SecureML: A System for Scalable Privacy-Preserving Machine Learning

Payman Mohassel*   Yupeng Zhang†

Abstract

Machine learning is widely used in practice to produce predictive models for applications such as image processing, speech and text recognition. These models are more accurate when trained on large amount of data collected from different sources. However, the massive data collection raises privacy concerns.

机器学习在实践中被广泛用于为诸如图像处理，语音和文本识别的应用产生预测模型。 在对从不同来源收集的大量数据进行训练时，这些模型更加准确。 但是，协助数据收集引起了隐私问题。

In this paper, we present new and efficient protocols for privacy preserving machine learning for linear regression, logistic regression and neural network training using the stochastic gradient descent method. Our protocols fall in the two-server model where data owners distribute their private data among two non-colluding servers who train various models on the joint data using secure two-party computation (2PC). We develop new techniques to support secure arithmetic operations on shared decimal numbers, and propose MPC-friendly alternatives to non-linear functions such as sigmoid and softmax that are superior to prior work.

在本文中，我们提出了新的和有效的隐私保护机器学习协议，用于线性回归，逻辑回归和使用随机梯度下降法的神经网络训练。
我们的协议属于双服务器模型，其中数据所有者在两个非串通服务器之间分发其私有数据，这两个服务器使用安全的双方计算（2PC）在联合数据上训练各种模型。
我们开发新技术以支持对共享十进制数的安全算术运算，并提出MPC友好的替代非线性函数，如sigmoid和softmax，它们优于以前的工作

We implement our system in C++. Our experiments validate that our protocols are several orders of magnitude faster than the state of the art implementations for privacy preserving linear and logistic regressions, and scale to millions of data samples with thousands of features. We also implement the first privacy preserving system for training neural networks.

我们用C ++实现我们的系统。 我们的实验证实，我们的协议比隐私保护线性和逻辑回归的现有技术实现快几个数量级，并且可以扩展到具有数千个特征的数百万个数据样本。 我们还实施了第一个用于训练神经网络的隐私保护系统。

1 Introduction

机器学习技术在实践中被广泛用于产生用于医学，银行业务，推荐服务，威胁分析和认证技术的预测模型。 随着时间的推移收集的大量数据为旧问题提供了新的解决方案，深度学习的进步已经在语音，图像和文本识别方面取得了突破。

Machine learning techniques are widely used in practice to produce predictive models for use in medicine, banking, recommendation services, threat analysis, and authentication technologies. Large amount of data collected over time have enabled new solutions to old problems, and advances in deep learning have led to breakthroughs in speech, image and text recognition.

Large internet companies collect users' online activities to train recommender systems that predict their future interest. Health data from different hospitals, and government organization can be used to produce new diagnostic models, while financial companies and payment networks can combine transaction history, merchant data, and account holder information to train more accurate fraud-detection engines.

大型互联网公司收集用户的在线活动，以训练预测他们未来兴趣的推荐系统。 来自不同医院和政府组织的健康数据可用于生成新的诊断模型，而金融公司和支付网络可以结合交易历史，商家数据和账户持有人信息来训练更准确的欺诈检测引擎。

While the recent technological advances enable more efficient storage, processing and computation on big data, *combining data from different sources* remains an important challenge. Competitive advantage, privacy concerns and regulations, and issues surrounding data sovereignty and jurisdiction prevent many organizations from openly sharing their data. Privacy-preserving machine learning via

*Visa Research. **Email:** `pmohasse@visa.com`.

†University of Maryland. **Email:** `zhangyp@umd.edu`. This work was partially done when the author was interning at Visa Research.

虽然最近的技术进步能够在大数据上实现更高效的存储，处理和计算，但是结合来自不同来源的数据仍然是一项重要挑战。 竞争优势，隐私问题和法规以及围绕数据所有权和使用权的问题阻止了许多组织公开共享其数据。 安全多方计算（MPC）为机器学习隐私保护提供了一种有前景的解决方案，它允许不同的实体在其联合数据上训练各种模型，而不会泄露任何超出结果的信息。

我们专注于训练线性回归，逻辑回归和神经网络模型的机器学习算法，并采用双服务器模型（更多细节见第3节），通常用在通过MPC保护隐私的机器学习系统上[36,35,20]。在此模型中，在设置阶段，数据所有者（客户端）在两个非串通服务器之间处理，加密和/或秘密共享其数据。在计算阶段，两个服务器可以在客户端的联合数据上训练各种模型，而无需学习训练模式之外的任何信息

secure multiparty computation (MPC) provides a promising solution by allowing different entities to train various models on their joint data without revealing any information beyond the outcome.[1]

We focus on machine learning algorithms for training linear regression, logistic regression and neural networks models, and adopt the *two-server* model (see section 3 for more details), commonly used by previous work on privacy-preserving machine learning via MPC [36, 35, 20]. In this model, in a setup phase, the data owners (clients) process, encrypt and/or secret-share their data among two non-colluding servers. In the computation phase, the two servers can train various models on the clients' joint data without learning any information beyond the trained model.

保护隐私线性回归[36,20]的现有技术解决方案比明文训练慢许多个数量级。在先前实现中的效率的主要来源是用于训练的大量计算发生在用于布尔电路的安全2PC内(例如，Yao的乱码电路)，其对表示为整数的十进制数执行算术运算。众所周知，布尔电路不适合执行算术运算，但鉴于现有的固定点或浮点乘法技术需要使用布尔电路最有效的位级操作，它们似乎是不可避免的

The state of the art solutions for privacy preserving linear regression [36, 20] are many orders of magnitude slower than plaintext training. The main source of inefficiency in prior implementations is that the bulk of computation for training takes place inside a secure 2PC for boolean circuits (e.g Yao's garbled circuit) that performs arithmetic operation on decimal numbers represented as integers. It is well-known that boolean circuits are not suitable for performing arithmetic operations, but they seem unavoidable given that existing techniques for fixed-point or floating-point multiplication require bit-level manipulations that are most efficient using boolean circuits.

在逻辑回归和神经网络的情况下，问题甚至更具挑战性，因为训练过程计算许多非线性激活函数的实例，例如在2PC内计算昂贵的sigmoid和sof-tmax。实际上，我们不知道这两种训练算法的任何隐私保护实现。

In case of logistic regression and neural networks, the problem is even more challenging as the training procedure computes many instances of non-linear activation functions such as sigmoid and softmax that are expensive to compute inside a 2PC. Indeed, we are not aware of any privacy preserving implementations for these two training algorithms.

1.1 Our Contributions

我们设计了新的和有效的协议，用于在上面讨论的双服务器模型中保护隐私线性回归，逻辑回归和神经网络训练，假设跨客户端对数据集进行任意划分。

We design new and efficient protocols for privacy preserving linear regression, logistic regression and neural networks training in the two-server model discussed above assuming an arbitrary partitioning of the dataset across the clients.

我们的隐私保护线性回归协议比针对同一问题的现有技术解决方案高出几个数量级。例如，对于具有100, 000个样本和500个特征的数据集，并且在可比较的设置和实验环境中，我们的协议比[36,20]中实现的协议快1100-1300倍。此外，正如我们的实验所示，我们显着缩小了隐私保护和明文训练之间的差距。

Our privacy preserving linear regression protocol is *several orders of magnitude* more efficient than the state of the art solutions for the same problem. For example, for a dataset with 100,000 samples and 500 features and in a comparable setup and experimental environment, our protocol is 1100-1300× faster than the protocols implemented in [36, 20]. Moreover, as our experiments show, we significantly reduce the gap between privacy-preserving and plaintext training.

我们还实现了第一个用于逻辑回归和神经网络训练的隐私保护协议，具有很高的效率。例如，在具有784个特征的大小为60,000的数据集上，我们的隐私保护逻辑回归的总运行时间为29秒，而我们用于训练具有3层和266个神经元的神经网络的隐私保护协议在21,000s内运行。

We also implement the first privacy preserving protocols for logistic regression and neural networks training with high efficiency. For example, on a dataset of size 60,000 with 784 features, our privacy preserving logistic regression has a total running time of 29s while our privacy-preserving protocol for training a neural network with 3 layers and 266 neurons runs in 21,000s.

我们的协议自然分为数据无关的off-line阶段和更快的online阶段。当排除offline阶段时，协议在明文训练中更具竞争力。例如，对于具有60000个样本和784个特征的数据集，并且在LAN设置中，线性回归协议在1.4s中运行，逻辑回归在8.9s中运行，神经网络训练在653.0s中运行

Our protocols are naturally divided into a data-independent offline phase and a much faster online phase. When excluding the offline phase, the protocols are even more competitive with plaintext training. For instance, for a dataset with 60,000 samples and 784 features, and in the LAN setting, the linear regression protocol runs in 1.4s, the logistic regression in 8.9s, and the neural network training in 653.0s.

共享十进制数的算运算

如前所述，先前工作的一个主要瓶颈是计算安全2PC内的定点运算，例如乱码电路。考虑到训练所需的大量乘法，这非常昂贵

Arithmetic on shared decimal numbers. As mentioned earlier, a major bottleneck in prior work is the computation of fixed-point arithmetic inside a secure 2PC such as garbled circuits. This is prohibitively expensive, given the large number of multiplications needed for training.

Fixed-point addition is fairly straightforward. For multiplication, we show that the following strategy is very effective: represent the two shared decimal numbers as shared integers in a finite

[1]In the more general variant of our protocols, even the model can remain private (secret shared).

定点加法相当简单。对于乘法，我们证明以下策略非常有效：将两个共享十进制数表示为有限字段中的共享整数; 使用offline生成的乘法三元组对共享整数执行乘法运算; 让每一方截断其结果，以便固定数量的位代表小数部分。

我们证明，与定点运算相比，从这些截断的共享部分重建时，运算结果很大概率上只是在在分数部分的最不重要位置最多为1位。我们对两个不同数据集MNIST和Arcene [6,1]的实验证实，当表示小数部分的位数很大时，小截断误差对训练模型的准确性没有影响（实际上精度与标准训练的准确度相匹配）。因此，隐私保护线性回归的online阶段不涉及任何加密操作，仅包括整数乘法和位移，而offline阶段包括生成必要的乘法三元组。我们的微基准测试表明，即使在考虑总时间（online和offline组合）时，与使用乱码电路的固定点乘法相比，我们的方法产生了4-8倍的改善。

field; perform a multiplication on shared integers using offline-generated multiplication triplets; have each party truncate its share of the product so that a fixed number of bits represent the fractional part. We prove that, with high probability, the product when reconstructed from these truncated shares, is at most 1 bit off in the least significant position of the fractional part compared to fixed-point arithmetic. Our experiments on two different datasets, MNIST and Arcene [6, 1], confirm that the small truncation error has no effect on accuracy of the trained model (in fact accuracies match those of standard training) when the number of bits representing the fractional part is sufficiently large. As a result, the online phase for privacy preserving linear regression does not involve any cryptographic operations and only consists of integer multiplications and bit shifting, while the offline phase consists of generating the necessary multiplication triplets. Our microbenchmarking shows that even when considering total time (online and offline combined) our approach yields a factor of 4-8× improvement compared to fixed-point multiplication using garbled circuits.

如前所述，逻辑回归和神经网络训练需要计算逻辑sigmoid和softmax函数，这些函数在共享值上计算起来很昂贵。我们通过实验证明，使用低次多项式逼近逻辑函数是无效的。

MPC友好的激活函数

MPC-friendly activation functions. As discussed earlier, logistic regression and neural network training require computing the logistic ($\frac{1}{1+e^{-x}}$), and the softmax ($\frac{e^{-x_i}}{\sum e^{-x_i}}$) functions which are expensive to compute on shared values. We experimentally show that the use of low-degree polynomials to approximate the logistic function is ineffective. In particular, one needs polynomials of degree at least 10 to approach the accuracy of training using the logistic function. We propose a new activation function that can be seen as the sum of two RELU functions (see Figure 7), and computed efficiently using a small garbled circuit. Similarly, we replace the softmax function with a combination of RELU functions, additions and a single division. Our experiments using the MNIST, and Arcene datasets confirm that accuracy of the models produced using these new functions either match or are very close to those trained using the original functions.

特别地，需要至少为10阶的多项式才能接近使用逻辑函数训练的准确性。我们提出了一种新的激活函数，它可以看作是两个RELU函数的和(见图7)，并且使用一个小的乱码电路进行了有效的计算。类似地，我们用RELU函数，添加和单个分区的组合替换softmax函数。我们使用MNIST和Arcene数据集的实验证实，使用这些新函数生成的模型的精度与使用原始函数训练的模型匹配或非常接近。

然后，我们提出了一种用于在算术共享和姚-共享之间切换的定制解决方案，并且对于我们的特定计算，我们通过最小化交互轮次和被调用的不经意传输(OT)的数量来显着降低成本。我们在6.5节中的微基准测试表明，评估我们的新函数的时间比用高次多项式逼近逻辑函数要快得多。

We then propose a customized solution for switching between arithmetic sharing and Yao sharing, and back, for our particular computation, that significantly reduces the cost by minimizing rounds of interaction and number of invoked oblivious transfers (OT). Our microbenchmarking in Section 6.5 shows that the time to evaluate our new function is much faster than to approximate the logistic function with a high degree polynomial.

我们使用相同的思想来安全地评估神经网络训练中使用的RELU函数。

We use the same ideas to securely evaluate the RELU functions used in neural networks training.

矢量化协议

Vectorizing the protocols. Vectorization, i.e. operating on matrices and vectors, is critical in efficiency of plaintext training. We show how to benefit from the same vectorization techniques in the shared setting. For instance, in the offline phase of our protocols which consists of generating many multiplication triplets, we propose and implement two solutions based on linearly homomorphic encryption (LHE) and oblivious transfer. The techniques are inspired by prior work (e.g., [17]) but are optimized for our vectorized scenario where we need to compute multiplication of shared matrices and vectors. As a result the complexity of our offline protocols is much better than the naive approach of generating independent multiplication triplets for each multiplication. In particular, the performance of the OT-based multiplication triplets generation is improved by a factor of 4×, and the LHE-based generation is improved by 41-66×.

矢量化，即在矩阵和矢量上运行，对于明文训练的效率至关重要。我们展示了如何在共享设置中使用相同的矢量化技术。例如，在我们的协议的offline阶段，包括生成许多乘法三元组，我们提出并实现了两个基于线性同态加密(LHE)和不经意传输的解决方案。

这些技术受到先前工作的启发（例如，[17]），但针对我们需要计算共享矩阵和向量的乘法的矢量化场景进行了优化。因此，我们的offline协议的复杂性比为每次乘法生成独立乘法三元组的朴素方法要好得多。特别是，比基于OT的乘法三元组生成提高了4倍，比基于LHE的生成提高了41-66倍

In a different security model similar to [20], we also propose a much faster offline phase where clients help generate the multiplication triplets. This provides a weaker security gauarantee than our standard setting. In particular, it requires the additional assumption that servers and clients do not collude, i.e. an attacker either corrupts a server or a subset of clients but not both. We discuss pros/cons of this approach and compare its performance with the standard approach in Section 5.

在类似于[20]的不同安全模型中，我们还提出了一个更快的离线阶段，其中客户端帮助生成乘法三元组。这提供了比我们的标准设置更弱的安全性。特别是，它需要额外的假设，即服务器和客户端不会串通，即攻击者破坏服务器或客户端子集，但不会破坏两者。我们讨论了这种方法的优缺点，并将其性能与第5节中的标准方法进行了比较。

1.2 Related Work

早期关于隐私保护机器学习的工作主要集中在决策树[30]，k-均值聚类[27,13]，SVM分类[47,43]，线性回归[18,19,39]和逻辑回归[41]。这些论文提出了基于安全多方计算的解决方案，但似乎会产生高效率开销并且缺乏实施/评估。

Earlier work on privacy preserving machine learning has focused on decision trees [30], k-means clustering [27, 13], SVM classification [47, 43], linear regression [18, 19, 39] and logistic regression [41]. These papers propose solutions based on secure multiparty computation, but appear to incur high efficiency overheads and lack implementation/evaluation.

Nikolaenko et. al. [36]使用LHE和乱码电路的组合在水平分区数据上呈现隐私保护线性回归协议，并在具有数百万个样本的数据集上对其进行评估

Gascon et. al. [20]将结果扩展到垂直分区数据并显示出改进的性能。然而，这两篇论文都减少了使用Yao的乱码电路协议解决线性系统的问题，这会在训练时间上引入很高的开销，并且不能推广到非线性模型。相比之下，我们使用随机梯度下降法，它可以训练非线性模型，如逻辑回归和神经网络。

最近，Gilad-Bachrach et. al.[22]提出了一个安全数据交换的框架，并支持隐私保护线性回归作为一个应用程序。但是，只测试了小型数据集，并且纯粹使用乱码电路实现协议，这不适用于较大的数据集

Nikolaenko et. al. [36] present a privacy preserving linear regression protocol on horizontally partitioned data using a combination of LHE and garbled circuits, and evaluate it on datasets with millions of samples. Gascon et. al. [20] extend the results to vertically partitioned data and show improved performance. However, both papers reduce the problem to solving a linear system using Yao's garbled circuit protocol, which introduces a high overhead on the training time and cannot be generalized to non-linear models. In contrast, we use the stochastic gradient descent method which enables training non-linear models such as logistic regression and neural networks. Recently, Gilad-Bachrach et. al. [22] propose a framework for secure data exchange, and support privacy preserving linear regression as an application. However, only small datasets are tested and the protocol is implemented purely using garbled circuit, which does not scale for larger datasets.

Wu et. al.[45]认为隐私保留逻辑回归。他们建议使用多项式逼近逻辑函数，并使用LHE训练模型。然而，复杂度在近似多项式的程度上是指数的，并且正如我们将在实验中所示，与使用逻辑函数相比，模型的准确性降低。

Aonoet. al. [9]考虑一种不同的安全模型，其中不受信任的服务器收集并组合来自多个客户端的加密数据，并将其传输到可信客户端以在明文上训练模型。通过用2次多项式仔细逼近逻辑回归的代价函数，可以通过求解线性系统来计算最优模型。但是，在此设置中，聚合数据的明文泄露给训练模型的客户端。我们不知道在双服务器模型中使用实用的隐私保护逻辑回归系统的任何先前工作。

Privacy preserving logistic regression is considered by Wu et. al. [45]. They propose to approximate the logistic function using polynomials, and train the model using LHE. However, the complexity is exponential in the degree of the approximation polynomial, and as we will show in experiments, the accuracy of the model is degraded compared to using the logistic function. Aono et. al. [9] consider a different security model where an untrusted server collects and combines the encrypted data from multiple clients, and transfers it to a trusted client to train the model on the plaintext. By carefully approximating the cost function of logistic regression with a degree 2 polynomial, the optimal model can be calculated by solving a linear system. However, in this setting, the plaintext of the aggregated data is leaked to the client who trains the model. We are not aware of any prior work with a practical system for privacy preserving logistic regression in the two-server model.

使用神经网络进行隐私保护机器学习更具挑战性。Shokri和Shmatikov [40]提出了一种解决方案，在这种解决方案中，两台服务器不是共享数据，而是在训练期间共享部分系数的变化。尽管该系统非常有效（根本不需要加密操作），但这些系数变化的泄漏并未得到充分理解，也没有获得正式的安全保证。此外，他们的方法仅适用于水平分区数据，因为每个服务器需要能够在其部分上单独执行训练，以便获得系数的变化。

Gilad-Bachrach et. al.[21]最近也研究了使用神经网络的隐私保护预测。使用完全同态加密，神经网络模型可以对加密数据进行预测。在这种情况下，假设神经网络是在明文数据上训练的，并且该模型对于在另一方的私人数据上评估它的一方是已知的。

Privacy preserving machine learning with neural networks is more challenging. Shokri and Shmatikov [40] propose a solution where instead of sharing the data, the two servers share the changes on a portion of the coefficients during the training. Although the system is very efficient (no cryptographic operation is needed at all), the leakage of these coefficient changes is not well-understood and no formal security guarantees are obtained. In addition, their approach only works for horizontally partitioned data since each server needs to be able to perform the training individually on its portion in order to obtain the coefficient changes. Privacy preserving predictions using neural networks were also studied recently by Gilad-Bachrach et. al. [21]. Using fully homomorphic encryption, the neural network model can make predictions on encrypted data. In this case, it is assumed that the neural network is trained on plaintext data and the model is known to one party who evaluates it on private data of another.

An orthogonal line of work considers the differential privacy of machine learning algorithms [15, 42, 8]. In this setting, the server has full access to the data in plaintext, but wants to guarantee that the released model cannot be used to infer the data used during the training. A common technique used in differentially private machine learning is to introduce an additive noise to the data or the update function (e.g., [8]). The parameters of the noise are usually predetermined by the dimensions of the data, the parameters of the machine learning algorithm and the security requirement, and hence are data-independent. Our system can be composed with such constructions given that the servers can always generate the noise according to the public parameters and add it directly onto

正交工作线考虑了机器学习算法的不同隐私[15,42,8]。在此设置中，服务器可以完全访问纯文本中的数据，但希望保证发布的模型不能用于推断训练期间使用的数据。在不同的机器学习中的常用技术是向数据或更新函数引入加噪声(例如，[8])。噪声的参数通常由数据的尺寸，机器学习算法的参数和安全要求预先确定，因此是与数据无关的。

考虑到服务器总是可以根据公共参数生成噪声并将其直接添加到训练中的共享值中，我们的系统可以由这样的结构组成。通过这种方式，经过训练的模型在重建后将基本上是私有的，而所有数据在训练期间仍然保持私密。

the shared values in the training. In this way, the trained model will be differentially private once reconstructed, while all the data still remains private during the training.

