifyNet$可以确保用户以相对较低的开销验证云服务器返回结果的正确性。



$Fig.19$ $(a)|U| = 1$，每个用户有不同的梯度数量

$(b)|U| = 1$，每个用户有不同的梯度数量

8. 相关工作

在本节中，我们从隐私保护和可验证性方面介绍深度学习的最新相关工作。

A. 隐私保护深度学习

大多数基于深度学习的隐私保护算法都专注于保护用户的数据隐私。它们的协议中使用的主要工具是差异隐私、安全多方计算[11][18]和加密原语[19]。然而，隐私泄露的问题仍然没有完全解决。例如$Shokri$和$Shmatikov$[18]提出了一种隐私保护的深度学习方法，利用差分隐私来实现安全性和准确性之间的平衡。不幸的是，如果对手利用GAN网络攻击协议，任何基于不同隐私的策略都将暴露为不安全的[14]。$Phong$等人[19]提出了一种利用加性同态加密和异步随机梯度的更安全的深度学习系统，但该实现要求所有用户共享相同的密钥以达到预期的安全级别。最近，$Bonawitz$等人[11]设计了一种联邦深度学习方法，利用安全多方计算保护每个用户的本地梯度，在训练过程中支持用户离线。

B. 可验证深度学习

在深度学习中，云服务器可能会因为意外情况而向用户返回错误的结果。

为了解决这个问题，几个方案[16][17]相继被提出。例如，$Ghodsi$等人[16]设计了一个名为$SafetyNets$的框架。它使用交互式证明系统[36]来检查云服务器返回计算结果的正确性。之后，$Tramer$和$Boneh$[17]提出了一个名为$Slalom$的可验证方案，利用SGX、$TrustZone$和$Sanctum$等可信硬件进行验证。然而，这些方案要么支持少量的激活函数，要么需要额外的硬件辅助。更值得注意的是，据我们所知，对于正在训练的神经网络，没有任何解决方案来支持对云计算结果正确性的验证。与现有方法相比，我们提出了$VerifyNet$，这是第一个在神经网络训练过程中支持验证的隐私保护方法。我们首先利用结合伪随机技术的同态哈希函数来支持每个用户的可验证性。然后，我们使用一种秘密共享技术和密钥协议来保护用户的本地梯度的隐私，并解决用户在训练过程中的离线问题。

9. 总结

在本文中，我们提出了$VerifyNet$，它支持每个用户验证服务器的计算结果。此外，$VerifyNet$支持在训练过程中用户离线。安全分析表明，我们的$VerifyNet$在诚实但好奇的安全设置下具有很高的安全性。另外，在实际数据上进行的实验也证明了我们所提出的方案的实际性能。作为未来研究工作的一部分，我们将专注于减少整个协议的通信开销。