2 Preliminaries

初步措施

2.1 Machine Learning

在本节中，我们简要回顾一下本文考虑的机器学习算法：线性回归，逻辑回归和神经网络。 我们提出的所有算法都是经典的，可以在标准的机器学习教科书中找到（比如[25]）

给定n个训练数据样本xi，每个训练数据样本xi包含d个特征和相应的输出标记yi，回归是学习函数g的统计过程，使得g（xi）≈yi。回归在现实生活中有很多应用。例如，在医学科学中，它被用来学习疾病和代表性特征之间的关系，例如年龄，体重，饮食习惯，并将其用于诊断目的

In this section, we briefly review the machine learning algorithms considered in this paper: linear regression, logistic regression and neural networks. All algorithms we present are classic and can be found in standard machine learning textbooks (e.g., [25]).

线性回归

Linear regression Given n training data samples $\mathbf{x}_i$ each containing d features and the corresponding output labels y_i, *regression* is a statistical process to learn a function g such that $g(\mathbf{x}_i) \approx y_i$. Regression has many applications in real life. For example, in medical science, it is used to learn the relationship between a disease and representative features, such as age, weight, diet habits and use it for diagnosing purposes.

在线性回归中，假设函数g是线性的，并且可以表示为xi的内积与系数向量w

In linear regression, the function g is assumed to be linear and can be represented as the inner product of $\mathbf{x}_i$ with the coefficient vector $\mathbf{w}$: $g(\mathbf{x}_i) = \sum_{j=1}^{d} x_{ij} w_j = \mathbf{x}_i \cdot \mathbf{w}$, where x_{ij} (resp. w_j) is the jth value in vector $\mathbf{x}_i$ (resp. $\mathbf{w}$), and $\cdot$ denotes the inner product of two vectors.[2]

要学习系数向量w

To learn the coefficient vector $\mathbf{w}$, a *cost function* $C(\mathbf{w})$ is defined and $\mathbf{w}$ is calculated by the optimization $\text{argmin}_{\mathbf{w}} C(\mathbf{w})$. In linear regression, a commonly used cost function is $C(\mathbf{w}) = \frac{1}{n}\sum C_i(\mathbf{w})$, where $C_i(\mathbf{w}) = \frac{1}{2}(\mathbf{x}_i \cdot \mathbf{w} - y_i)^2$. [3]

The solution for this optimization problem can be computed by solving the linear system $(\mathbf{X}^T \times \mathbf{X}) \times \mathbf{w} = \mathbf{X}^T \times \mathbf{Y}$, where $\mathbf{X}$ is a $n \times d$ matrix representing all the input data, and $\mathbf{Y}$ is a $n \times 1$ matrix for the output labels. However, the complexity of the matrix multiplication $\mathbf{X}^T \times \mathbf{X}$ is $O(nd^2)$ and the complexity of solving the linear system is $O(d^3)$. Due to its high complexity, it is rarely used in practice except for small values of n and d.

由于其高复杂性，除了n和d的小值之外，它在实践中很少使用。

SGD是一种有效的近似算法，用于逐步逼近函数的局部最小值。由于上述线性回归的优化函数是凸的，因此SGD可证明地收敛于全局最小值并且在实践中通常非常快。此外，SGD可以推广用于逻辑回归和神经网络训练，其中对于相应的优化问题不存在封闭形式的解决方案。因此，SGD是在实践中训练此类模型的最常用方法，也是这项工作的主要重点。

随机梯度下降

Stochastic gradient descent (SGD) SGD is an effective approximation algorithm for approaching a local minimum of a function, step by step. As the optimization function for the linear regression described above is *convex*, SGD provably converges to the global minimum and is typically very fast in practice. In addition, SGD can be generalized to work for logistic regression and neural network training, where no closed-form solution exists for the corresponding optimization problems. As a result, SGD is the most commonly used approach to train such models in practice and the main focus of this work.

The SGD algorithm works as follows: $\mathbf{w}$ is initialized as a vector of random values or all 0s. In each iteration, a sample $(\mathbf{x}_i, y_i)$ is selected randomly and a coefficient w_j is updated as

w被初始化为随机值或全0的向量。 在每次迭代中，随机选择样本(xi,yi)，并将系数wj更新为

$$w_j := w_j - \alpha \frac{\partial C_i(\mathbf{w})}{\partial w_j}. \tag{1}$$

[2]Usually a bias b is introduced such that $g(x_i) = x_i \cdot w + b$. However, this can be easily achieved by appending a dummy feature equal to 1 for each x_i. To simplify the notation, we assume b is already embedded in w in this paper.

[3]In *ridge regression*, a penalty term $\lambda||\mathbf{w}||^2$ is added to the cost function to avoid overfitting where λ is the regularization parameter. This is supported in an obvious way by the protocols in this paper, and is omitted for simplicity.

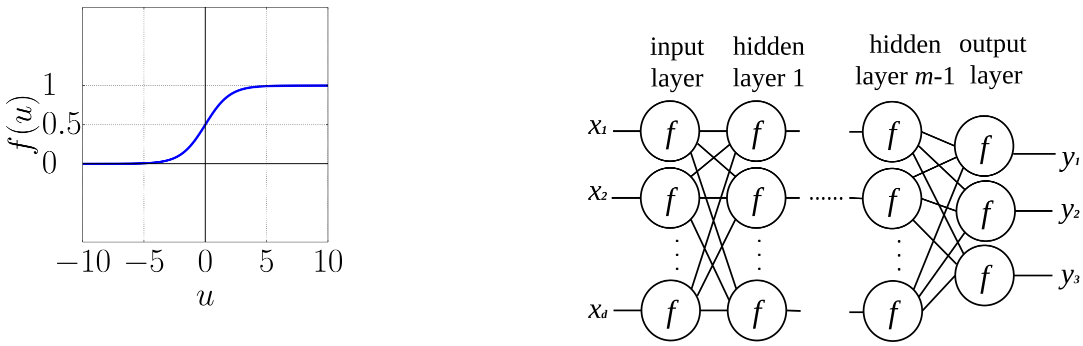


Figure 1: (a) Logistic function. (b) An example of neural network.

其中α是学习率，用于定义每次迭代中向最小值移动的幅度。代替线性回归的成本函数，公式变为wj

计算预测输出的阶段称为前向传播，计算权重变化的阶段叫做反向传播

where α is a learning rate defining the magnitude to move towards the minimum in each iteration. Substituting the cost function of linear regression, the formula becomes $w_j := w_j - \alpha(\mathbf{x}_i \cdot \mathbf{w} - y_i)x_{ij}$. The phase to calculate the predicted output $y_i^* = \mathbf{x}_i \cdot \mathbf{w}$ is called *forward propagation*, and the phase to calculate the change $\alpha(y_i^* - y_i)x_{ij}$ is called *backward propagation*.

小批量

实际上，不是每次迭代选择一个数据样本，而是随机选择一小批样本，并通过平均当前w上所有样本的偏导数来更新w。

我们用B表示在小批量中选择的一组索引。这称为小批量SGD和|B| 表示小批量大小，通常在2到200之间。小批量的好处是可以使用矢量化库来加速计算，使得一个小批量的计算时间比运行|B|快得多。迭代没有小批量。此外，通过小批量，w可以更平滑，更快速地收敛。使用小批量，更新功能可以以矢量化形式表示：

Mini-batch. In practice, instead of selecting one sample of data per iteration, a small batch of samples are selected randomly and $\mathbf{w}$ is updated by averaging the partial derivatives of all samples on the current $\mathbf{w}$. We denote the set of indices selected in a mini-batch by B. This is called a *mini-batch* SGD and $|B|$ denotes the mini-batch size, usually ranging from 2 to 200. The benefit of mini-batch is that vectorization libraries can be used to speed up the computation such that the computation time for one mini-batch is much faster than running $|B|$ iterations without mini-batch. Besides, with mini-batch, $\mathbf{w}$ converges smoother and faster to the minimum. With mini-batch, the update function can be expressed in a vectorized form:

$$\mathbf{w} := \mathbf{w} - \frac{1}{|B|}\alpha \mathbf{X}_B^T \times (\mathbf{X}_B \times \mathbf{w} - \mathbf{Y}_B). \tag{2}$$

$\mathbf{X}_B$ and $\mathbf{Y}_B$ are $B \times d$ and $B \times 1$ submatrices of $\mathbf{X}$ and $\mathbf{Y}$ selected using indices in B, representing $|B|$ samples of data and labels in an iteration. Here $\mathbf{w}$ is viewed as a column vector.

学习率调整

如果学习率α太大，则SGD的结果可能偏离最小值。因此，测试数据集用于测试当前w的准确性。w的内积和测试数据集中的每个数据样本被计算为预测，并与相应的标签进行比较。准确度是测试数据集上正确预测的百分比。如果准确度在下降，则学习率会降低，训练将以新的学习率开始。

为了平衡测试所花费的开销，通常的做法是选择所有训练样本并按顺序选择每个迭代中的小批量，直到所有样本都使用一次。这被称为一个时代。在一个时期之后，测试当前w的准确度。此时，如果准确度降低，则学习率降低一半并且训练开始；否则数据将被删除，并执行下一个训练时期。

Learning rate adjustment. If the learning rate α is too large, the result of SGD may diverge from the minimum. Therefore, a testing dataset is used to test the accuracy of the current $\mathbf{w}$. The inner product of $\mathbf{w}$ and each data sample in the testing dataset is calculated as the prediction, and is compared to the corresponding label. The accuracy is the percentage of the correct predictions on the testing dataset. If the accuracy is decreasing, the learning rate is reduced and the training starts over with the new learning rate. To balance the overhead spent on testing, the common practice is to shuffle all the training samples and select the mini-batch in each iteration sequentially, until all the samples are used once. This is referred to as one *epoch*. After one epoch, the accuracy of the current $\mathbf{w}$ is tested. At this point, if the accuracy decreases, the learning rate is reduced by half and the training starts over; otherwise the data is reshuffled and the next epoch of training is executed.

Termination. When the difference in accuracy compared to the previous epoch is below a small threshold, $\mathbf{w}$ is viewed as having converged to the minimum and the algorithm terminates. We denote the number of epochs to train a model as E and denote the total number of iterations as t. Note that we have the following relationship: $n \cdot E = |B| \cdot t$.

终止。当与前一个时期相比精度的差异低于小阈值时，w被视为已收敛到最小值并且算法终止。我们将模型训练为E的时期数表示，并将迭代总数表示为t。请注意，我们有以下关系：

Logistic回归在两个类的分类问题中，输出标签y是二进制的。 例如，鉴于一些医学特征，我们有兴趣预测患者是健康还是生病。 在这种情况下，最好将预测的输出限制在0和1之间。

如图1（a）所示，逻辑函数的两个尾部收敛于0和1.使用此激活函数，线性回归的原始成本函数不再是凸的，因此应用SGD可能会给出局部最小值而不是全局最小值

因此，激活函数f应用于内积之上，并且该关系表示为

在逻辑回归中，激活函数被定义为逻辑函数

因此，成本函数改变为交叉熵函数

Logistic Regression In classification problems with two classes, the output label y is binary. E.g., given some medical features, we are interested to predict whether the patient is healthy or sick. In this case, it is better to bound the output of the prediction between 0 and 1. Therefore, an *activation function* f is applied on top of the inner product and the relationship is expressed as: $g(\mathbf{x}_i) = f(\mathbf{x}_i \cdot \mathbf{w})$. In logistic regression, the activation function is defined as the logistic function $f(u) = \frac{1}{1+e^{-u}}$. As shown in Figure 1(a), the two tails of the logistic function converge to 0 and 1.

With this activation function, the original cost function for linear regression is no longer convex, thus applying SGD may give a local minimum instead of the global minimum. Therefore, the cost function is changed to the *cross entropy* function $C_i(\mathbf{w}) = -y_i \log y_i^* - (1 - y_i) \log(1 - y_i^*)$ and $C(\mathbf{w}) = \frac{1}{n} \sum C_i(\mathbf{w})$, where $y_i^* = f(\mathbf{x}_i \cdot \mathbf{w})$.

The mini-batch SGD algorithm for logistic regression updates the coefficients in each iteration as follows: 用于逻辑回归的小批量SGD算法更新每次迭代中的系数

$$\mathbf{w} := \mathbf{w} - \frac{1}{|B|} \alpha \mathbf{X}_B^T \times (f(\mathbf{X}_B \times \mathbf{w}) - \mathbf{Y}_B). \tag{3}$$

请注意，逻辑回归的向后传播与线性回归具有完全相同的形式，但它是使用不同的激活和成本函数导出的。SGD中逻辑回归的唯一不同之处是在前向传播中对内积应用额外的逻辑函数。

Notice that the backward propagation of logistic regression has exactly the same form as linear regression, yet it is derived using a different activation and cost function. The only difference in the SGD for logistic regression is to apply an extra logistic function on the inner product in the forward propagation.

神经网络。 神经网络是回归的推广，用于学习高维输入和输出数据之间更复杂的关系。 它广泛应用于图像处理，语音和文本识别等广泛领域，通常可以在每个领域取得突破。 图1（b）显示了具有m-1个隐藏层的神经网络的示例。 隐藏层和输出层中的每个节点是回归的实例，并且与激活函数和系数向量相关联。

Neural Networks. Neural networks are a generalization of regression to learn more complicated relationships between high dimensional input and output data. It is extensively used in a wide range of areas such as image processing, voice and text recognition, often leading to breakthroughs in each area. Figure 1(b) shows an example of a neural network with $m-1$ hidden layers. Each node in the hidden layer and the output layer is an instance of regression and is associated with an activation function and a coefficient vector. Nodes are also called *neurons*. Popular activation functions include the logistic and the RELU function ($f(u) = \max(0, u)$).

节点也称为神经元。 流行的激活功能包括logistic和RELU功能

对于具有多个类别的分类问题，通常在输出层应用softmax函数，其中dm表示输出层中的神经元总数。 洞察力是softmax函数之后的输出始终是概率分布：每个输出介于0和1之间，所有输出总和为1。

For classification problems with multiple classes, usually a *softmax* function $f(u_i) = \frac{e^{-u_i}}{\sum_{i=1}^{d_m} e^{-u_i}}$ is applied at the output layer, where d_m denotes the total number of neurons in the output layer. The insight is that the output after the softmax function is always a probability distribution: each output is between 0 and 1 and all the outputs sum up to 1.

为了使用SGD训练神经网络，在每次迭代中应用等式1来更新所有神经元的所有系数，其中每个神经元被处理类似于回归。 特别地，令di是层i中的神经元的数量，并且d0 = d是输入数据中的特征的数量。 dm是输出的维度。

我们将第i层的系数矩阵表示为di-1×di矩阵Wi，并且将值表示为|B |×di矩阵Xi。 X0初始化为XB。 在每次迭代的前向传播中，第i层的矩阵Xi被计算为Xi = f（Xi-1×Wi）

在后向传播中，给定诸如交叉熵函数的成本函数，每个神经元中的每个系数的更新函数可以以闭合形式表示。

To train a neural network using SGD, Equation 1 is applied in every iteration to update all coefficients of all neurons where each neuron is treated similar to a regression. In particular, let d_i be the number of neurons in layer i and $d_0 = d$ be the number of features in the input data. d_m is the dimension of the output. We denote the coefficient matrix of the ith layer as a $d_{i-1} \times d_i$ matrix $\mathbf{W}_i$, and the values as a $|B| \times d_i$ matrix $\mathbf{X}_i$. $\mathbf{X}_0$ is initialized as $\mathbf{X}_B$. In the forward propagation for each iteration, the matrix $\mathbf{X}_i$ of the ith layer is computed as $\mathbf{X}_i = f(\mathbf{X}_{i-1} \times \mathbf{W}_i)$. In the backward propagation, given a cost function such as the cross entropy function, the update function for each coefficient in each neuron can be expressed in a closed form. To calculated it, we compute the vectors $\mathbf{Y}_i = \frac{\partial C(\mathbf{W})}{\partial \mathbf{U}_i}$ iteratively, where $\mathbf{U}_i = \mathbf{X}_{i-1} \times \mathbf{W}_i$. $\mathbf{Y}_m$ is initialized to $\frac{\partial C}{\partial \mathbf{X}_m} \odot \frac{\partial f(\mathbf{U}_m)}{\partial \mathbf{U}_m}$, where $\frac{\partial f(\mathbf{U}_m)}{\partial \mathbf{U}_m}$ is simply the derivative of the activation function, and $\odot$ is the element-wise product. By the chain rule, $\mathbf{Y}_i = (\mathbf{Y}_{i+1} \times \mathbf{W}_i^T) \odot \frac{\partial f(\mathbf{U}_i)}{\partial \mathbf{U}_i}$. Finally, the coefficients are updated by letting $\mathbf{W}_i := \mathbf{W}_i - \frac{\alpha}{|B|} \cdot \mathbf{X}_i \times \mathbf{Y}_i$.

为了计算它，我们计算分别向量Yi，其中Ui=Xi-1*Wi，Ym被初始化为激活函数的偏导数的逐元素乘，链式规则计算

Parameters: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$.
Main: On input $(SELECT, sid, b)$ from $\mathcal{R}$ and $(SEND, sid, x_0, x_1)$ from $\mathcal{S}$, return $(RECV, sid, x_b)$ to $\mathcal{R}$.

Figure 2: $\mathcal{F}_{ot}$ Ideal Functionality

安全计算

2.2 Secure Computation

不经意传输。 不经意传输（OT）是一种基本的加密原语，通常用作MPC中的构建块。 在不经意的传输协议中，发送方S具有两个输入x0和x1，并且接收方R具有选择位b并且想要获得xb而不学习任何其他内容或者通过S的b 。 图2描述了这种协议实现的理想功能

OT扩展[26,10]通过允许发送器和接收器以λ基本OT(具有公钥操作)和O(m)快速对称密钥的成本执行m个OT来最小化该成本，其中λ是安全性参数。 我们的实现利用OT扩展来提高效率。 我们还使用了一种特殊的OT扩展，称为相关OT扩展[10]。在我们用COT表示的这个变体中，发送者对每个OT的两个输入不是独立的。 相反，每个OT实例的两个输入是

Oblivious Transfer. Oblivious transfer (OT) is a fundamental cryptographic primitive that is commonly used as building block in MPC. In an oblivious transfer protocol, a sender $\mathcal{S}$ has two inputs x_0 and x_1, and a receiver $\mathcal{R}$ has a selection bit b and wants to obtain x_b without learning anything else or revealing b to $\mathcal{S}$. Figure 2 describes the ideal functionality realized by such a protocol. We use the notation $(\perp; x_b) \leftarrow \mathsf{OT}(x_0, x_1; b)$ to denote a protocol realizing this functionality.

表示实现此功能的协议

我们使用OT作为我们的离线协议的一部分，用于生成乘法三元组，并在逻辑回归和神经网络训练的在线阶段，以便安全地计算激活函数。 可以使用[38]的协议实现一轮OT，但它需要双方的公钥操作

We use OTs both as part of our offline protocol for generating multiplication triplets and in the online phase for logistic regression and neural network training in order to securely compute the activation functions. One-round OT can be implemented using the protocol of [38], but it requires public-key operations by both parties. OT extension [26, 10] minimizes this cost by allowing the sender and receiver to perform m OTs at the cost of λ base OTs (with public-key operations) and $O(m)$ fast symmetric-key ones, where λ is the security parameter. Our implementations take advantage of OT extension for better efficiency. We also use a special flavor of OT extension called correlated OT extension [10]. In this variant which we denote by COT, the sender's two inputs to each OT are not independent. Instead, the two inputs to each OT instance are: a random value s_0 and a value $s_1 = f(s_0)$ for a correlation function f of the sender's choice. The communication for a COT of l-bit messages, denoted by COT_l, is $\lambda + l$ bits, and the computation consists of 3 hashing.

对于发送者选择的相关函数f，随机值s0和值s1 = f(s0)由COT,表示的用于l比特消息的COT的通信是λ+1比特，并且计算由3个散列组成。

乱码电路2PC。[46]首先介绍了乱码电路

Garbled Circuit 2PC. Garbled Circuits were first introduced by [46]. A garbling scheme consists of a garbling algorithm that takes a random seed σ and a function f and generates a garbled circuit F and a decoding table dec; the encoding algorithm takes input x and the seed σ and generates garbled input $\widehat{x}$; the evaluation algorithm takes $\widehat{x}$ and F as input and returns the garbled output $\widehat{z}$; and finally, a decoding algorithm that takes the decoding table dec and $\widehat{z}$ and returns $f(x)$. We require the garbling scheme to satisfy the standard security properties formalized in [12].

一种乱码方案包括一个带有随机种子σ和函数f的乱码算法，并产生一个乱码电路F和一个解码表dec; 编码算法采用输入x和种子σ并生成乱码输入X'

评估算法以X'和F为输入，返回乱码输出Z'; 最后，解码算法采用解码表dec和Z'并返回f(x) 我们要求乱码方案满足[12]中规定的标准安全属性

Given such a garbling scheme, it is possible to design a secure two-party computation protocol as follows: Alice generates a random seed σ and runs the garbling algorithm for function f to obtain a garbled circuit GC. She also encodes her input $\widehat{x}$ using σ and x as inputs to the encoding algorithm. Alice sends GC and $\widehat{x}$ to Bob. Bob obtains his encoded (garbled) input $\widehat{y}$ using an oblivious transfer for each bit of y[4]. He then runs the evaluation algorithm on $GC, \widehat{x}, \widehat{y}$ to obtain the garbled output $\widehat{z}$. We can have Alice, Bob, or both learn an output by communicating the decoding table accordingly. The above protocol securely realizes the ideal functionality $\mathcal{F}_f$ that simply takes the parties inputs and computes f on them. See [31] for a more detailed description and proof of security against a semi-honest adversary. In our protocols, we denote this garbled circuit 2PC by $(z_a, z_b) \leftarrow \mathsf{GarbledCircuit}(x; y, f)$

Secret Sharing and Multiplication Triplets. In our protocols, all intermediate values are secret-shared between the two servers. We employ three different sharing schemes: Additive sharing,

[4]While and OT-based encoding is not a required property of a garbling scheme, all existing constructions permit such interacting encodings

给定这样的乱码方案，可以如下设计安全的双方计算协议：Alice生成随机种子σ并运行函数f的乱码算法以获得乱码电路GC。 她还使用σ和x对输入X'进行编码，作为编码算法的输入。 Alice将GC和X'发送给Bob。 对于y的每个比特，Bob使用不经意传输获得他的编码（乱码）输入Y'。 然后他在GC,X',Y'上运行评估算法以获得乱码输出Z'。 我们可以让Alice，Bob或两者通过相应地传送解码表来学习输出。
上述协议安全地实现了理想的功能Ff，它简单地使各方输入并计算f。 有关半诚实对手的更详细描述和证据，请参见[31]。 在我们的协议中，我们用（za,zb）←GarbledCircuit（x; y,f）表示这个乱码电路2PC

秘密共享和乘法三元组。 在我们的协议中，所有中间值都在两个服务器之间进行秘密共享。 我们采用了三种不同的共享方案：添加共享，布尔共享和Yao共享。 我们简要回顾一下这些方案，但请参阅[17]了解更多细节。

Boolean sharing and Yao sharing. We briefly review these schemes but refer the reader to [17] for more details.

To additively share ($\mathsf{Shr}^A(\cdot)$) an ℓ-bit value a, the first party P_0 generates $a_0 \in \mathbb{Z}_{2^\ell}$ uniformly at random and sends $a_1 = a - a_0 \bmod 2^\ell$ to the second party P_1. We denote the first party's share by $\langle a \rangle_0^A = a_0$ and the second party's by $\langle a \rangle_1^A = a_1$. For ease of composition we omit the modular operation in the protocol descriptions. In this paper, we mostly use the additive sharing, and denote it by $\langle \cdot \rangle$ for short. To reconstruct ($\boxed{\mathsf{Rec}^A(\cdot,\cdot)}$) an additively shared value $\langle a \rangle$, P_i sends $\langle a \rangle_i$ to P_{1-i} who computes $\langle a \rangle_0 + \langle a \rangle_1$.

计算两方和，重构原始值

Given two shared values $\langle a \rangle$ and $\langle b \rangle$, it is easy to non-interactively add the shares by having P_i compute $\langle c \rangle_i = \langle a \rangle_i + \langle b \rangle_i \bmod 2^\ell$. We overload the addition operation to denote the addition protocol by $\langle a \rangle + \langle b \rangle$.

To multiply ($\mathsf{Mul}^A(\cdot,\cdot)$) two shared values $\langle a \rangle$ and $\langle b \rangle$, we take advantage of Beaver's pre-computed multiplication triplet technique. Lets assume that the two parties already share $\langle u \rangle, \langle v \rangle, \langle z \rangle$ where u, v are uniformly random values in $\mathbb{Z}_{2^\ell}$ and $z = uv \bmod 2^\ell$. Then P_i locally computes $\langle e \rangle_i = \langle a \rangle_i - \langle u \rangle_i$ and $\langle f \rangle_i = \langle b \rangle_i - \langle v \rangle_i$. Both parties run $\mathsf{Rec}(\langle e \rangle_0, \langle e \rangle_1)$ and $\mathsf{Rec}(\langle f \rangle_0, \langle f \rangle_1)$, and P_i lets $\langle c \rangle_i = -i \cdot e \cdot f + f \cdot \langle a \rangle_i + e \cdot \langle b \rangle_i + \langle z \rangle_i$.

Boolean sharing can be seen as additive sharing in $\mathbb{Z}_2$ and hence all the protocols discussed above carry over. In particular, the addition operation is replaced by the XOR operation ($\oplus$) and multiplication is replaced by the AND operation ($\mathsf{AND}(\cdot,\cdot)$). We denote party P_i's share in a Boolean sharing by $\langle a \rangle_i^B$.

Finally, one can also think of a garbled circuit protocol as operating on Yao sharing of inputs to produce Yao sharing of outputs. In particular, in all garbling schemes, for each wire w the garbler (P_0) generates two random strings k_0^w, k_1^w. When using the point-and-permute technique [33] the garbler also generates a random permutation bit r_w and lets $K_0^w = k_0^w || r_w$ and $K_1^w = k_1^w || (1 - r_w)$. The concatenated bits are then used to permute the rows of each garbled truth table. A Yao sharing of a is $\langle a \rangle_0^Y = K_0^w, K_1^w$ and $\langle a \rangle_1^Y = K_a^w$. To reconstruct the shared value, parties exchange their shares. XOR and AND operations can be performed by garbling/evaluating the corresponding gates.

To switch from a Yao sharing $\langle a \rangle_0^Y = K_0^w, K_1^w$ and $\langle a \rangle_1^Y = K_a^w$ to a Boolean sharing, P_0 lets $\langle a \rangle_0^B = K_0^w[0]$ and P_1 lets $\langle a \rangle_1^B = \langle a \rangle_1^Y[0]$. In other words, the permutation bits used in the garbling scheme can be used to switch to boolean sharing for free. We denote this Yao to Boolean conversion by $\mathsf{Y2B}(\cdot,\cdot)$. We note that we do not explicitly use a Yao sharing in our protocol description as it will be hidden inside the garbling scheme, but explicitly use the $\mathsf{Y2B}$ conversion to convert the garbled output to a boolean sharing.

安全模型

3 Security Model

我们考虑一组客户C1，...，Cm，他们希望在他们的联合数据上训练各种模型。我们不会对如何在客户端之间分配数据做出任何假设。 特别地，数据可以是水平或垂直分区的，或者作为先前计算的一部分在它们之间秘密共享。

3.1 Architecture 架构

We consider a set of clients $\mathcal{C}_1, \ldots, \mathcal{C}_m$ who want to train various models on their joint data. We do not make any assumptions on how the data is distributed among the clients. In particular, the data can be horizontally or vertically partitioned, or be secret-shared among them as part of a previous computation.

A natural solution is to perform a secure multiparty computation where each client plays the role of one party. While this approach satisfies the privacy properties we are aiming for, it has

一种自然的解决方案是执行安全多方计算，其中每个客户端扮演一方的角色。 虽然这种方法满足了我们的目标隐私属性，但它有几个缺点。 首先，它要求客户参与整个协议。 其次，与两方案例不同，超过两方（以及不诚实的大多数）的技术显着地更加昂贵，并且不能扩展到较大输入规模或大量客户端。

因此，我们考虑服务器辅助设置，其中客户端将计算外包给两个不可信但非串通的服务器S0和S1。 服务器辅助的MPC已经形式化并用于以前的各种工作中（例如见[28]）。 它也被用于隐私保护机器学习的先前工作[36,35,20]。 这种设置的两个重要优点是（i）客户可以在设置阶段在两台服务器之间分配（秘密共享）它们的输入，但不参与任何未来的计算，以及（ii）我们可以从效率的组合中获益。 用于布尔运算的技术，例如乱码电路和OT扩展，以及算术计算，例如o ffl ine /在线乘法三元组共享。

several drawbacks. First, it requires the clients to be involved throughout the protocol. Second, unlike the two-party case, techniques for more than two parties (and a dishonest majority) are significantly more expensive and not scalable to large input sizes or a large number of clients.

Hence, we consider a server-aided setting where the clients outsource the computation to two untrusted but non-colluding servers $\mathcal{S}_0$ and $\mathcal{S}_1$. Server-aided MPC has been formalized and used in various previous work (e.g. see [28]). It has also been utilized in prior work on privacy-preserving machine learning [36, 35, 20]. Two important advantages of this setting are that (i) clients can distribute (secret-share) their inputs among the two servers in a setup phase but not be involved in any future computation, and (ii) we can benefit from a combination of efficient techniques for boolean computation such as garbled circuits and OT-extension, and arithmetic computation such as offline/online multiplication triplet shares.

根据应用场景，以前的工作是指两个服务器作为评估者和加密服务提供者(CSP)[36]，或评估者和维护数据的云服务提供者[23]。这两个服务器也可以是不同客户子集的代表，也可以是拥有数据的客户端。 无论分配给服务器的特定角色如何，信任模型都是相同的，并假设这两个服务器不受信任但不串通。我们接下来详细讨论安全定义

Depending on the application scenario, previous work refers to the two servers as the *evaluator* and the *cryptography service provider* (CSP) [36], or the evaluator and a *cloud service provider* who maintains the data [23]. The two servers can also be representatives of the different subsets of clients or themselves be among the clients who possess data. Regardless of the specific role assigned to the servers, the trust model is the same and assumes that the two servers are untrusted but do not collude. We discuss the security definition in detail next.

安全定义

3.2 Security Definition

回想一下，所涉及的各方是m个客户端C1, ..., Cm和两个服务器S0, S1. 我们假设一个半诚实的对手A可以破坏客户端的任何子集，并且最多可以破坏两个服务器中的一个。 这印证了两个服务器没有串通的属性，即如果一个是由对手控制的，则第二个服务器诚实地行事。 请注意，我们不对客户端之间以及客户端与服务器之间的串通进行任何限制。 我们称这样的对手为可接受的对手。 在一个特定场景中（参见第5节），我们通过要求服务器不与客户端串通来削弱安全模型。

Recall that the involved parties are m clients $\mathcal{C}_1, \ldots, \mathcal{C}_m$ and two servers $\mathcal{S}_0, \mathcal{S}_1$. We assume a *semi-honest* adversary $\mathcal{A}$ who can corrupt any subset of the clients and at most one of the two servers. This captures the property that the two servers are not colluding, i.e. if one is controlled by the adversary, the second one behaves honestly. Note that we do not put any restrictions on collusion among the clients and between the clients and the servers. We call such an adversary an *admissible adversary*. In one particular scenario (see Section 5), we weaken the security model by requiring that servers do not collude with the clients.

安全定义应该要求这样的对手只学习它已经损坏的客户端的数据和最终的输出，而不是剩下的其他诚实客户的数据。 例如，破坏C1, C2和S1的对手A不应该在训练模型之外学习有关C3数据的任何信息。 我们使用通用组成（UC）[14]的框架去定义安全性。 我们在这里简要介绍了定义，但请参阅[14]了解详细信息。 我们的协议的目标理想功能Fml如图3所示

The security definition should require that such an adversary only learns the data of the clients it has corrupted and the final output but nothing else about the remaining honest clients' data. For example, an adversary $\mathcal{A}$ who corrupts $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{S}_1$ should not learn any information about $\mathcal{C}_3$'s data beyond the trained model. We define security using the framework of Universal Composition (UC) [14]. We give a brief overview of the definition here, but refer the reader to [14] for the details. The target ideal functionality $\mathcal{F}_{ml}$ for our protocols is described in Figure 3.

UC框架中的执行涉及（非均匀）交互式图灵机的集合。 在这项工作中，我们考虑如上所述的可接受和半诚实的对手A. 双方根据协议交换消息。 未损坏方的协议输入由环境机器选择。 未受损害的各方也会向环境报告其协议输出。 在交互结束时，环境输出一个位。 攻击者还可以与环境任意交互 - 不失一般性，对手是虚拟对手，它只是将所有收到的协议消息转发到环境中，并按照环境的指示在协议中行动

An execution in the UC framework involves a collection of (non-uniform) interactive Turing machines. In this work we consider an admissible and semi-honest adversary $\mathcal{A}$ as discussed above. The *parties* exchange messages according to a protocol. Protocol inputs of uncorrupted parties are chosen by an *environment* machine. Uncorrupted parties also report their protocol outputs to the environment. At the end of the interaction, the environment outputs a single bit. The adversary can also interact arbitrarily with the environment — without loss of generality the adversary is a *dummy* adversary which simply forwards all received protocol messages to the environment and acts in the protocol as instructed by the environment.

通过比较真实和理想的交互来定义安全性。 设real [Z, A, π, λ]表示与对手A和在安全参数λ上执行协议π的诚实方交互时环境Z的最终(单比特)输出。 这种相互作用被称为涉及协议π的真实交互。

Security is defined by comparing a real and ideal interaction. Let $\text{REAL}[\mathcal{Z}, \mathcal{A}, \pi, \lambda]$ denote the final (single-bit) output of the environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and honest parties who execute protocol π on security parameter λ. This interaction is referred to as the **real** interaction involving protocol π.

In the **ideal** interaction, parties simply forward the inputs they receive to an uncorruptable

在理想的交互中，各方简单地将他们收到的输入转发给不可破坏的功能机器，并转发功能对环境的响应。 因此，可信任功能代表各方执行整个计算。 协议的目标理想功能Fml在图3中描述。令 ideal[Z, S, Fml, λ]表示与对手S和在存在功能F时运行伪协议的诚实方交互时环境Z的输出 安全参数λ。

Parameters: Clients $\mathcal{C}_1, \ldots, \mathcal{C}_m$ and servers $\mathcal{S}_0, \mathcal{S}_1$.
Uploading Data: On input x_i from $\mathcal{C}_i$, store x_i internally.
Computation: On input f from $\mathcal{S}_0$ or $\mathcal{S}_1$, compute $(y_1, \ldots, y_m) = f(x_1, \ldots, x_m)$ and send y_i to $\mathcal{C}_i$. This step can be repeated multiple times with different functions.

Figure 3: Ideal Functionality $\mathcal{F}_{ml}$

functionality machine and forward the functionality's response to the environment. Hence, the trusted functionality performs the entire computation on behalf of the parties. The target ideal functionality $\mathcal{F}_{ml}$ for protocols is described in Figure 3. Let $\text{IDEAL}[\mathcal{Z}, \mathcal{S}, \mathcal{F}_{ml}, \lambda]$ denote the output of the environment $\mathcal{Z}$ when interacting with adversary $\mathcal{S}$ and honest parties who run the dummy protocol in presence of functionality $\mathcal{F}$ on security parameter λ.

我们说协议π安全地实现了功能Fml，如果对于每个可接受的对手A攻击真实的交互(不失一般性，我们可以将A作为虚拟对手)，存在攻击者S(称为模拟器)攻击理想的相互作用，使得对于所有环境Z，以下数量可忽略不计(参数为λ)：

We say that a protocol π **securely realizes** a functionality $\mathcal{F}_{ml}$ if for every *admissible* adversary $\mathcal{A}$ attacking the real interaction (without loss of generality, we can take $\mathcal{A}$ to be the dummy adversary), there exists an adversary $\mathcal{S}$ (called a simulator) attacking the ideal interaction, such that for all environments $\mathcal{Z}$, the following quantity is negligible (in λ):

$$\left| \Pr\left[\text{REAL}[\mathcal{Z}, \mathcal{A}, \pi, \lambda] = 1\right] - \Pr\left[\text{IDEAL}[\mathcal{Z}, \mathcal{S}, \mathcal{F}_{ml}, \lambda] = 1\right] \right|.$$

Intuitively, the simulator must achieve the same effect (on the environment) in the ideal interaction that the adversary achieves in the real interaction. Note that the environment's view includes (without loss of generality) all of the messages that honest parties sent to the adversary as well as the outputs of the honest parties.

直观地，模拟器必须在对手在真实交互中实现的理想交互中实现相同的效果（在环境上）。请注意，环境视图包括（不失一般性）诚实方发送给对手的所有消息以及诚实方的输出。

隐私保护机器学习

4 Privacy Preserving Machine Learning

在本节中，我们将介绍使用SGD进行隐私保护机器学习的协议。我们首先在4.1节中描述了一个线性回归协议，完全基于算术秘密共享和乘法三元组。接下来，我们将讨论如何在4.2节的o ffl ine阶段中有效地生成这些乘法三元组。然后，我们概括了我们的技术，以支持4.3和4.4节中的逻辑回归和神经网络训练。最后，第4.5节介绍了支持预测，学习率调整和终止确定的技术。

In this section, we present our protocols for privacy preserving machine learning using SGD. We first describe a protocol for linear regression in Section 4.1, based solely on arithmetic secret sharing and multiplication triplets. Next, we discuss how to efficiently generate these multiplication triplets in the offline phase in Section 4.2. We then generalize our techniques to support logistic regression and neural networks training in Sections 4.3 and 4.4. Finally, techniques to support predication, learning rate adjustment and termination determination are presented in Section 4.5.

隐私保护线性回归

4.1 Privacy Preserving Linear Regression

Recall that we assume the training data is secret shared between two servers $\mathcal{S}_0$ and $\mathcal{S}_1$. We denote the shares by $\langle \mathbf{X} \rangle_0, \langle \mathbf{Y} \rangle_0$ and $\langle \mathbf{X} \rangle_1, \langle \mathbf{Y} \rangle_1$. In practice, the clients can distribute the shares between the two servers, or encrypt the first share using the public key of $\mathcal{S}_0$, upload both the first encrypted share and the second plaintext share to $\mathcal{S}_1$. $\mathcal{S}_1$ then passes the encrypted shares to $\mathcal{S}_0$ to decrypt. In our protocol, we also let the coefficients $\mathbf{w}$ be secret shared between the two servers. It is initialized to random values simply by setting $\langle \mathbf{w} \rangle_0, \langle \mathbf{w} \rangle_1$ to be random, without any communication between the two servers. It is updated and remains secret shared after each iteration of SGD, until the end when it is reconstructed.

As described in Section 2.1, the update function for linear regression is $w_j := w_j - \alpha(\sum_{k=1}^{d} x_{ik} w_k - y_i) x_{ij}$, only consisting of additions and multiplications. Therefore, we apply the corresponding

回想一下，我们假设训练数据是在两个服务器S0和S1之间秘密共享的。我们用X0，Y0和X1，Y1表示这的共享。实际上，客户端可以在两个服务器之间分配共享，或者使用S0的公钥加密第一个共享，将第一个加密共享和第二个明文共享上传到S1。S1然后将加密的共享传递给S0以进行解密。在我们的协议中，我们还令两个服务器之间进行秘密共享的系数为w。只需将w0，w1设置为随机，就可以将其初始化为随机值，而无需在两个服务器之间进行任何通信。它在每次SGD迭代后更新并保持秘密共享，直到重建结束。

addition and multiplication algorithms for secret shared values to update the coefficients, which is $\langle w_j \rangle := \langle w_j \rangle - \alpha \mathsf{Mul}^A\big(\sum_{k=1}^{d} \mathsf{Mul}^A(\langle x_{ik}\rangle, \langle w_k\rangle) - \langle y_i\rangle, \langle x_{ij}\rangle\big)$. We separate our protocol into two phases: online and offline. The online phase trains the model given the data, while the offline phase consists mainly of multiplication triplet generation. We focus on the online phase in this section, and discuss the offline phase in Section 4.2.

我们还希望从2.1节中讨论的小批量和矢量化技术中获益(见公式2)

共享设置中的矢量化

Vectorization in the Shared Setting. We also want to benefit from the mini-batch and vectorization techniques discussed in Section 2.1 (see Equation 2). To achieve this, we generalize the addition and multiplication operations on share values to shared matrices. Matrices are shared by applying Shr^A to every element. Given two shared matrices $\langle \mathbf{A} \rangle$ and $\langle \mathbf{B} \rangle$, matrix addition can be computed non-interactively by letting $\langle \mathbf{C} \rangle_i = \langle \mathbf{A} \rangle_i + \langle \mathbf{B} \rangle_i$ for $i \in \{0, 1\}$. To multiply two shared matrices, instead of using independent multiplication triplets, we take shared matrices $\langle \mathbf{U} \rangle, \langle \mathbf{V} \rangle, \langle \mathbf{Z} \rangle$, where each element in $\mathbf{U}$ and $\mathbf{V}$ is uniformly random in $\mathbb{Z}_{2^l}$, $\mathbf{U}$ has the same dimension as $\mathbf{A}$, $\mathbf{V}$ has the same dimension as $\mathbf{B}$ and $\mathbf{Z} = \mathbf{U} \times \mathbf{V} \mod 2^l$. $\mathcal{S}_i$ computes $\langle \mathbf{E} \rangle_i = \langle \mathbf{A} \rangle_i - \langle \mathbf{U} \rangle_i$, $\langle \mathbf{F} \rangle_i = \langle \mathbf{B} \rangle_i - \langle \mathbf{V} \rangle_i$ and sends it to the other server. Both servers reconstruct $\mathbf{E}$ and $\mathbf{F}$ and set $\langle \mathbf{C} \rangle_i = -i \cdot \mathbf{E} \times \mathbf{F} + \langle \mathbf{A} \rangle_i \times \mathbf{F} + \mathbf{E} \times \langle \mathbf{B} \rangle_i + \langle \mathbf{Z} \rangle_i$. The idea of this generalization is that each element in matrix $\mathbf{A}$ is always masked by the same random element in $\mathbf{U}$, while it is multiplied by different elements in $\mathbf{B}$ in the matrix multiplication. Our security proof confirms that this does not affect security of the protocol, but makes the protocol significantly more efficient due to vectorization.

为实现这一目标，我们将共享值的加法和乘法运算推广到共享矩阵. 通过将ShrA应用于每个元素来共享矩阵. 给定两个共享矩阵hAi和hBi，可以通过使Ci = Ai + Bi for i∈{0,1}来非交互地计算矩阵加法

这种推广的想法是矩阵A中的每个元素总是被U中的相同随机元素掩盖，而它在矩阵乘法中乘以B中的不同元素。 我们的安全证明确认这不会影响协议的安全性，但会使协议因矢量化而显着提高效率。

Applying the technique to linear regression, in each iteration, we assume the set of mini-batch indices B is public, and perform the update $\langle \mathbf{w} \rangle := \langle \mathbf{w} \rangle - \frac{1}{|B|}\alpha \mathsf{Mul}^A(\langle \mathbf{X}_B^T \rangle, \mathsf{Mul}^A\big(\langle \mathbf{X}_B \rangle, \langle \mathbf{w} \rangle) - \langle \mathbf{Y}_B \rangle\big)$. We further observe that one data sample will be used several times in different epochs, yet it suffices to mask it by the same random multiplication triplet. Therefore, in the offline phase, one shared $n \times d$ random matrix $\langle \mathbf{U} \rangle$ is generated to mask the data samples $\langle \mathbf{X} \rangle$. At the beginning of the online phase, $\langle \mathbf{E} \rangle_i = \langle \mathbf{X} \rangle_i - \langle \mathbf{U} \rangle_i$ is computed and exchanged to reconstruct $\mathbf{E}$ through one interaction. After that, in each iteration, $\mathbf{E}_B$ is selected and used in the multiplication protocol, without any further computation and communication. In particular, in the offline phase, a series of min-batch indices $B_1, \ldots, B_t$ are agreed upon by the two servers. This only requires the knowledge of n, d, t or an upperbound, but not any real data. Then the multiplication triplets $\langle \mathbf{U} \rangle, \langle \mathbf{V} \rangle, \langle \mathbf{Z} \rangle, \langle \mathbf{V}' \rangle, \langle \mathbf{Z}' \rangle$ are precomputed with the following property: $\mathbf{U}$ is an $n \times d$ matrix to mask the data $\mathbf{X}$, $\mathbf{V}$ is a $d \times t$ matrix, each column of which is used to mask $\mathbf{w}$ in one iteration (forward propagation), and $\mathbf{V}'$ is a $|B| \times t$ matrix wherein each column is used to mask the difference vector $\mathbf{Y}^* - \mathbf{Y}$ in one iteration (backward propagation). We then let $\mathbf{Z}[i] = \mathbf{U}_{B_i} \times \mathbf{V}[i]$ and $\mathbf{Z}'[i] = \mathbf{U}_{B_i}^T \times \mathbf{V}'[i]$ for $i = 1, \ldots, t$, where $\mathbf{M}[i]$ denotes the ith column of the matrix $\mathbf{M}$. Using the multiplication triplets in matrix form, the computation and communication in both the online and the offline phase are reduced dramatically. We will analyze the cost later.

We denote the ideal functionality realizing the generation of these matrices in the offline phase by $\mathcal{F}_{offline}$.

我们表示理想的功能是通过Foffline实现这些矩阵在离线阶段的生成。

Arithmetic Operations on Shared Decimal Numbers. As discussed earlier, a major source of inefficiency in prior work on privacy preserving linear regression stems from computing on shared/encrypted decimal numbers. Prior solutions either treat decimal numbers as integers and preserve full accuracy after multiplication by using a very large finite field [21], or utilize 2PC for boolean circuits to perform fixed-point [20] or floating-point [34] multiplication on decimal numbers. The former can only support a limited number of multiplications, as the range of the result grows exponentially with the number of multiplications. This is prohibitive for training where the number of multiplications is large. The latter introduces high overhead, as the boolean circuit for multiplying

共享小数的算术运算。 如前所述，隐私保护线性回归的先前工作中的效率的主要来源源于对共享/加密十进制数的计算。 先前的解决方案要么将十进制数视为整数，而且在乘法后使用非常大的有限字段保持完全准确[21]，要么利用2PC进行布尔电路以对十进制数执行定点[20]或浮点数[34]乘法。 前者只能支持有限数量的乘法，因为结果的范围随着乘法的数量呈指数增长。 这对于乘法次数较多的训练来说是不可行的。 后者引入了高开销，因为用于乘以两个l位的布尔电路具有O（l^2）门，并且对于每次执行的乘法都需要在2PC（例如，Yao的乱码电路）中计算这样的电路。

我们提出了一个简单但有效的解决方案,以支持整数字段中的十进制算术。考虑小数部分中最多为l_D位两个十进制数x和y的定点乘法，我们首先通过让x'= 2^l_D*x和y'= 2^l_D*y将数字转换为整数，然后将它们相乘以获得乘积z = x'y'。注意，z最多有2*l_D位表示乘积的小数部分，因此我们简单地截断z的最后l_D位，使得它最多具有代表小数部分的l_D位。从数学上讲，如果z被分解为两个部分z = z1·2^l_D + z2, 其中0≤z2<2^l_D, 则截断结果为z1. 我们用z表示这种截断操作

two l-bit numbers has $O(l^2)$ gates, and such a circuit needs to be computed in a 2PC (e.g. Yao's garbled circuits) for each multiplication performed.

我们表明，当z是秘密共享时，这种截断技术也有效。特别是，这两个服务器可以独立地截断它们各自的z共享。在下面的定理中，我们证明了对于足够大的场，这些截断的共享在重建时具有高概率，与期望的z最多相差1。换句话说，与标准固定点算法相比，我们在小数部分的最小重要位中产生一个小误差。

We propose a simple but effective solution to support decimal arithmetics in an integer field. Consider the fixed-point multiplication of two decimal numbers x and y with at most l_D bits in the fractional part. We first transform the numbers to integers by letting $x' = 2^{l_D}x$ and $y' = 2^{l_D}y$ and then multiply them to obtain the product $z = x'y'$. Note that z has at most $2l_D$ bits representing the fractional part of the product, so we simply *truncate* the last l_D bits of z such that it has at most l_D bits representing the fractional part. Mathematically speaking, if z is decomposed into two parts $z = z_1 \cdot 2^{l_D} + z_2$, where $0 \le z_2 < 2^{l_D}$, then the truncation results is z_1. We denote this truncation operations by $\lfloor z \rfloor$.

We show that this truncation technique also works when z is secret shared. In particular, the two servers can truncate their individual shares of z independently. In the following theorem we prove that for a large enough field, these truncated shares when reconstructed, with high probability, are at most 1 off from the desired $\lfloor z \rfloor$. In other words, we incur a small error in the least significant bit of the fractional part compared to standard fixed-point arithmetic.

我们还注意到，如果十进制数z为负,它将在字段中表示为2^l-|z|,其中|z|是它的绝对值，截断操作变为z = 2^l-|z|. 我们证明了正数和负数的以下定理。

We also note that if a decimal number z is negative, it will be represented in the field as $2^l - |z|$, where $|z|$ is its absolute value and the truncation operation changes to $\lfloor z \rfloor = 2^l - \lfloor |z| \rfloor$. We prove the following theorem for both positive and negative numbers.

Theorem 1. *In field $\mathbb{Z}_{2^l}$, let $x \in [0, 2^{l_x}] \cup [2^l - 2^{l_x}, 2^l)$, where $l > l_x + 1$ and given shares $\langle x \rangle_0, \langle x \rangle_1$ of x, let $\langle \lfloor x \rfloor \rangle_0 = \lfloor \langle x \rangle_0 \rfloor$ and $\langle \lfloor x \rfloor \rangle_1 = 2^l - \lfloor 2^l - \langle x \rangle_1 \rfloor$. Then with probability $1 - 2^{l_x+1-l}$, $\mathsf{Rec}^A(\langle \lfloor x \rfloor \rangle_0, \langle \lfloor x \rfloor \rangle_1) \in \{\lfloor x \rfloor - 1, \lfloor x \rfloor, \lfloor x \rfloor + 1\}$, where $\lfloor \cdot \rfloor$ denotes truncation by $l_D \le l_x$ bits.*

Proof. Let $\langle x \rangle_0 = x + r \mod 2^l$, where r is uniformly random in $\mathbb{Z}_{2^l}$, then $\langle x \rangle_1 = 2^l - r$. We decompose r as $r_1 \cdot 2^{l_D} + r_2$, where $0 \le r_2 < 2^{l_D}$ and $0 \le r_1 < 2^{l - l_D}$. We prove that if $2^{l_x} \le r < 2^l - 2^{l_x}$, $\mathsf{Rec}^A(\langle \lfloor x \rfloor \rangle_0, \langle \lfloor x \rfloor \rangle_1) \in \{\lfloor x \rfloor - 1, \lfloor x \rfloor, \lfloor x \rfloor + 1\}$. Consider the following two cases.

Case 1: If $0 \le x \le 2^{l_x}$, then $0 < x + r < 2^l$ and $\langle x \rangle_0 = x + r$, without modulo. Let $x = x_1 \cdot 2^{l_D} + x_2$, where $0 \le x_2 < 2^{l_D}$ and $0 \le x_1 < 2^{l_x - l_D}$. Then we have $x + r = (x_1 + r_1) \cdot 2^{l_D} + (x_2 + r_2) = (x_1 + r_1 + c) \cdot 2^{l_D} + (x_2 + r_2 - c \cdot 2^{l_D})$, where the carry bit $c = 0$ if $x_2 + r_2 < 2^{l_D}$ and $c = 1$ otherwise. After the truncation, $\langle \lfloor x \rfloor \rangle_0 = \lfloor x + r \rfloor = x_1 + r_1 + c$ and $\langle \lfloor x \rfloor \rangle_1 = 2^l - r_1$. Therefore, $\mathsf{Rec}^A(\langle \lfloor x \rfloor \rangle_0, \langle \lfloor x \rfloor \rangle_1) = x_1 + c = \lfloor x \rfloor + c$.

Case 2: If $2^l - 2^{l_x} \le x < 2^l$, then $x + r \ge 2^l$ and $\langle x \rangle_0 = x + r - 2^l$. Let $x = 2^l - x_1 \cdot 2^{l_D} - x_2$, where $0 \le x_2 < 2^{l_D}$ and $0 \le x_1 < 2^{l_x - l_D}$. We have $x + r - 2^l = (r_1 - x_1) \cdot 2^{l_D} + (r_2 - x_2) = (r_1 - x_1 - c) \cdot 2^{l_D} + (r_2 - x_2 + c \cdot 2^{l_D})$, where the carry bit $c = 0$ if $r_2 > x_2$ and $c = 1$ otherwise. After the truncation, $\langle \lfloor x \rfloor \rangle_0 = \lfloor x + r - 2^l \rfloor = r_1 - x_1 - c$ and $\langle \lfloor x \rfloor \rangle_1 = 2^l - r_1$. Therefore, $\mathsf{Rec}^A(\langle \lfloor x \rfloor \rangle_0, \langle \lfloor x \rfloor \rangle_1) = 2^l - x_1 - c = \lfloor x \rfloor - c$.

Finally, the probability that our assumption holds, i.e. the probability of a random r being in the range $[2^{l_x}, 2^l - 2^{l_x})$ is $1 - 2^{l_x+1-l}$. □

通过在证明中用p替换2^l，定理1可以以自然方式扩展到素数字段Zp。我们还注意到截断不会影响秘密共享的安全性，因为每个参与者在没有任何交互的情况下独立地截断共享。

Theorem 1 can be extended to a prime field $\mathbb{Z}_p$ in a natural way by replacing 2^l with p in the proof. We also note that the truncation does not affect security of the secret sharing as the shares are truncated independently by each party without any interaction.

The complete protocol between the two servers for the online phase of privacy preserving linear regression is shown in Figure 4. It assumes that the data-independent shared matrices $\langle \mathbf{U} \rangle, \langle \mathbf{V} \rangle, \langle \mathbf{Z} \rangle, \langle \mathbf{V}' \rangle, \langle \mathbf{Z}' \rangle$ were already generated in the offline phase. Besides multiplication and addition of shared decimal numbers, the protocol also requires multiplying the coefficient vector by

两个服务器之间用于隐私保护线性回归的在线阶段的完整协议如图4所示。它假设数据无关的共享矩阵U, V, Z, V', Z'已经在离线阶段生成。除了乘法和加法共享十进制数之外，该协议还要求在每次迭代中将系数向量乘以A. 为了使这个操作有效，我们将A设置为2的幂，即A = 2^-k。然后，可以通过让各方从其系数的共享中截断k个附加位来代替与A的乘法。

Protocol $\mathsf{SGD\_Linear}(\langle \mathbf{X}\rangle, \langle \mathbf{Y}\rangle, \langle \mathbf{U}\rangle, \langle \mathbf{V}\rangle, \langle \mathbf{Z}\rangle, \langle \mathbf{V}'\rangle, \langle \mathbf{Z}'\rangle)$:

$\mathcal{S}_i$ computes $\langle \mathbf{E}\rangle_i = \langle \mathbf{X}\rangle_i - \langle \mathbf{U}\rangle_i$ for $i \in \{0,1\}$. Then parties run $\mathsf{Rec}(\langle \mathbf{E}\rangle_0, \langle \mathbf{E}\rangle_1)$ to obtain $\mathbf{E}$.
for $j = 1, \dots, t$ **do**
    Parties select the mini-batch $\langle \mathbf{X}_{B_j}\rangle, \langle \mathbf{Y}_{B_j}\rangle$.
    $\mathcal{S}_i$ computes $\langle \mathbf{F}_j\rangle_i = \langle \mathbf{w}\rangle_i - \langle \mathbf{V}[j]\rangle$ for $i \in \{0,1\}$. Then parties run $\mathsf{Rec}(\langle \mathbf{F}_j\rangle_0, \langle \mathbf{F}_j\rangle_1)$ to recover $\mathbf{F}_j$.
    $\mathcal{S}_i$ computes $\langle \mathbf{Y}^*_{B_j}\rangle_i = -i \cdot \mathbf{E}_{B_j} \times \mathbf{F}_i + \langle \mathbf{X}_{B_j}\rangle_i \times \mathbf{F}_i + \mathbf{E}_{B_j} \times \langle \mathbf{w}\rangle_i + \langle \mathbf{Z}_j\rangle_i$ for $i \in \{0,1\}$.
    $\mathcal{S}_i$ compute the difference $\langle \mathbf{D}_{B_j}\rangle_i = \langle \mathbf{Y}^*_{B_j}\rangle_i - \langle \mathbf{Y}_{B_j}\rangle_i$ for $i \in \{0,1\}$.
    $\mathcal{S}_i$ computes $\langle \mathbf{F}'_j\rangle_i = \langle \mathbf{D}_{B_j}\rangle_i - \langle \mathbf{V}'_j\rangle_i$ for $i \in \{0,1\}$. Parties then run $\mathsf{Rec}(\langle \mathbf{F}'_j\rangle_0, \langle \mathbf{F}'_j\rangle_1)$ to obtain $\mathbf{F}'_j$.
    $\mathcal{S}_i$ computes $\langle \mathbf{\Delta}\rangle_i = -i \cdot \mathbf{E}^T_{B_j} \times \mathbf{F}'_j + \langle \mathbf{X}^T_{B_j}\rangle_i \times \mathbf{F}'_j + \mathbf{E}^T_{B_j} \times \langle \mathbf{D}_{B_j}\rangle_i + \langle \mathbf{Z}'_j\rangle_i$ for $i \in \{0,1\}$.
    $\mathcal{S}_i$ truncates its shares of Δ element-wise to get $\lfloor \langle \mathbf{\Delta}\rangle_i \rfloor$.
    $\mathcal{S}_i$ computes $\langle \mathbf{w}\rangle_i := \langle \mathbf{w}\rangle_i - \frac{\alpha}{|B|}\lfloor \langle \mathbf{\Delta}\rangle_i \rfloor$ for $i \in \{0,1\}$.
Parties run $\mathsf{Rec}^A(\langle \mathbf{w}\rangle_0, \langle \mathbf{w}\rangle_1)$ and output $\mathbf{w}$.

Figure 4: The online phase of privacy preserving linear regression.

图4：隐私保护线性回归的在线阶段。

$\frac{\alpha}{|B|}$ in each iteration. To make this operation efficient, we set $\frac{\alpha}{|B|}$ to be a power of 2, i.e. $\frac{\alpha}{|B|} = 2^{-k}$. Then the multiplication with $\frac{\alpha}{|B|}$ can be replaced by having the parties truncate k additional bits from their shares of the coefficients.

定理2.考虑一种协议，其中客户端在运行图4协议的两个服务器之间分配其数据的算术份额，并将输出发送给客户端。在Offline混合模型中，该协议实现了图3中线性回归函数的理想函数Fml，存在半诚实的可接受对手（参见第3节）

We sketch a proof for the following Theorem on security of the online protocol.

我们概述了以下关于在线协议安全性的定理的证明

Theorem 2. *Consider a protocol where clients distribute arithmetic shares of their data among two servers who run the protocol of Figure 4 and send the output to clients. In the $\mathcal{F}_{offline}$ hybrid model, this protocol realizes the ideal functionality $\mathcal{F}_{ml}$ of Figure 3 for the linear regression function, in presence of a semi-honest* admissible *adversary (see section 3).*

草图。我们模型中的可接受对手可能会破坏一个服务器和客户端的任何子集。鉴于协议相对于两个服务器是对称的，我们只需要考虑对手破坏S0和除了一个客户端之外的所有客户端的场景，即C1, ..., Cm-1。

sketch. An admissible adversary in our model can corrupt one server and any subset of the clients. Given that the protocol is symmetric with respect to the two servers, we simply need to consider the scenario where the adversary corrupts $\mathcal{S}_0$ and all but one of the clients, i.e. $\mathcal{C}_1, \dots, \mathcal{C}_{m-1}$.

我们描述了一个模拟器S，模拟了理想世界中的上述对手。S将损坏的客户端输入数据提交给功能，并接收线性回归的最终输出，即系数的最终值。

We describe a simulator $\mathcal{S}$ that simulates the above adversary in the ideal world. $\mathcal{S}$ submits the corrupted clients' inputs data to the functionality and receives the final output of the linear regression i.e. the final value of the coefficients $\mathbf{w}$ back.

S然后运行A.代表诚实的客户端S将Z2l中的随机共享发送给A，用于该客户端持有的每个值。这是客户参与的唯一消息。在协议的其余部分中，生成与诚实服务器的X，Y，U，V，Z，V'，Z'的份额相对应的随机矩阵和向量，并使用随机的那些在与A的交互中扮演诚实服务器的角色 生成的值

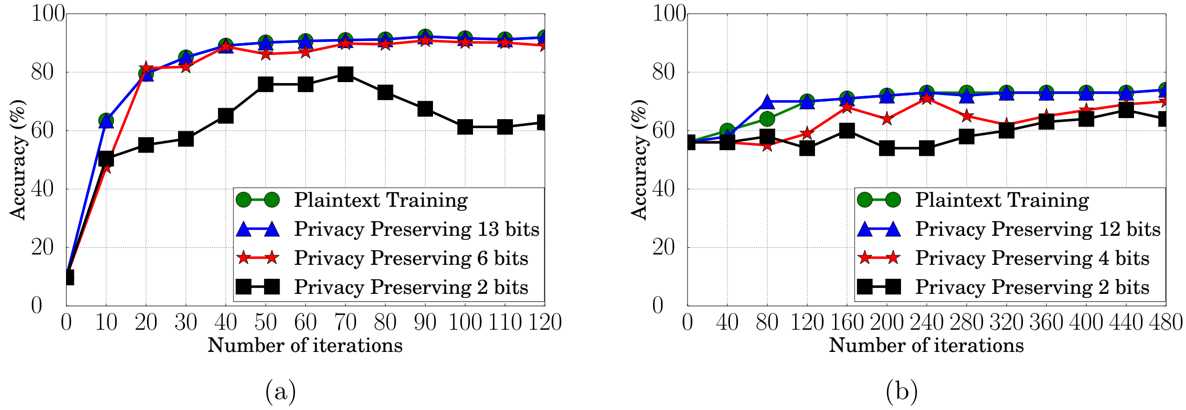
$\mathcal{S}$ then runs $\mathcal{A}$. On behalf of the honest client(s) $\mathcal{S}$ sends a random share in $\mathbb{Z}_{2^l}$ to $\mathcal{A}$ for each value in the held by that client. This is the only message where clients are involved. In the remainder of the protocol, generate random matrices and vectors corresponding to the honest server's shares of $\langle \mathbf{X}\rangle, \langle \mathbf{Y}\rangle, \langle \mathbf{U}\rangle, \langle \mathbf{V}\rangle, \langle \mathbf{Z}\rangle, \langle \mathbf{V}'\rangle, \langle \mathbf{Z}'\rangle$, and play the role of the honest server in interactions with $\mathcal{A}$ using those randomly generated values.

最后，在要恢复w的最后一步中，S调整诚实服务器的w份额，使得恢复的值确实是它从功能中接收的系数向量。这结束了模拟

Finally, in the very last step where $\mathbf{w}$ is to be recovered, $\mathcal{S}$ adjusts the honest servers' share of $\mathbf{w}$ such that the recovered value is indeed the coefficient vector it received from the functionality. This concludes the simulation.

We briefly argue that the $\mathcal{A}$'s view in the real and ideal worlds and as a result, the environment's view in the two worlds is indistinguishable. This immediately follows from the security of the arithmetic secret sharing and the fact that the matrices/vectors generated in the offline phase are indeed random. In particular, all messages sent and received and reconstructed in the protocol (with the exception of $\mathbf{w}$ are generated using uniformly random shares in both the real protocol and

我们简单地认为A在真实世界和理想世界中的观点，因此，环境在两个世界中的观点是难以区分的。这直接来自算术秘密共享的安全性以及在offline阶段中生成的矩阵/向量确实是随机的这一事实。特别是，在协议中发送和接收并重建的所有消息（除了w之外，都是使用上述真实协议和模拟中的均匀随机共享生成的，因此实际上视图都是相同分布的。这就是我们的论点。



我们注意到这个论点隐含地解释了为什么使用一个掩码矩阵U足以隐藏数据矩阵X.原因是攻击者只能在第一次交互中看到掩码值一次，而在X上的其余计算发生 没有诚实和损坏的服务器之间的交互

Figure 5: Comparison of accuracy of privacy preserving linear regression with truncation and plaintext training on decimal numbers. (a) MNIST dataset, $|B| = 128$, (b) Arcene dataset, $|B| = 32$.

图5：隐私保护线性回归与十进制数截断和明文训练的准确性比较。（a）MNIST数据集,|B| = 128,（b）Arcene数据集,|B| = 32。

the simulation described above, so indeed the view are both identically distributed. this concludes our argument.

We note that this argument implicitly explains why using one mask matrix $\mathbf{U}$ is sufficient to hide the data matrix $\mathbf{X}$. The reason is that the adversary only gets to see the masked value once in the first interaction and the rest of the computation on $\mathbf{X}$ takes place without interactions between the honest and the corrupted server. □

截断误差的影响。请注意，当字段的大小足够大时，可以每次迭代执行一次截断，而不是每次乘法执行一次。因此，在我们的实现中，执行截断(|B| + d)·t次，并且通过联合边界，训练中失败的概率是

Effect of Truncation Error. Note that when the size of the field is large enough, truncation can be performed once per iteration instead of once per multiplication. Hence in our implementations, the truncation is performed $(|B|+d)\cdot t$ times and by the union bound, the probability of failure in the training is $\boxed{(|B|+d)\cdot t\cdot 2^{l_x+1-l}}$. For typical parameters $|B| = 128, d = 784, t = 1000, l_x = 32, l = 64$, the probability of a single failure happening during the whole training is around 2^{-12}. Moreover, even if a failure in the truncation occurs, it is unlikely to translate to a failure in training. Such a failure makes one feature in one sample invalid, yet the final model should not be affected by small changes in data, or else the trainig strategy suffers from overfitting. We confirm these observations by running experiments on two different datasets (MNIST [6] and Arcene [1]). In particular, we show that accuracy of the models trained using privacy preserving linear regression with truncation matches those of plaintext training using standard arithmetic.

对于典型参数|B| = 128， d = 784， t = 1000， lx = 32， l = 64，整个训练期间发生单次故障的概率大约为2-12。 此外，即使发生截断失败，也不太可能转化为训练失败。这样的失败使得一个样本中的一个特征无效，但最终模型不应该受到数据的微小变化的影响，否则训练策略会受到过度配置的影响。 我们通过在两个不同的数据集（MNIST [6]和Arcene [1]）上进行实验来证实这些观察结果。 特别地，我们表明使用具有截断的隐私保护线性回归训练的模型的准确性与使用标准算法的明文训练的准确性相匹配。

我们使用截断技术在MNIST数据集[6]上运行我们的隐私保护线性回归协议，该技术由手写数字图像组成，并将训练模型的准确性与标准十进制数操作的明文训练进行比较。 小批量大小设置为|B|=128，学习率为 α = 2^-7. 输入数据具有784个特征，每个特征是在0和1之间缩放的像素的灰度级，使用8个小数位表示。 我们将字段设置为Z_2^64。 为了公平比较，系数都被初始化为0，并且小批量索引的相同序列用于所有训练。 为简化任务，我们将数字“0”的标签更改为0，非零数字的标签更改为1。

We run our privacy preserving linear regression protocol with the truncation technique on the MNIST dataset [6] consisting of images of handwriting digits and compare accuracy of the trained model to plaintext training with standard decimal numbers operations. The mini-batch size is set to $|B| = 128$ and the learning rate is $\alpha = 2^{-7}$. The input data has 784 features, each a gray scale of a pixel scaled between 0 and 1, represented using 8 decimal bits. We set the field to $\mathbb{Z}_{2^{64}}$. For a fair comparison, coefficients are all initialized to 0s and the same sequence of the mini-batch indices are used for all trainings. To simplify the task, we change the labels to be 0 for digit “0” and 1 for non-zero digits. In Figure 5a, the x-axis is the number of iterations of the SGD algorithm and the y-axis is the accuracy of the trained model on the testing dataset. Here we reconstruct the coefficient vector after every iteration in our protocol to test the accuracy. As shown in Figure 5a, when we use 13 bits for the fractional part of $\mathbf{w}$, the privacy preserving training behaves almost exactly the same as the plaintext training. This is because we only introduce a small error on the

在图5a中，x轴是SGD算法的迭代次数，y轴是训练模型在测试数据集上的精度。在这里，我们在我们的协议中的每次迭代之后重建系数向量以测试准确性。如图5a所示，当我们对w的小数部分使用13位时，隐私保护训练的行为几乎与明文训练完全相同。这是因为我们只在w的小数部分的第13位引入了一个小错误。我们的实验从未触发定理1中的失败条件。但是，当我们使用6位作为w的小数部分时，我们的协议的准确性在训练期间振荡。这是因为现在误差在第6位，其具有更大的效果并且可能使模型远离最佳值。当距离最佳值的距离足够大时，SGD将再次向最佳状态移动。最后，当我们对小数部分使用2位时，振荡行为更加极端。当我们在另一个名为Arcene [1]的数据集上进行训练时，我们观察到类似的效果，如图5b所示。换句话说，当使用足够的比特来表示系数的小数部分时，我们用于共享十进制数的固定点乘法的新方法对训练模型的准确性几乎没有影响。

13th bit of the decimal part of $\mathbf{w}$. Our experiments never triggered the failure condition in theorem 1. However, when we use 6 bits for the decimal part of $\mathbf{w}$, the accuracy of our protocol oscillates during the training. This is because now the error is on the 6th bit which has a larger effect and may push the model away from the optimum. When the distance to the optimum is large enough, the SGD will move back towards the optimum again. Finally, when we use 2 bits for the fractional part, the oscillating behavior is more extreme. We observe a similar effect when training on another dataset called Arcene [1] as shown in Figure 5b. In other words, when sufficient bits are used to represent the fractional part of the coefficients, our new approach for fixed-point multiplication of shared decimal numbers has little impact on accuracy of the trained model.

效率讨论。图4的计算成本中的主导项是步骤5和8中的矩阵乘法。在每次迭代中，每一方执行4次这样的矩阵乘法，而在明文SGD训练中，根据等式2，执行2次矩阵乘法。因此，每一方的计算时间仅是对明文数据进行训练的时间的两倍

Efficiency Discussion. The dominating term in the computation cost of Figure 4 is the matrix multiplications in step 5 and 8. In each iteration, each party performs 4 such matrix multiplications[5] while in plaintext SGD training, according to Equation 2, 2 matrix multiplications are performed. Hence, the computation time for each party is only twice the time for training on plaintext data.

The total communication of the protocol is also nearly optimal. In step 1, each party sends an $n \times d$ matrix, which is of the same size as the data. In step 4 and 7, $|B| + d$ elements are sent per iteration. Therefore, the total communication is $n \cdot d + (|B| + d) \cdot t = nd \cdot (1 + \frac{E}{d} + \frac{E}{|B|})$ for each party. In practice, the number of epochs E is only 2-3 for linear and logistic regressions and 10-15 for neural networks, which is much smaller than $|B|$ and d. Therefore, the total communication is only a little more than the size of the data. The time spent on the communication can be calculated by dividing the total communication by the bandwidth between the two parties.

协议的总通信也几乎是最佳的。在步骤1中，每一方发送n×d矩阵，其尺寸与数据相同。在步骤4和7中，每次迭代发送|B|+d个元素。因此，每一个迭代总的沟通是

离线阶段

4.2 The Offline Phase

在实践中，对于线性和逻辑回归，时期E的数量仅为2-3，对于神经网络，时期数量为10-15，这比|B|和d小得多。因此，总通信量只比数据量大一点。可以通过将总通信量除以双方之间的带宽来计算在通信上花费的时间。

我们描述了如何通过生成所需的共享乘法三元组来将离线阶段实现为S0和S1之间的两方协议。我们提出了两种基于线性同态加密（LHE）和不经意转移（OT）的协议。这些技术类似于先前的工作（例如，[17]），但针对我们对矩阵进行操作的矢量化场景进行了优化。因此，我们的离线协议的复杂性要比生成独立乘法三元组的天真方法好得多

We describe how to implement the offline phase as a two-party protocol between $\mathcal{S}_0$ and $\mathcal{S}_1$ by generating the desired shared multiplication triplets. We present two protocols for doing so based on linearly homomorphic encryption (LHE) and oblivious transfer (OT). The techniques are similar to prior work (e.g., [17]) but are optimized for the vectorized scenario where we operate on matrices. As a result the complexity of our offline protocol is much better than the naive approach of generating independent multiplication triplets.

Recall that given shared random matrices $\langle \mathbf{U} \rangle$ and $\langle \mathbf{V} \rangle$, the key step is to choose a $|B| \times d$ submatrix from $\langle \mathbf{U} \rangle$ and a column from $\langle \mathbf{V} \rangle$ and compute the shares of their product. This is repeated t times to generate $\langle \mathbf{Z} \rangle$. $\langle \mathbf{Z}' \rangle$ is computed in the same way with the dimensions reversed. Thus, for simplicity, we focus on this basic step, where given shares of a $|B| \times d$ matrix $\langle \mathbf{A} \rangle$, and shares of a $d \times 1$ matrix $\langle \mathbf{B} \rangle$, we want to compute shares of a $|B| \times 1$ matrix $\langle \mathbf{C} \rangle$ such that $\mathbf{C} = \mathbf{A} \times \mathbf{B}$.

We utilize the following relationship: $\mathbf{C} = \langle \mathbf{A} \rangle_0 \times \langle \mathbf{B} \rangle_0 + \langle \mathbf{A} \rangle_0 \times \langle \mathbf{B} \rangle_1 + \langle \mathbf{A} \rangle_1 \times \langle \mathbf{B} \rangle_0 + \langle \mathbf{A} \rangle_1 \times \langle \mathbf{B} \rangle_1$. It suffices to compute $\langle \langle \mathbf{A} \rangle_0 \times \langle \mathbf{B} \rangle_1 \rangle$ and $\langle \langle \mathbf{A} \rangle_1 \times \langle \mathbf{B} \rangle_0 \rangle$ as the other two terms can be computed locally.

LHE-based generation. To compute the shares of the product $\langle \mathbf{A} \rangle_0 \times \langle \mathbf{B} \rangle_1$, $\mathcal{S}_1$ encrypts each element of $\langle \mathbf{B} \rangle_1$ using an LHE and sends them to $\mathcal{S}_0$. The LHE can be initiated using the cryptosystem of Paillier [37] or Damgard-Geisler-Kroigaard(DGK) [16]. $\mathcal{S}_0$ then performs the matrix multiplication on the ciphertexts, with additions replaced by multiplications and multiplications by exponentiations. Finally, $\mathcal{S}_0$ masks the resulting ciphertexts by random values, and sends them back to $\mathcal{S}_1$ to decrypt. The protocol can be found in Figure 6.

[5] Party $\mathcal{S}_1$ can simplify the formula to $\mathbf{E} \times (\langle \mathbf{w} \rangle - \mathbf{F}) + \langle \mathbf{X} \rangle \times \mathbf{F} + \langle \mathbf{Z} \rangle$, which has only 2 matrix multiplications.

基于LHE的一代。为了计算乘积A0 * B1的份额，S1使用LHE加密B1的每个元素并将它们发送到S0。LHE可以使用Paillier [37]或Damgard-Geisler-Kroigaard（DGK）[16]的密码系统初始化。S0然后在密文上执行矩阵乘法，加法替换为乘法，乘法乘以取幂。最后，S0通过随机值屏蔽得到的密文，并将它们发送回S1进行解密。该协议可以在图6中找到。

Protocol $\mathsf{LHE\_MT}(\langle\mathbf{A}\rangle_0; \langle\mathbf{B}\rangle_1)$:

(Let a_{ij} be the (i,j)th element in $\langle\mathbf{A}\rangle_0$ and b_j be the jth element in $\langle\mathbf{B}\rangle_1$.)

1: $\mathcal{S}_1 \to \mathcal{S}_0$: $\mathsf{Enc}(b_j)$ for $i = 1, \ldots, d$.

2: $\mathcal{S}_0 \to \mathcal{S}_1$: $c_i = \Pi_{j=0}^{d} \mathsf{Enc}(b_j)^{a_{ij}} \cdot \boxed{\mathsf{Enc}(r_i)}$, for $i = 1, \ldots, |B|$.

3: $\mathcal{S}_0$ sets $\langle\langle\mathbf{A}\rangle_0 \times \langle\mathbf{B}\rangle_1\rangle_0 = \mathbf{r}$, where $\mathbf{r} = (-r_1, \ldots, -r_{|B|})^T \mod 2^l$.

4: $\mathcal{S}_1$ sets $\langle\langle\mathbf{A}\rangle_0 \times \langle\mathbf{B}\rangle_1\rangle_1 = (\mathsf{Dec}(c_1), \ldots, \mathsf{Dec}(c_{|B|}))^T$,

Figure 6: The offline protocol based on linearly homomorphic encryption.

图6：基于线性同态加密的离线协议。

这里S1执行d加密，|B| 解密和S0执行|B|×d指数。 密文上的乘法成本是非支配的并且被省略可以类似地计算A1 × B0的份额

Here $\mathcal{S}_1$ performs d encryptions, $|B|$ decryptions and $\mathcal{S}_0$ performs $|B| \times d$ exponentiations. The cost of multiplications on the ciphertext is non-dominating and is omitted. The shares of $\langle\mathbf{A}\rangle_1 \times \langle\mathbf{B}\rangle_0$ can be computed similarly.

Using this basic step, the overall computation performed in the offline phase per party is $(|B|+d) \cdot t$ encryptions, $(|B|+d) \cdot t$ decryptions and $2|B| \cdot d \cdot t$ exponentiations. The total communication is $2(|B|+d) \cdot t$ ciphertexts, which is much smaller than the size of the data. If we had generated the multiplication triplets independently, the number of encryptions, decryptions and the communication would increase to $2|B| \cdot d \cdot t$. Finally, unlike the online phase, all communication in the offline phase can be done in one interaction.

使用该基本步骤，在每一方的离线阶段中执行的总计算是（| B | + d）·t加密，（| B | + d）·t解密和2 | B |·d·t指数。 总通信为2（| B | + d）·t密文，远小于数据的大小。如果我们独立生成乘法三元组，则加密，解密和通信的数量将增加到2 | B |·d·t。 最后，与在线阶段不同，o、离线阶段的所有通信都可以在一次交互中完成。

OT-based generation. The shares of the product $\langle\mathbf{A}\rangle_0 \times \langle\mathbf{B}\rangle_1$ can also be computed using OTs. We first compute the shares of the product $\langle a_{ij} \cdot b_j\rangle$ for all $i = 1, \ldots, |B|$ and $j = 1, \ldots, d$. To do so, $\mathcal{S}_1$ uses each bit of b_j to select two values computed from a_{ij} using correlated OTs. In particular, for $k = 1, \ldots, l$, $\mathcal{S}_0$ sets the correlation function of COT to $f_k(x) = a_{i,j} \cdot 2^k + x \mod 2^l$ and $\mathcal{S}_0$, $\mathcal{S}_1$ run $\mathsf{COT}(r_k, f_k(x); b_j[k])$. If $b_j[k] = 0$, $\mathcal{S}_1$ gets r_k; if $b_j[k] = 1$, $\mathcal{S}_1$ gets $a_{i,j} \cdot 2^k + r_k \mod 2^l$. This is equivalent to $b_j[k] \cdot a_{ij} \cdot 2^k + r_k \mod 2^l$. Finally, $\mathcal{S}_1$ sets $\langle a_{ij} \cdot b_j\rangle_1 = \sum_{k=1}^{l}(b_j[k] \cdot a_{ij} \cdot 2^k + r_k) = a_{ij} \cdot b_j + \sum_{k=1}^{l} r_k \mod 2^l$, and $\mathcal{S}_0$ sets $\langle a_{ij} \cdot b_j\rangle_0 = \sum_{k=1}^{l}(-r_k) \mod 2^l$.

为了进一步提高效率，[17]的作者观察到，对于每个k，aij·2k的最后k位都是0。因此，只需要传输第一个l-k位。因此，消息长度为l, l-1, ..., 1, 而不是全部为l位。 这相当于运行COT(l + 1)/2的l个实例到目前为止，所描述的所有技术都在[17]中讨论过

To further improve efficiency, authors of [17] observe that for each k, the last k bits of $a_{ij} \cdot 2^k$ are all 0s. Therefore, only the first $l-k$ bits need to be transferred. Therefore, the message lengths are $l, l-1, \ldots, 1$, instead of all being l-bits. This is equivalent to running l instances of $\mathrm{COT}_{(l+1)/2}$. So far, all the techniques described are as discussed in [17].

The optimization described above does not improve the computation cost of OTs. The reason is that in OT, each message is XORed with a mask computed from the random oracle applied to the selection bit. In practice, the random oracle is instantiated by a hash function such as SHA256 or AES, which at least has 128 bit output. Hence, the fact that l is only 64 does not reduce time to compute the masks.

上述优化不会改善OT的计算成本。原因是在OT中，每个消息与从应用于选择位的随机预言计算的掩码进行异或。 在实践中，随机oracle由诸如SHA256或AES的散列函数实例化，其至少具有128位输出。 因此，l仅为64的事实不会减少计算掩模的时间。

We further leverage the matrix structure to improve on this. Note that $a_{1j}, \ldots, a_{|B|j}$ are all multiplied by b_j, which means the same selection bit $b_j[k]$ is used for all a_{ij}s. Equivalently, we can view it as using $b_j[k]$ to select messages with length $(l-k) \cdot |B|$ bits. Therefore, they can be masked by $\lceil \frac{(l-k)\cdot|B|}{128} \rceil$ hash outputs. For a reasonable mini-batch size, each multiplication needs $\frac{l}{4}$ instances of COT_{128}. In this way, the total number of hashes can be reduced by a factor of 4 and the total communication can be reduced by a factor of 2.

Finally, after computing $\langle a_{ij} \cdot b_j\rangle$, the ith element of $\langle\langle\mathbf{A}\rangle_0 \times \langle\mathbf{B}\rangle_1\rangle$ can be computed by $\langle\langle\mathbf{A}\rangle_0 \times \langle\mathbf{B}\rangle_1\rangle[i] = \sum_{j=0}^{d}\langle a_{ij} \cdot b_j\rangle$. The shares of $\langle\mathbf{A}\rangle_1 \times \langle\mathbf{B}\rangle_0$ can be computed similarly.

In total, both parties perform $\frac{|B|\cdot d\cdot t\cdot l}{2}$ instances of COT_{128} and the total communication is

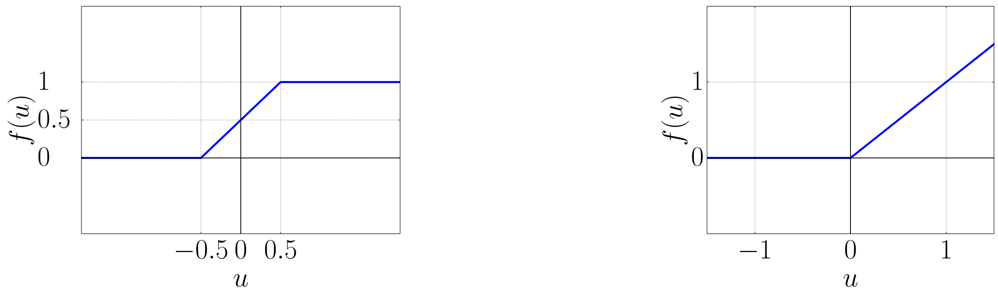


Figure 7: (a) Our new activation function. (b) RELU function.

$|B| \cdot d \cdot t \cdot l \cdot (l + \lambda)$ bits. In addition, a set of base OTs need to be performed at the beginning for OT extension. In Section 6.1 we show that the size of communication for the OT-based generation is much higher than the LHE-based generation, yet the total running time is faster. The reason is that, given OT extension, each OT operation is very cheap ($\sim 10^6$ OTs per second).

在本节中，我们提出了一个支持隐私保护逻辑回归的协议。除了针对线性回归解决的问题之外，主要的额外挑战是在共享数字上计算逻辑函数

隐私保留逻辑回归

4.3 Privacy Preserving Logistic Regression

In this section, we present a protocol to support privacy preserving logistic regression. Besides issues addressed for linear regression, the main additional challenge is to compute the logistic function $f(u) = \frac{1}{1+e^{-u}}$ on shared numbers. Note that the division and the exponentiation in the logistic function are computed on real numbers, which are hard to support using a 2PC for arithmetic or boolean circuit. Hence, prior work proposes to approximate the function using polynomials [9]. It can be shown that approximation using a high-degree polynomial is very accurate [32]. However, for efficiency reasons, the degree of the approximation polynomial in secure computation is set to 2 or 3, which results in a large accuracy loss of the trained model compared to logistic regression.

注意，逻辑函数中的除法和取幂是根据实数计算的，使用2PC算术或布尔电路很难支持。因此，先前的工作建议使用多项式逼近函数[9]。可以证明，使用高次多项式的近似是非常准确的[32]。然而，出于效率原因，安全计算中的近似多项式的程度被设置为2或3，这导致与逻辑回归相比训练模型的大的准确度损失

安全的计算友好激活功能。我们提出了一种新的激活函数，而不是使用多项式逼近逻辑函数，这种激活函数可以使用安全计算技术进行有效计算。该函数在等式4中描述并在图7(a)中绘出。

Secure computation friendly activation functions. Instead of using polynomials to approximate the logistic function, we propose a new activation function that can be efficiently computed using secure computation techniques. The function is described in Equation 4 and drawn in Figure 7(a).

$$f(x) = \begin{cases} 0, & \text{if } x < -\frac{1}{2} \\ x + \frac{1}{2}, & \text{if } -\frac{1}{2} \le x \le \frac{1}{2} \\ 1, & \text{if } x > \frac{1}{2} \end{cases} \tag{4}$$

The intuition for this choice of activation is as follows (we also confirm its effectiveness with experiments): as mentioned in section 2.1, the main reason logistic regression works well for classification problems is that the prediction is bounded between 0 and 1. Therefore, it is very important for the two tails of the activation function to converge to 0 and 1, and both the logistic function and the function in Equation 4 have such behavior. In contrast, approximation with low degree polynomials fails to achieve this property. The polynomial might be close to the logistic function in certain intervals, but the tails are unbounded. If a data sample yields a very large input u to the activation function, $f(u)$ will be far beyond the $[0, 1]$ interval which affects accuracy of the model significantly in the backward propagation. Our choice of the activation function is also inspired by its similarity to the RELU function (Figure 7(b)) used in neural networks. One of the justifications used for replacing logistic function by the RELU function in neural networks is that

这种激活选择的直觉如下（我们还通过实验证实了它的有效性）：如2.1节所述，逻辑回归在分类问题中运行良好的主要原因是预测在0和1之间。因此，它对于激活函数的两个尾部收敛到0和1是非常重要的，并且逻辑函数和等式4中的函数都具有这样的行为。相反，低次多项式的近似不能实现这种性质。多项式可能在某些时间间隔内接近逻辑函数，但尾部是无界的。如果数据样本对激活函数产生非常大的输入u，则f(u)将远远超出[0,1]间隔，这在后向传播中显着地影响模型的准确性。我们对激活功能的选择也受其与神经网络中使用的RELU功能(图7(b))的相似性的启发。用于通过神经网络中的RELU函数替换逻辑函数的一个原因是用o ff集减去两个RELU函数产生等式4的激活函数，其反过来非常模仿逻辑函数。

	Logistic	Our approaches		Polynomial Approx.		
		first	second	deg. $\boxed{2}$	deg. 5	deg. 10
MNIST	98.64	98.62	97.96	42.17	84.64	98.54
Arcene	86	86	85	72	82	86

Table 1: Accuracy (%) comparison of different approaches for logistic regression.

表1：逻辑回归的不同方法的准确度（%）比较。

the subtraction of two RELU functions with an offset yields the activation function of Equation 4 which in turn, closely imitates the logistic function.

Once we use the new activation function, we have two choices when computing the backward propagation. We can either use the same update function as the logistic function (i.e. continue to compute the partial derivative using the logistic function), or compute the partial derivative of the new function and substitute it into the update function. We test both options and find out that the first approach yields better accuracy matching that of using the logistic function. Therefore, we will use the first approach in the rest of the paper. We believe one reason for lower accuracy of the second approach is that by replacing the activation function, the cross entropy cost function is no longer convex; using the first approach, the update formula is very close to training using the distance cost function, which might help produce a better model. Better theoretical analysis of these observations is an interesting research direction.

一旦我们使用新的激活函数，我们在计算反向传播时有两个选择。我们可以使用与逻辑函数相同的更新函数（即继续使用逻辑函数计算偏导数），或者计算新函数的偏导数并将其替换为更新函数。我们测试了两个选项，并发现第一种方法可以获得与使用逻辑函数相匹配的更高精度。因此，我们将在本文的其余部分使用第一种方法。我们认为第二种方法精度较低的一个原因是，通过替换激活函数，交叉熵代价函数不再是凸的;使用第一种方法，更新公式非常接近使用距离成本函数的训练，这可能有助于产生更好的模型。更好的理论分析这些观察是一个有趣的研究方向。

To justify our claims, we compare the accuracy of the produced model using our approaches with logistic regression, and polynomial approximation with different degrees. For the polynomial approximation, we fix the constant to $\frac{1}{2}$ so that $f(0) = \frac{1}{2}$. Then we select as many points on the logistic function as the degree of the polynomial. The points are symmetric to the original, and evenly spread in the range of the data value (e.g., [0,1] for MNIST, [0,1000] for Arcene). The unique polynomial passing through all these points is selected for approximation. The test is run on the MNIST data with mini-batch size $|B| = 128$. The series of random mini-batches are the same for all approaches. Here we train the models on plaintext data only. As shown in Table 1, the performance of our approaches are much better than polynomial approximation. In particular, our first approach reaches almost the same accuracy (98.62%) as logistic regression, and our second approach performs slightly worse. On the contrary, when a degree 3 polynomial is used to approximate the logistic function, the accuracy can only reach 42.17%, which is even worse than a linear regression. The reason is that the tails diverge even faster than a linear activation function. When the degree is 5, the accuracy can reach 84%; when the degree is 10, the accuracy finally matches that of logistic regression. However, computing a polynomial of degree 10 in secure computation introduces a high overhead. Similar effects are also verified by experiments on the Arcene dataset.

为了证明我们的主张，我们使用我们的方法与逻辑回归和不同程度的多项式近似来比较生成模型的准确性。对于多项式近似，我们将常数固定为0.5，使得f(0)= 0.5. 然后我们在逻辑函数上选择与多项式的次数一样多的点。这些点与原始对称，并且均匀地分布在数据值的范围内(例如，对于MNIST为[0,1],对于Arcene为[0,1000]). 选择通过所有这些点的唯一多项式用于近似。测试在具有小批量大小|B|的MNIST数据上运行 对于所有方法，一系列随机小批量是相同的。在这里，我们仅在明文数据上训练模型。

如表1所示，我们的方法的性能比多项式近似要好得多。特别是，我们的第一种方法与逻辑回归达到几乎相同的准确度(98.62%), 而我们的第二种方法表现稍差。相反，当使用3次多项式逼近逻辑函数时，精度只能达到42.17%，甚至比线性回归更差。原因是尾部比线性激活功能发散得更快。当学位为5时，准确率可达到84%; 当度数为10时，精度最终与逻辑回归的精度相匹配。然而，在安全计算中计算度数为10的多项式引入了高开销。类似的效果也通过Arcene数据集上的实验进行了验证。

Nevertheless, we suggest furthur work to explore more MPC-friendly activation functions that can be computed efficiently using simple boolean or arithmetic circuits.

然而，我们建议进一步研究更多MPC友好的激活函数，这些函数可以使用简单的布尔或算术电路有效地计算

The privacy preserving protocol. The new activation function proposed above is circuit friendly. It only involves testing whether the input is within the $[-1/2, 1/2]$ interval. However, applying Yao's garbled circuit protocol naively to the whole logistic regression is very inefficient. Instead, we take advantage of techniques to switch between arithmetic sharing and Yao sharing proposed in [17]. The observation is that as mentioned in Section 2.1, the only difference between the SGD for logistic regression and linear regression is the application of an extra activation function in each forward propagation. Therefore, following the same protocol for privacy preserving linear regression, after computing the inner product of the input data and the coefficient vector, we switch the arithmetic

隐私保护协议。上面提出的新激活功能是电路友好的。它只涉及测试输入是否在[-1/2,1/2]区间内。然而，将Yao的乱码电路协议天真地应用于整个逻辑回归是非常低效的。相反，我们利用技术在[17]中提出的算术共享和姚共享之间切换。观察结果如第2.1节所述，SGD用于逻辑回归和线性回归之间的唯一差异是在每个前向传播中应用额外的激活函数。因此，遵循相同的隐私保护线性回归协议，在计算输入数据的内积和系数向量之后，我们将算术共享切换到Yao共享并使用乱码电路评估激活函数。然后，我们切换回算术共享并继续向后传播。

Protocol SGD_Logistic($\langle\mathbf{X}\rangle, \langle\mathbf{Y}\rangle, \langle\mathbf{U}\rangle, \langle\mathbf{V}\rangle, \langle\mathbf{Z}\rangle, \langle\mathbf{V'}\rangle, \langle\mathbf{Z'}\rangle$):

```
Do step 1–5 as in Figure 4. Both parties obtain the shares ⟨U_{B_i}⟩ = ⟨X_{B_i} × w⟩ (it was
  defined as ⟨Y*_{B_i}⟩ in Figure 4).
for every element ⟨u⟩ in ⟨U_{B_i}⟩ do
    (⟨b_3⟩^B, ⟨b_4⟩^B) ← Y2B(GarbledCircuit(⟨u⟩_0 + 1/2, ⟨u⟩_0 − 1/2; ⟨u⟩_1, f)), where f sets b_1 as the
    most significant bit of (⟨u⟩_0 + 1/2) + ⟨u⟩_1 and b_2 as the most significant bit of (⟨u⟩_0 − 1/2) + ⟨u⟩_1.
    It then outputs b_3 = ¬b_1 and b_4 = b_1 ∧ (¬b_2).
    S_0 sets m_0 = ⟨b_4⟩_0^B · ⟨u⟩_0 + r_1 and m_1 = (1 − ⟨b_4⟩_0^B) · ⟨u⟩_0 + r_1. S_0 and S_1 run (⊥; m_{⟨b_4⟩_1^B}) ←
    OT(m_0, m_1; ⟨b_4⟩_1^B). m_{⟨b_4⟩_1^B} is equal to (⟨b_4⟩_0^B ⊕ ⟨b_4⟩_1^B) · ⟨u⟩_0 + r_1 = b_4 · ⟨u⟩_0 + r_1.
    P_1 sets m_0 = ⟨b_4⟩_1^B · ⟨u⟩_1 + r_2 and m_1 = (1 − ⟨b_4⟩_1^B) · ⟨u⟩_1 + r_2. S_1 and S_0 run
    (⊥; m_{⟨b_4⟩_0^B}) ← OT(m_0, m_1; ⟨b_4⟩_0^B). m_{⟨b_4⟩_0^B} is equal to b_4 · ⟨u⟩_1 + r_2.
    S_0 sets m_0 = ⟨b_3⟩_0^B + r_3 and m_1 = (1 − ⟨b_3⟩_0^B) + r_3. S_0 and S_1 run (⊥; m_{⟨b_3⟩_1^B}) ←
    OT(m_0, m_1; ⟨b_3⟩_1^B). m_{⟨b_3⟩_1^B} is equivalent to b_3 + r_3.
    S_0 sets ⟨y*⟩_0 = m_{⟨b_4⟩_0^B} − r_1 − r_3 and S_1 sets ⟨y*⟩_1 = m_{⟨b_4⟩_1^B} + m_{⟨b_3⟩_1^B} − r_2.
end for
Both parties set ⟨Y*⟩_i as a vector of all ⟨y*⟩_is computed above and continue to step
  6–12 in Figure 4.
```

Figure 8: Privacy preserving logistic regression protocol.

在这里，我们提出了一个更复杂的协议，以进一步优化电路大小，交互次数和使用的乘法三元组的数量

sharing to a Yao sharing and evaluate the activation function using a garbled circuit. Then, we switch back to arithmetic sharing and continue the backward propagation.

Here, we propose a more involved protocol to further optimize the circuit size, the number of interactions and the number of multiplication triplets used. Note that if we let $b_1 = 0$ if $u + \frac{1}{2} \geq 0$, $b_1 = 1$ otherwise, and $b_2 = 0$ if $u - \frac{1}{2} \geq 0$, $b_2 = 1$ otherwise, then the activation function can be expressed as $f(u) = (\neg b_2) + (b_2 \wedge (\neg b_1))u$. Therefore, given $\langle u \rangle$, we construct a garbled circuit that takes the bits of $\langle u + \frac{1}{2} \rangle_0$ and $\langle u \rangle_1$ as input, adds them and sets b_1 as the most significant bit (msb) of the result (the msb indicates whether a value is positive or negative). To be more precise, the "$+\frac{1}{2}$" value is represented in the field and scaled to have the same number of bit representing the fractional part as u. In particular, since u is the product of two values before truncation, "$+\frac{1}{2}$" is expressed as $\frac{1}{2} \cdot 2^{l_u}$, where l_u is the sum of bit-length of the decimal part in the data x and the coefficient w, but we use $+\frac{1}{2}$ for ease of presentation. b_2 is computed in a similar fashion. Instead of computing the rest of the function in the garbled circuit which would require a linear number of additional AND gates, we let the garbled circuit output the Yao sharing (output labels) of the bits $(\neg b_2)$ and $b_2 \wedge (\neg b_1)$. We then switch to boolean sharing of these bits and use them in two OTs to compute $\langle (\neg b_2) + (b_2 \wedge (\neg b_1))u \rangle$ and continue with the rest of the training. The detailed protocol is described in Figure 8. The following theorem states the security of privacy-preserving logistic regression. The proof is omitted due to lack of space but we note that it is implied by the security of the secret sharing scheme, the garbling scheme, and OT.

详细协议如图8所示。以下定理说明了保护隐私的逻辑回归的安全性。由于缺乏空间而省略了证明，但我们注意到秘密共享方案，乱码方案和OT的安全性暗示了这一点

Theorem 3. *Consider a protocol where clients distribute arithmetic shares of their data among two servers who run the protocol of Figure 8 and send the output to clients. Given a secure garbling scheme, in the $\mathcal{F}_{offline}$ and $\mathcal{F}_{ot}$ hybrid model, this protocol realizes the ideal functionality $\mathcal{F}_{ml}$ of Figure 3 for the logistic regression function, in presence of a semi-honest* admissible *adversary (see*

定理3.考虑一种协议，其中客户端在运行图8协议的两个服务器之间分配其数据的算术份额，并将输出发送给客户端。鉴于一种安全的拼接方案，在Foffline和Fot混合模型中，该协议实现了逻辑回归函数的图3的理想函数Fml，存在半诚实的可接受对手（参见第3节）。

section 3).

效率讨论。逻辑回归的额外开销非常小。大多数步骤与4.1节中的线性回归协议完全相同。另外，在每个前向传播中执行一个乱码电路协议和3个额外OT。乱码电路执行两次加法和一次AND，为每个值u产生总共2l -1个AND门。OT扩展的基础OT可以在o ffl ine阶段执行。因此，对于每一方，总通信开销是| B |·t·（（2l-1）·2λ+ 3l）。注意，来自S0的乱码电路和OT中的消息可以同时发送到S1。因此，逻辑回归仅在每次迭代中引入一次以上的交互，并且在两方之间产生总共3t的交互。由于我们不使用激活函数的算术运算，因此不需要额外的乘法三元组

Efficiency Discussion. The additional overhead of the logistic regression is very small. Most of the steps are exactly the same as the linear regression protocol in Section 4.1. In addition, one garbled circuit protocol and 3 extra OTs are performed in each forward propagation. The garbled circuit performs two additions and one AND, yielding a total $2l - 1$ AND gates for each value u. The base OT for OT extension can be performed in the offline phase. Therefore, the total communication overhead is $|B| \cdot t \cdot ((2l-1) \cdot 2\lambda + 3l)$ for each party. Note that the garbled circuit and the messages in OTs from $\mathcal{S}_0$ can be sent simultaneously to $\mathcal{S}_1$. Thus, the logistic regression only introduces one more interaction per iteration, and yields a total of $3t$ interactions between the two parties. No extra multiplication triplets are required since we do away with arithmetic operations for the activation function.

隐私保护神经网络训练

4.4 Privacy Preserving Neural Network Training

我们提出的用于隐私保护线性和逻辑回归的所有技术都可以扩展到支持隐私保护神经网络训练。我们可以使用RELU函数作为每个神经元中的激活函数，并使用交叉熵函数作为成本函数。每个神经元中每个系数的更新函数可以以第2.1节中讨论的封闭形式表示。除了评估激活函数及其偏导数之外，前向和后向传播中的所有函数仅涉及简单的加法和乘法，并且使用针对线性回归所讨论的相同技术来实现。为了评估RELU函数f(u)=(u> 0)·u及其导数f'(u)=(u> 0)，我们使用与逻辑回归相同的方法切换到Yao共享。乱码电路简单地添加了两个共享并输出最重要的位，这比我们新的逻辑函数所需的电路更简单。注意，RELU函数及其导数可以在一次迭代中一起评估，后者的结果用于后向传播

All techniques we proposed for privacy preserving linear and logistic regression naturally extend to support privacy preserving neural network training. We can use the RELU function as the activation function in each neuron and the cross entropy function as the cost function. The update function for each coefficient in each neuron can be expressed in a closed form as discussed in Section 2.1. All the functions in both forward and backward propagation, other than evaluating the activation function and its partial derivative, involve only simple additions and multiplications, and are implemented using the same techniques discussed for linear regression. To evaluate the RELU function $f(u) = (u > 0) \cdot u$ and its derivative $f'(u) = (u > 0)$, we use the same approach as for logistic regression by switching to Yao sharing. The garbled circuit simply adds the two shares and outputs the most significant bit, which is even simpler than the circuit we needed for our new logistic function. Note that both the RELU function and its derivative can be evaluated together in one iteration, and the result of the latter is used in the backward propagation.

我们还提出了softmax函数的安全计算友好替代方案

我们首先用RELU函数替换分子中的取幂，使得结果仍然是e-ui所预期的非负的.然后,我们通过添加所有RELU 函数的输出来计算总和，并使用除法乱码电路将每个输出除以总和. 这样，输出保证是概率分布. 在实验部分，我们展示了使用示例神经网络和MNIST数据集上的训练，由TensorFlow (使用softmax)训练的模型在所有10个类别上可以达到94.5%的准确度,而使用我们提出的函数我们达到93.4 %. 由于空间限制,我们省略了对协议的详细描述

We also propose a secure computation friendly alternative to the softmax function $f(u_i) = \frac{e^{-u_i}}{\sum_{i=1}^{d_m} e^{-u_i}}$. We first replace the exponentiations in the numerator with RELU functions such that the results remain non-negative as intended by e^{-u_i}. Then, we compute the total sum by adding the outputs of all RELU functions, and divide each output by the total sum using a division garbled circuit. In this way, the output is guaranteed to be a probability distribution[6]. In the experiment section we show that using an example neural network and training on the MNIST dataset, the model trained by Tensorflow (with softmax) can reach 94.5% accuracy on all 10 classes, while we reach 93.4% using our proposed function. We omit a detailed description of the protocol due to space limits.

As we observe in our experiments, the time spent on garbled circuits for the RELU functions dominates the online training time. Therefore, we also consider replacing the activation function with the square function $f(u) = u^2$, as recently proposed in [21] but for prediction only. (We still use RELU functions for approximating softmax.) With this modification, we can reach 93.1% accuracy. Now a garbled circuit computing a RELU function is replaced by a multiplication on shared values, thus the online efficiency is improved dramatically. However, this approach consumes more multiplication triplets and increases cost of the offline phase.

[6] If the sum is 0, which means all the results of RELU functions are 0s, we assign the same probability to each output. This is done with a garbled circuit.

正如我们在实验中观察到的那样，RELU功能在乱码电路上花费的时间主导了在线训练时间。因此，我们还考虑用方形函数f(u)= u^2替换激活函数，如[21]中最近提出的那样，但仅用于预测。（我们仍然使用RELU函数来逼近softmax。）通过这种修改，我们可以达到93.1%的准确度。现在，计算RELU功能的乱码电路被共享值的乘法所取代，从而显着提高了在线效率。然而，这种方法消耗更多的乘法三元组并增加了相位的成本。

效率讨论。 在在线阶段，计算复杂度是矩阵算术运算的明文训练的两倍，加上使用乱码电路和OT评估RELU功能和划分的开销。 在我们的实验中，我们使用EMP工具包[3]中的除法电路，它具有用于l位数的O(l^2)AND门。 总通信是矩阵乘法和逐元乘法所涉及的所有矩阵的大小之和，即……., 迭代总数为5m·t。

离线阶段，与回归相比，乘法三联体的总数增加了一个因子O(Pm i = 1 dm)，回归正好是神经网络中神经元的数量。一些乘法三元组可以以矩阵形式生成，用于在线矩阵乘法。其他需要独立生成以进行逐元素乘法。我们在6.3节中通过实验显示了成本。

Efficiency Discussion. In the online phase, the computation complexity is twice that of the plaintext training for the matrix arithmetic operations, plus the overhead of evaluating the RELU functions and divisions using garbled circuits and OTs. In our experiments, we use the division circuit from the EMP toolkit [3], which has $O(l^2)$ AND gates for l-bit numbers. The total communication is the sum of the sizes of all matrices involved in the matrix multiplication and element-wise multiplication, which is $O(t \cdot \sum_{i=1}^{m}(|B| \cdot d_{i-1} + d_{i-1} \cdot d_i))$. The total number of iterations is $5m \cdot t$.

In the offline phase, the total number of multiplication triplets is increased by a factor of $O(\sum_{i=1}^{m} d_m)$ compared to regression, which is exactly the number of neurons in the neural network. Some of the multiplication triplets can be generated in the matrix form for online matrix multiplication. Others need to be generated independently for element-wise multiplications. We show the cost experimentally in Section 6.3.

到目前为止开发的技术还可以用于安全地进行预测，因为预测仅仅是训练中一次迭代的前向传播分量。 同样，我们也可以安全地测试每个时期之后当前模型的准确性，因为准确性只是测试数据预测的汇总结果。 准确度测试可用于调整学习率或决定何时终止训练，而不是使用固定的学习率并且训练模型以获得固定数量的时期

预测和准确度测试

4.5 Predictions and Accuracy Testing

The techniques developed so far can also be used to securely make predictions, since the prediction is simply the forward propagation component of one iteration in the training. Similarly, we can also test the accuracy of the current model after each epoch securely, as the accuracy is simply an aggregated result of the predictions on the testing data. The accuracy test can be used to adjust the learning rate or decide when to terminate the training, instead of using a fixed learning rate and training the model for a fixed number of epochs.

隐私保护预测。该算法与计算线性回归，逻辑回归和神经网络的预测值y *完全相同，并且成本仅为一次迭代的一半。我们在第6.4节中展示了我们保护隐私的预测的性能。我们迭代我们可以隐藏输入数据，模型，预测结果或它们的任何组合，因为它们都可以在我们的协议中秘密共享。如果可以显示输入数据或模型，则可以进一步提高效率。例如，如果模型是明文的，则可以直接在没有预先计算的乘法三元组的共享上计算输入数据与系数的乘法。

Privacy preserving prediction. The algorithm is exactly the same as computing the predicted value y^* for linear regression, logistic regression and neural networks and the cost is only half of one iteration. We show the performance of our privacy-preserving predictions in Section 6.4. We iterate that we can hide the input data, the model, the prediction result or any combinations of them, as they can all be secret shared in our protocols. If either the input data or the model can be revealed, the efficiency can be further improved. E.g., if the model is in plaintext, the multiplications of the input data with the coefficients can be computed directly on the shares without precomputed multiplication triplets.

在分类问题中，预测通常四舍五入到最接近的类。 例如，在逻辑回归中，如果预测值是0.8，则数据可能被分类为1，并且确切的结果可以揭示关于输入的额外信息。 这种舍入可以被视为测试秘密共享值减去0.5是否大于0，并且可以通过应用额外的乱码电路来支持，类似于我们近似逻辑函数的方式。 乱码电路将添加两个份额并输出最重要的位。

In classification problems, the prediction is usually rounded to the closest class. E.g., in logistic regression, if the predicted value is 0.8, the data is likely to be classified as 1, and the exact result may reveal extra information on the input. This rounding can be viewed as testing whether a secret shared value minus $\frac{1}{2}$ is larger than 0, and can be supported by applying an extra garbled circuit, similar to how we approximated the logistic function. The garbled circuit would add the two shares and output the most significant bit.

隐私保护准确性测试。 确定学习速率的一种简单方法是预先在同一类别的一些不敏感数据上进行测试，并将其设置为常数，而不需要在整个训练过程中进行任何调整。 类似地，迭代次数可以提前固定。

Privacy preserving accuracy testing. A simple way to decide the learning rate is to test it on some insensitive data of the same category beforehand, and set it to a constant without any adjustment throughout training. Similarly, the number of iterations can be fixed in advance.

At the cost of some leakage, we propose an alternative solution that enables adjusting the rate and number of iteration in the same fashion as plaintext training. To do so, we need to test the accuracy of the current model after each epoch on a testing dataset. As a first step, we simply perform a privacy preserving prediction for each testing data sample. Then, we test whether it is the same as the label and aggregate the result. Again we use a simple garbled circuit to perform the equality test, in which the number of gates is linear in the bit length of the values. Finally, each party sums up all the secret-shared results of equality tests as the shared accuracy. The cost of doing so is only running half of an iteration plus some extra garbled circuits for rounding and

以一些泄漏为代价，我们提出了一种替代解决方案，能够以与明文训练相同的方式调整迭代率和次数。 为此，我们需要在测试数据集的每次迭代之后测试当前模型的准确性。 作为第一步，我们只是为每个测试数据样本执行隐私保护预测。 然后，我们测试它是否与标签相同并聚合结果。 我们再次使用简单的乱码电路来执行相等测试，其中门的数量在值的位长度上是线性的。 最后，每一方都将平等测试的所有秘密共享结果总结为共享准确性。 这样做的成本只是运行一半的迭代加上一些额外的乱码电路进行舍入和相等测试。 由于测试数据的大小通常明显小于训练数据，因此精确度测试所花费的时间只是训练的一小部分。

为了调整学习速率，我们使用乱码电路比较两个时期的共享精度，如果精度下降则降低学习速率。类似地，我们使用乱码电路计算精度的差异并测试它是否小于阈值，如果模型收敛则终止。所有这些测试都是在聚合精度上完成的，聚合精度是每个时期的单个值，与训练和测试数据样本的数量无关，因此开销可以忽略不计。请注意，在每个时代，与使用固定学习和孤行的迭代次数相比，我们是否调整学习率或是否终止或不泄漏一个额外的信息因此提供效率（减少的时期数）和安全性之间的 trade-off 速率和。

equality testing. As the size of the testing data is usually significantly smaller than the training data, the time spent on the accuracy testing is only a small portion of the training.

To adjust the learning rate, we compare the shared accuracy of two epochs using a garbled circuit and reduce the learning rate if the accuracy is decreasing. Similarly, we calculate the difference of the accuracy and test if it is smaller than a threshold using a garbled circuit, and terminate if the model converges. All these tests are done on the aggregated accuracy, which is a single value per epoch and independent of the number of the training and testing data samples, thus the overhead is negligible. Notice that in each epoch, whether or not we adjust the learning rate or whether we terminate or not leaks one extra bit of information hence providing a trade-off between the efficiency (reduced number of epochs) and security, compared to using a fixed learning rate and a fixed number of iterations.

正如预期和实验所示,我们的隐私保护机器学习协议的主要瓶颈是离线阶段.它涉及大量的加密操作,如OT或LHE,它们比在线阶段的有限字段中的简单加法和乘法要慢得多.这促使我们探索生成乘法三元组的另一种方法.特别,我们可以让客户生成乘法三元组.由于客户需要首先秘密共享他们的数据,因此进一步要求他们秘密共享一些额外的乘法三元组是很自然的.现在,这些乘法三元组可以以可靠的方式生成,没有繁重的加密操作,从而显着提高了效率.

客户辅助的离线协议

5 Client-Aided Offline Protocol

As expected and shown by the experiments, the main bottleneck in our privacy preserving machine learning protocols is the offline phase. It involves a large number of cryptographic operations such as OT or LHE, which are much slower than simple addition and multiplication in a finite field in the online phase. This motivates us to explore an alternative way of generating multiplication triplets. In particular, we can let the clients generate the multiplication triplets. Since the clients need to secretly share their data in the first place, it is natural to further ask them to secretly share some extra multiplication triplets. Now, these multiplication triplets can be generated in a trusted way with no heavy cryptographic operations, which improves the efficiency significantly. However, despite its benefits, it changes the trust model and introduces some overhead for the online phase.

然而,尽管它有好处,但它改变了信任模型,并为在线阶段引入了一些开销

客户辅助乘法三元组。为简单起见，我们从线性回归开始。注意，在整个训练中，每个数据样本中的每个特征仅在每个时期的两次乘法中使用：一个在前向传播中，另一个在后向传播中。因此，它为持有此值的客户端生成2E乘法三元组提供了支持。特别是，对于每个样本的每个特征，拥有数据的客户端生成随机值u以掩盖数据，并为k = 1，…，E生成随机值vk, v'k并计算zk = u· vk, z'k = u·v'k. 最后，客户端将u, vk, v'k，zk, z'k的份额分配给两个服务器。

Client-Aided Multiplication Triplets. We start with the linear regressions for simplicity. Note that in the whole training, each feature in each data sample is used exactly in two multiplications per epoch: one in the forward propagation and the other in the backward propagation. Therefore, it suffices for the client holding this value to generate $2E$ multiplication triplets. In particular, for each feature of each sample, the client possessing the data generates a random value u to mask the data, and generates random values v_k, v'_k for $k = 1, \ldots, E$ and computes $z_k = u \cdot v_k, z'_k = u \cdot v'_k$. Finally, the client distributes shares of $\langle u \rangle, \langle v_k \rangle, \langle v'_k \rangle, \langle z_k \rangle, \langle z'_k \rangle$ to the two servers.

请注意，我们不假设客户端在生成三元组时知道数据拥有的分区。这意味着我们不能再将矢量化方程用于在线阶段。例如，在4.1节中，在图4的步骤5的前向传播中，我们计算XB×w，我们使用U×V = Z的预先计算的矩阵乘法三元组，其具有与在线阶段完全相同的维度。现在，当客户生成乘法三元组时，小批量XB中的数据可能属于不同的客户，他们可能不知道他们处于相同的小批量训练中，因此无法就常见的随机向量达成一致 V 来计算Z

Notice that we do not assume the clients know the partitioning of the data possession when generating the triplets. This means that we can no longer utilize the vectorized equation for the online phase. For example, in Section 4.1, in the forward propagation at step 5 of Figure 4, where we compute $\mathbf{X}_B \times \mathbf{w}$, we use precomputed matrix multiplication triplets of $\mathbf{U} \times \mathbf{V} = \mathbf{Z}$ with exactly the same dimensions as the online phase. Now, when the multiplication triplets are generated by the clients, the data in the mini-batch $\mathbf{X}_B$ may belong to different clients who may not know they are in the same mini-batch of the training, and thus cannot agree on a common random vector $\mathbf{V}$ to compute $\mathbf{Z}$.

相反，对于XB中的每个数据样本x，双方使用独立生成的乘法三元组计算y* = MulA(x, w), 并且将Y *i设置为y *的向量。因此，增加了计算量，在线阶段的通信和两个服务器的存储

Instead, for each data sample $\mathbf{x}$ in $\mathbf{X}_B$, the two parties compute $\langle y^* \rangle = \mathsf{Mul}^A(\langle \mathbf{x} \rangle, \langle \mathbf{w} \rangle)$ using independently generated multiplication triplets, and set $\langle \mathbf{Y}^* \rangle$ to be a vector of $\langle y^* \rangle$s. Because of this, the computation, communication of the online phase and the storage of the two servers are increased.

The client-aided multiplication triplets generation significantly improves the efficiency of the offline phase, as there is no cryptographic operation involved. However, it introduces overhead to the online phase. The matrix multiplications are replaced by vector inner products. Though the

由于不涉及加密操作，客户辅助乘法三元组生成显着提高了offline阶段的效率。但是，它为在线阶段引入了开销。矩阵乘法由矢量内积替代。尽管执行的乘法的总数完全相同，但是在现代编程语言中使用矩阵库通常更快地使用矩阵乘法算法。这是实验中描述的客户辅助方法引入的主要开销。

沟通也增加了。以前，系数向量被单个随机向量掩盖以计算单个矩阵乘法，而现在它被每个内积的不同随机向量多次掩盖。这些屏蔽值在安全计算协议中在双方之间传输。特别是，与第4节中的协议相比，开销是线性和逻辑回归的t·(2|B|·d-|B|-d). 这在LAN设置中并不重要，但在WAN设置中变得很重要。

total number of multiplications performed is exactly the same, matrix multiplication algorithms are in general faster using matrix libraries in modern programming languages. This is the major overhead introduced by the client-aided approach as depicted in the experiments.

The communication is also increased. Previously, the coefficient vector is masked by a single random vector to compute a single matrix multiplication, while now it is masked multiple times by different random vectors for each inner products. These masked values are transferred between the two parties in the secure computation protocol. In particular, the overhead compared to the protocols in Section 4 is $t \cdot (2|B| \cdot d - |B| - d)$ for linear and logistic regressions. this is not significant in the LAN setting but becomes important in the WAN setting.

最后，存储也增加了. 以前, 矩阵V和Z远小于数据大小,矩阵U与数据大小相同.现在,由于乘法三元组是独立生成的,因此V的大小变为|B|·d·t = n·d·E, 它比数据大小大E.U的大小仍然是 同样，因为每个数据仍被一个随机值掩盖，并且Z的大小仍然相同，因为一旦服务器从所有客户端收集共享，就可以聚合这些值。

Finally, the storage is also increased. Previously, the matrix $\mathbf{V}$ and $\mathbf{Z}$ is much smaller than the data size and the matrix $\mathbf{U}$ is of the same size as the data. Now, as the multiplication triplets are generated independently, the size of $\mathbf{V}$ becomes $|B| \cdot d \cdot t = n \cdot d \cdot E$, which is larger than the size of the data by a factor of E. The size of $\mathbf{U}$ is still the same, as each data is still masked by one random value, and the size of $\mathbf{Z}$ is still the same because the values can be aggregated once the servers collect the shares from all the clients.

Despite all these overheads, the online phase is still very efficient, while the performance of the offline phase is improved dramatically. Therefore, the privacy preserving machine learning with client-aided multiplication triplets generation is likely the most promising option for deployment in existing machine learning frameworks.

尽管存在所有这些开销，但在线阶段仍然非常有效，而离线阶段的性能得到了显着提升。因此，利用客户端辅助乘法三元组生成的隐私保护机器学习可能是在现有机器学习框架中部署的最有希望的选择。

新的安全模型。安全模型也随着客户端辅助阶段的变化而变化。 我们只是非正式地勾勒出这里的差异. 以前，客户端仅负责上传自己的数据，因此当服务器与客户端子集串联时，服务器显然无法学习任何额外信息。现在，由于客户端也在生成乘法三元组，如果客户端的子集与一个服务器串联，它们可能会在迭代中重建系数向量，间接泄漏有关来自诚实客户端的数据的信息。

The new security model. The security model also changes with the client-aided offline phase. We only informally sketch the differences here. Previously, a client is only responsible to upload his own data, and thus the server clearly cannot learn any extra information when he colludes with a subset of clients. Now, as the clients are also generating multiplication triplets, if a subset of clients are colluding with one server, they may reconstruct the coefficient vector in an iteration, which indirectly leaks information about the data from honest clients. Therefore, in the client-aided scenario, we change the security model to not allow collusion between a server and a client. Similar models have appeared in prior work. E.g., in [20], the CSP provides multiplication triplets to the clients to securely compute inner products of their data. If a client is colluding with the CSP, he can immediately learns others' data. Our client-aided protocols are secure under the new model, because the clients learn no extra information after uploading the data and the multiplication triplets. As long as the multiplication triplets are correct, which is the case for semihonest clients we consider, the training is correct and secure.

因此，在客户端辅助方案中，我们将安全模型更改为不允许服务器和客户端之间的串通。在先前的工作中出现了类似的模型。 例如，在[20]中，CSP为客户提供乘法三元组，以安全地计算其数据的内部产品。 如果客户与CSP勾结，他可以立即获知他人的数据。 我们的客户端辅助协议在新模型下是安全的，因为客户端在上传数据和乘法三元组后不会学习额外的信息。 只要乘法三元组是正确的，我们认为半诚实客户就是这种情况，训练是正确和安全的

6 Experimental Results

实验结果

We implement a privacy preserving machine learning system based on our protocols and show the experimental results in this section.

我们基于我们的协议实现隐私保护机器学习系统，并在本节中显示实验结果。

The Implementation. The system is implemented in C++. In all our experiments, the field size is set to 2^{64}. Hence, we observe that the modulo operations can be implemented using regular arithmetics on the unsigned long integer type in C++ with no extra cost. This is significantly faster than any number-theoretic library that is able to handle operations in arbitrary fields. E.g., we tested that an integer addition (multiplication) is $100\times$ faster than a modular addition (multiplication) in the same field implemented in the GMP [5] or the NTL [7] library. More generally, any element in the finite field $\mathbb{Z}_{2^l}$ can be represented by one or several unsigned long integers and an addition (multiplication) can be calculated by one or several regular additions (multiplications) plus some bit

实施。该系统是用C ++实现的。在我们所有的实验中，字段大小设置为264.因此，我们观察到模运算可以使用C ++中无符号长整数类型的常规算术来实现，而无需额外成本。这比任何能够处理任意字段操作的数论库快得多。例如，我们测试了在GMP [5]或NTL [7]库中实现的相同字段中，整数加法（乘法）比模块加法（乘法）快100倍。更一般地，有限字段Z21中的任何元素可以由一个或多个无符号长整数表示，并且可以通过一个或多个常规加法（乘法）加上一些比特运算来计算加法（乘法）。与使用通用数理论库相比，这享有与加速相同的顺序。我们使用特征库[2]来处理矩阵运算。 OT和乱码电路使用EMP工具包[3]实现。它实现了[10]的OT扩展，并对乱码电路应用了免费的XOR [29]和固定密钥AES拼接[11]优化。详情见[44]。我们使用DGK的密码系统[16]用于LHE，这是由Demmler等人提出的[17]。

operations. This enjoys from the same order of speedup compared to using general purpose number theoretic libraries. We use the Eigen library [2] to handle matrix operations. OTs and garbled circuits are implemented using the EMP toolkit [3]. It implements the OT extension of [10], and applies free XOR [29] and fixed-key AES garbling [11] optimizations for garbled circuits. Details can be found in [44]. We use the cryptosystem of DGK [16] for LHE, implemented by Demmler et. al. in [17].

Experimental settings. The experiments are executed on two Amazon EC2 c4.8xlarge machines running Linux, with 60GB of RAM each. For the experiments on a LAN network, we host the two machines in the same region. The average network delay is 0.17ms and the bandwidth is 1GB/s. The setting is quite representative of the LAN setting, as we further tested that two computers connected by a cable have similar network delay and bandwidth. For the experiments on a WAN network, we host the two machines in two different regions, one in the US east and the other in the US west. The average network delay is 72ms and the bandwidth is 9MB/s. We collected 10 runs for each data point in the results and report the average.

实验设置。实验在两台运行Linux的Amazon EC2 c4.8 xlarge机器上执行，每台机器有60GB的RAM。对于LAN网络上的实验，我们在同一区域托管两台机器。平均网络延迟为0.17ms，带宽为1GB / s。该设置非常具有LAN设置的代表性，因为我们进一步测试了通过电缆连接的两台计算机具有相似的网络延迟和带宽。对于WAN网络上的实验，我们在两个不同的区域托管两台机器，一台位于美国东部，另一台位于美国西部。平均网络延迟为72毫秒，带宽为9MB /秒。我们为结果中的每个数据点收集了10次运行并报告了平均值。

Our experiments in the LAN setting capture the scenario where the two servers in our protocols have a high-bandwidth/low-latency network connection, but otherwise are not administered/controlled by the same party. The primary reason for reporting experiments in the LAN setting is more accurate benchmarking and comparison as the majority of prior work, including all previous MPC implementations for machine learning only report results in the LAN setting. Moreover, contrasting our results in the LAN and WAN setting highlights the significance of network bandwidth in our various protocols. For example, as our experiments show, the total time for the offline phase in the LAN and WAN setting are very close when using LHE techniques to generate multiplication triplets while there is a signficant gap between the two when using OT extension (see Table 2).

我们在LAN设置中的实验捕获了我们的协议中的两个服务器具有高带宽/低延迟网络连接但不由同一方管理/控制的情况。在LAN设置中报告实验的主要原因是更准确的基准测试和比较，因为大多数先前的工作,包括所有以前用于机器学习的MPC实现仅报告LAN设置中的结果。此外，对比我们在LAN和WAN设置中的结果，突出了我们各种协议中网络带宽的重要性。例如，正如我们的实验所示，当使用LHE技术生成乘法三元组时，LAN和WAN设置中的离线阶段的总时间非常接近，而使用OT扩展时两者之间存在显着的差距（参见表2）。

Furthermore, while the LAN setting is understandably not always a realistic assumption, there are scenarios where a high bandwidth link (or even a direct dedicated link) between the two servers is plausible. For example, in payment networks, it is not uncommon for the various involved parties (issuing Banks, aquiring Banks, large merchants, and payment networks) to communicate over fast dedicated links connecting them. Similarly, in any international organization that needs to abide by different privacy regulations and data sovereignty restrictions, the two servers may indeed be connected using a high bandwidth direct link but be administered in different countries. In such scenarios, the logical, administrative, or legal separation of the two servers plays a more significant role.

此外，虽然LAN设置可以理解并不总是一个现实的假设，但有些情况下两个服务器之间的高带宽链路(甚至直接专用链路)似乎是合理的。例如，在支付网络中，各种相关方(发行银行，查询银行，大型商家和支付网络)通过连接它们的快速专用链路进行通信的情况并不少见,同样,在任何需要遵守不同隐私法规和数据主权限制的国际组织中，这两台服务器确实可以使用高带宽直接链路进行连接，但可以在不同的国家进行管理。在这种情况下，两个服务器的逻辑，管理或法律分离起着更重要的作用。

Offline vs. Online. We report experimental numbers for both the offline and the online phase of our protocols separately, but only use total costs (online + offline) when comparing to related work. The offline phase includes all computation and communication that can be performed without presence of data, while the online phase consists of all data-dependent steps of the protocol. Optimizing the online cost is useful for application scenarios where a fast turn-around is required. In particular, when using our protocols for privacy-preserving prediction (e.g. fraud detection), new data needs to be classified with low latency and high throughput. Indeed, we run a set of experiments to demonstrate that online cost of privacy-preserving prediction can be made fast enough to run for latency critical applications (See Table 5). Similarly, when training small models dynamically and on a regular basis, it is important to have high online efficiency. In contrast, when training large models (e.g. a large neural networks), the separation of the offline and the online costs is less important.

Offline vs. Online。我们分别报告了我们协议的offline和在线阶段的实验数字，但在与相关工作进行比较时仅使用总成本(niline + offline)。offline阶段包括可以在没有数据存在的情况下执行的所有计算和通信，而online阶段包括协议的所有数据相关步骤。优化online成本对于需要快速周转的应用场景非常有用。特别是，当使用我们的协议进行隐私保护预测（例如欺诈检测）时，需要以低延迟和高吞吐量对新数据进行分类。实际上，我们运行了一系列实验来证明保护隐私的在线成本可以足够快地运行以用于延迟关键应用程序（参见表5）。同样，在动态定期培训小型模型时，具有较高的在线效率非常重要。相反，当训练大型模型（例如大型神经网络）时，offline和online成本就不那么重要了。

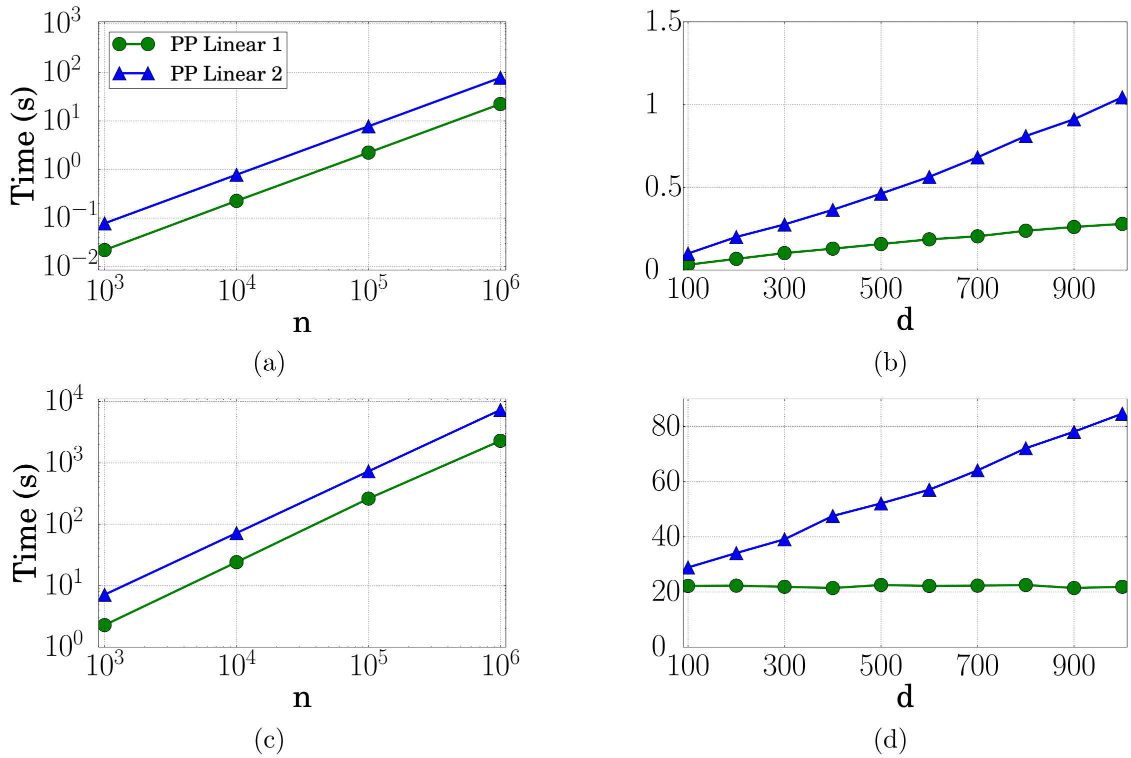


Figure 9: Online cost of privacy preserving linear regression in standard and client-aided settings. $|B|$ is set to 128. Figure (a), (b) are for LAN network and Figure (c), (d) are for WAN network. Figure (a) and (c) are in log-log scale and for $d = 784$. Figure (b) and (d) are in regular scale and for $n = 10,000$.

图9：在标准和客户端辅助设置中隐私保护线性回归的online成本。|B|设置为128 。
图(a)，(b)在LAN网络，图(c)，(d)在WAN网络。图(a)和(c)是d=784的log-log规模。图(b)和(d)是n = 10,000常规规模。

Data sets. In our experiments, we use the following datasets. The MNIST dataset [6] contains images of handwritten digits from "0" to "9". It has 60,000 training samples, each with 784 features representing 28 × 28 pixels in the image. Each feature is a grayscale between 0∼255. The Gisette dataset [4, 24] contains images of digits "4" and "9". It has 13,500 samples and 5,000 features between 0∼1,000. We also use the Arcene dataset [1, 24]. It contains mass-spectrometric data and is used to determine if the patient has cancer. There are 200 data samples with 10,000 features. Each value is between 0 and 1000. Besides these real-world datasets, we also use synthetic datasets to test the scalability of our protocols to larger sizes (e.g. a million samples).

数据集。 在我们的实验中，我们使用以下数据集。 MNIST数据集[6]包含从 “0” 到 “9” 的手写数字的图像。 它有60,000个训练样本，每个样本有784个特征，代表图像中的28×28像素。 每个功能都是0到255之间的灰度。 Gisette数据集[4,24]包含数字“4”和“9”的图像。 它有13,500个样本和5,000个功能，介于0到1,000之间。 我们还使用Arcene数据集[1,24]。 它包含质谱数据，用于确定患者是否患有癌症。 有200个数据样本，10,000个特征。 每个值都在0到1000之间。除了这些真实世界的数据集之外，我们还使用合成数据集来测试我们的协议对更大尺寸（例如一百万个样本）的可扩展性。

线性回归的实验

6.1 Experiments for Linear Regression

我们从不同设置中的隐私保护线性回归协议的实验结果开始，并将其与以前的隐私保护解决方案进行比较。

We start with the experimental results for our privacy preserving linear regression protocols in different settings, and compare it with previous privacy preserving solutions.

Online phase. To examine how the the online phase scales, we run experiments on datasets with size (n) from 1,000 to 1,000,000 and d from 100 to 1,000. When $n \leq 60000$ and $d \leq 784$, the samples are directly drawn from the MNIST dataset. When n and d are larger than that of MNSIT, we duplicate the dataset and add dummy values for missing features. Note that when n, d, E are fixed, the actual data used in the training does not affect the running time.

Figure 9a shows the results in the LAN setting. "PP Linear 1" denotes the online phase of our

在线阶段。 为了研究在线阶段如何扩展，我们对数据集进行了实验，这些数据集的大小(n)为1,000到1,000,000，d为100到1,000。 当n≤60000且d≤784时，样本直接从MNIST数据集中提取。 当n和d大于MNSIT时，我们复制数据集并为缺失的要素添加虚拟值。 请注意，当固定n，d，E时，训练中使用的实际数据不会影响运行时间。

图9a显示了LAN设置中的结果。 “PP线性1”表示我们的隐私保护线性回归的在线阶段，矩阵形式的乘法三元组，“PP线性2”表示客户辅助变体的在线阶段。 报告的运行时间是两台服务器同时运行并相互交互的总在线时间。 根据我们的实验，双方大致相同的时间。 学习率是预先确定的，我们不计算在图中找到合适的学习率的时间。 特征数量固定为784，n从1,000到1,000,000不等。

如图所示，我们的线性回归的在线时间在LAN设置中非常快。 特别是，在100万个数据样本上安全地训练线性模型只需22.3秒，每个样本有784个特征。 从隐私保护训练所需的22.3s开始，只有一小部分，即少于2s，用于交互的网络延迟。 考虑到LAN网络的高带宽，传输数据的通信时间可以忽略不计。 我们使用客户端生成的乘法三元组的第二个协议的开销大约为3.5倍。 特别是，训练模型需要77.6s，n = 1,000,000，d = 784.如图9a和9b所示，我们协议的运行时间与n和d呈线性关系。 我们还观察到，我们测试的所有数据集的线性和逻辑回归的SGD总是在第一个时期内收敛，并在第二个时期之后终止，这证明了SGD在实践中非常有效和高效。

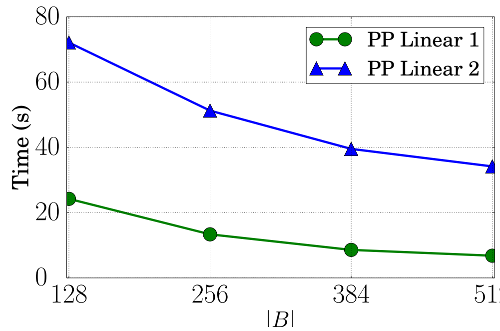


Figure 10: Performance of online phase of linear regression on WAN with different mini-batch sizes. $n = 10,000$, $d = 784$.

图10：具有不同小批量大小的WAN上线性回归的在线阶段的性能。 n = 10,000，d = 784。

privacy preserving linear regression with multiplication triplets in matrix form, and “PP Linear 2” denotes the online phase of the client-aided variant. The running time reported is the total online time when two servers are running simultaneously and interacting with each other. The two parties take roughly the same time based on our experiments. The learning rate is predetermined and we do not count the time to find an appropriate learning rate in the figures. The number of features is fixed to 784 and n varies from 1,000 to 1,000,000.

As shown in the figure, the online time of our linear regression is very fast in the LAN setting. In particular, it only takes 22.3s to train a linear model securely on 1 million data samples with 784 features each. From 22.3s needed for privacy preserving training, only a small portion, namely less than 2s, is spent on the network delay for the interactions. The communication time to transfer the data is negligible given the high bandwidth of the LAN network. Our second protocol using client-generated multiplication triplets has an overhead of roughly 3.5×. In particular, it takes 77.6s to train the model with $n = 1,000,000$ and $d = 784$. As shown in Figure 9a and 9b, the running time of our protocol scales linearly with both n and d. We also observe that the SGD for linear and logistic regressions on all the datasets we tested always converges within the first epoch, and terminate after the second epoch, which confirms that the SGD is very effective and efficient in practice.

Figure 9c shows the corresponding performance on a WAN network. The running time of our privacy preserving protocols increase significantly. In particular, our first protocol takes 2291.8s to train the model when $n = 1,000,000$ and $d = 784$. The reason is that now the network delay is the dominating factor in the training time. The computation time is exactly the same as the LAN setting, which is around 20s; the communication time is still negligible even under the bandwidth of the WAN network. The total running time is almost the same as the network delay times the number of iterations. Our second protocol is still roughly 3.3× slower than the first protocol, but the reason is different from the LAN setting. In the WAN setting, this overhead comes from the increment of the communication, as explained in Section 5. Even under this big network delay in the WAN network, as we will show later, the performance of our privacy preserving machine learning is still orders of magnitude faster than the state of the art. Besides, it is also shown in Figure 9c that the training time grows linearly with the number of the samples in WAN networks. However, in Figure 9d, when fixing $n = 10,000$, the training time of our first protocol only grows slightly when d increases, which again has to do with the fact that number of interactions is independent of d. The overhead of our second protocol compared to the first one is increasing with d, because

图9c显示了WAN网络上的相应性能。我们的隐私保护协议的运行时间显着增加。特别是，当n = 1,000,000和d = 784时，我们的第一个协议需要2291.8秒训练模型。原因是现在网络延迟是训练时间的主导因素。计算时间与局域网设置完全相同，大约为20秒;即使在WAN网络的带宽下，通信时间仍然可以忽略不计。总运行时间与网络延迟乘以迭代次数几乎相同。我们的第二个协议仍然比第一个协议慢大约3.3倍，但原因与LAN设置不同。在WAN设置中，这种开销来自通信的增量，如第5节所述。即使在WAN网络中的这种大网络延迟下，正如我们稍后将要说明的那样，我们的隐私保护机器学习的性能仍然是比现有技术更快。此外，图9c中还示出了训练时间随着WAN网络中的样本数量线性增长。然而，在图9d中，当确定n = 10,000时，我们的第一个协议的训练时间仅在d增加时略微增长，这又与交互数量与d无关的事实有关。与第一个协议相比，我们的第二个协议的开销随着d的增加而增加，因为通信在第二个协议中与d线性增长。当d = 100时，训练时间几乎相同，因为它受到交互的支配;当d = 1000时，由于通信开销，训练时间慢4倍。

我们还表明，我们可以通过增加小批量大小来提高WAN设置的性能，以平衡计算时间和网络延迟。图10显示了此参数调整的结果。我们让n = 10,000和d = 784并且增加| B |衡量其对绩效的影响。如图所示，当我们增加小批量时，在线阶段的运行时间正在减少。特别是，当| B |时，在我们的第一个协议中训练模型需要6.8秒= 512，这几乎是| B |所需时间的4倍这是因为当时期数相同时，迭代次数（或相互作用）与小批量大小成反比。当小批量大小增加时，计算时间基本保持不变，而在交互上花费的时间减少。但是，运行时间不能总是在减少。当计算时间占主导地位时，运行时间将保持不变。此外，如果| B |如果设置得太大，则迭代次数在一个时期中太小，使得模型可能不会像以前那样快地达到最佳值，这可能导致必要时期E的数量增加，其本身可以影响性能。通常考虑到明文训练中模型的矢量化，并行化和稳健性的加速来确定小批量大小。在隐私保护设置中，我们建议还应考虑网络条件并找到合适的小批量大小以优化训练时间。

		LHE-based			OT-based			Client aided		Dataset
n	d	LAN	WAN	Comm.	LAN	WAN	Comm.	Time	Comm.	size
1,000	100	23.9s	24.0s	2MB	0.86s	43.2s	190MB	0.028s	7MB	0.8MB
	500	83.9s	84.8s	6MB	3.8s	210.6s	1GB	0.16s	35MB	3.8MB
	1000	158.4s	163.2s	10MB	7.9s	163.2s	1.9GB	0.33s	69MB	7.6MB
10,000	100	248.4s	252.9s	20MB	7.9s	420.2s	1.9GB	0.33s	69MB	7.6MB
	500	869.1s	890.2s	60MB	39.2s	2119.1s	9.5GB	1.98s	344MB	38MB
	1000	1600.9s	1627.0s	100MB	80.0s	4097.1s	19GB	4.0s	687MB	76MB
100,000	100	2437.1s	2478.1s	200MB	88.0s	4125.1s	19GB	3.9s	687MB	76MB
	500	8721.5s	8782.4s	600MB	377.9s	20000s*	95GB	20.2s	3435MB	380MB
	1000	16000s*	16100s*	1000MB	794.0s	40000s*	190GB	49.9s	6870MB	760MB

Table 2: Performance of the offline phase. $|B| = 128$ and $E = 2$. (* means estimated via extrapolation.)

表2：o ffl ine阶段的表现。| B | = 128且E = 2.（*表示通过外推估计。）

the communication grows linearly with d in the second protocol. When $d = 100$, the training time is almost the same, as it is dominated by the interaction; when $d = 1000$, the training time is $4\times$ slower because of the overhead of communication.

We also show that we can improve the performance in the WAN setting by increasing the mini-batch size, in order to balance the computation time and the network delay. Figure 10 shows the result of this parameter tweaking. We let $n = 10,000$ and $d = 784$ and increases $|B|$ to measure its effect on performance. As shown in the figure, the running time of the online phase is decreasing when we increase the mini-batch size. In particular, it takes 6.8s to train the model in our first protocol when $|B| = 512$, which is almost 4 times faster than the time needed when $|B| = 128$. This is because when the number of epochs is the same, the number of iterations (or interactions) is inverse proportional to the mini-batch size. When the mini-batch size is increasing, the computation time remains roughly unchanged, while the time spent on interaction decreases. However, the running time cannot always keep decreasing. When the computation time becomes dominating, the running time will remain unchanged. Furthermore, if $|B|$ is set too large, the number of iterations is too small in an epoch such that the model may not reach the optimum as fast as before, which may result in an increase in the number of necessary epochs E which itself can affect the performance. Mini-batch size is usually determined considering the speed up of vectorization, parallelization and robustness of the model in plaintext training. In the privacy preserving setting, we suggest that one should also take the network condition into consideration and find an appropriate mini-batch size to optimize the training time.

离线阶段。离线阶段的性能总结在表2中。我们报告了LAN和WAN网络上的运行时间以及基于OT和基于LHE的乘法三元组生成的总体通信。 对于客户端辅助设置，我们通过在一台机器上生成所有三元组来模拟总计算时间。 我们报告其总时间和总通信，但不区分LAN和WAN设置，因为实际上数据将从具有不同网络条件的多个客户端发送。作为参考，我们还包括数据集大小，假设每个值都存储为64位十进制数。我们将n从1000变为100,000，将d从100变为1000.迷你批量大小设置为128，时期数设置为2，因为我们通常在线阶段只需要2个时期。 如果需要更多的历元，表中报告的所有结果都明显地随着历元的数量线性增长

Offline phase. The performance of the offline phase is summarized in Table 2. We report the running time on LAN and WAN networks and the total communication for OT-based and LHE-based multiplication triplets generation. For the client-aided setting, we simulate the total computation time by generating all the triplets on a single machine. We report its total time and total communication, but do not differentiate between the LAN and WAN settings, since in practice the data would be sent from multiple clients with different network conditions. As a point of reference, we also include the dataset size assuming each value is stored as 64-bit decimal number. We vary n from 1000 to 100,000 and d from 100 to 1000. The mini-batch size is set to 128 and the number of epochs is set to 2, as we usually only need 2 epochs in the online phase. If more epochs are needed, all the results reported in the table clearly grow linearly with the number of epochs.

As shown in the table, the LHE-based multiplication triplets generation is the slowest among

如表中所示，基于LHE的乘法三元组生成是所有方法中最慢的。 特别是，对于n = 10,000和d = 1000，需要1600.9s。原因是LHE中的每个基本操作，即加密和解密都非常慢，这使得该方法不切实际。 例如，一次加密需要3ms，比一次OT（使用OT扩展时）慢大约10,000×。 然而，基于LHE的方法产生最佳通信。 如4.2节所述，渐近复杂度远小于数据集大小。 考虑大的密文（2048位），整体通信仍然与数据集大小的顺序相同。 在LAN和WAN网络上运行时，此通信几乎不会产生任何开销。 与在线阶段不同，o ffl ine阶段仅需要1次交互，因此网络延迟可忽略不计。

	MNIST	Gisette	Arcene
Cholesky	92.02%	96.7%	87%
SGD	91.95%	96.5%	86%

Table 3: Comparison of accuracy for SGD and Cholesky

表3：SGD和Cholesky准确度的比较

在局域网设置中，基于OT的乘法三元组生成的性能要好得多。 特别是，对于n = 10,000和d = 1000，它只需要80.0s。它在通信上引入了巨大的开销，即19GB，而数据仅为76MB。 这种通信开销使WAN网络上的运行时间慢得多。 由于这种通信开销（这是OT的主要成本），总运行时间甚至比基于LHE的WAN网络上的生成要慢。

all approaches. In particular, it takes 1600.9s for $n = 10,000$ and $d = 1000$. The reason is that each basic operation in LHE, i.e., encryption, and decryption are very slow, which makes the approach impractical. E.g., one encryption takes 3ms, which is around $10,000\times$ slower than one OT (when using OT extension). However, the LHE-based approach yields the best communication. As calculated in Section 4.2, the asymptotic complexity is much smaller than the dataset size. Taking the large ciphertext (2048 bits) into consideration, the overall communication is still on the same order as the dataset size. This communication introduces almost no overhead when running on both LAN and WAN networks. Unlike the online phase, the offline phase only requires 1 interaction and hence the network delay is negligible.

The performance of the OT-based multiplication triplets generation is much better in the LAN setting. In particular, it only takes 80.0s for $n = 10,000$ and $d = 1000$. It introduces a huge overhead on the communication, namely 19GB while the data is only 76MB. This communication overhead makes the running time much slower on WAN networks. Because of this communication overhead, which is the major cost of OT, the total running time is even slower than the LHE-based generation on WAN networks.

最后，客户端辅助乘法三元组生成是最快的，因为不涉及加密操作。 对于n=10,000和d=1000，它只需要4.0s。总通信的开销仅为数据集大小的9倍，这在实践中是可接受的。

Finally, the client-aided multiplication triplets generation is the fastest because no cryptographic operation is involved. It only takes 4.0s for $n = 10,000$ and $d = 1000$. The overhead on the total communication is only around 9 times the dataset size which is acceptable in practice.

表2还显示，所有运行时间均随n和d大致线性增长7，这与4.2节中得到的渐近复杂度一致。

It is also shown in Table 2 that all the running times grow roughly linearly[7] with both n and d, which agrees with the asymptotic complexity derived in Section 4.2.

结合在线和现阶段的结果，我们的系统仍然非常有效。 例如，在局域网设置中，当使用客户端辅助乘法三元组时，在线阶段我们的隐私保护线性回归只需1.0秒，其中n=10,000 ,d = 1000.只有阶段的总时间才是 4.0s，将在实践中进一步分发给多个客户。 当使用基于OT的生成时，在线阶段需要0.28秒，而o ffl ine阶段需要80.0秒。

Combining the results presented for both the online and the offline phase, our system is still quite efficient. E.g., in the LAN setting, when client-aided multiplication triplets are used, it only takes 1.0s for our privacy preserving linear regression in the online phase, with $n = 10,000$ and $d = 1000$. The total time for the offline phase is only 4.0s, which would be further distributed to multiple clients in practice. When OT-based generation is used, the online phase takes 0.28s and the offline phase takes 80.0s.

与以前的工作比较。 正如1.2节所述，在类似的双服务器设置中，[36] (NWI + 13)和[20] (GSB + 16)也考虑了隐私保护线性回归。 这两篇论文不是使用SGD方法，而是建议通过求解我们在2.1节中描述的线性系统来计算最优值。 我们表明，通过SGD方法训练的模型可以在表3中，MNIST, Gisette和Arcene数据集上达到相同的精度

Comparison with prior work. As surveyed in Section 1.2, privacy preserving linear regression was also considered by [36] (NWI^+13) and [20] (GSB^+16) in a similar two-server setting. Instead of using the SGD method, these two papers propose to calculate the optimum by solving a linear system we described in Section 2.1. We show that the model trained by the SGD method can reach the same accuracy in Table 3, on the MNIST, Gisette and Arcene datasets.

The protocols in NWI^+13 and GSB^+16 can be decomposed into two steps. In the first step, the $d \times d$ matrix $\mathbf{X}^T \times \mathbf{X}$ is constructed securely, which defines a linear system. In the second step, the Cholesky algorithm or its variants are implemented using a garbled circuit. In the first step of NWI^+13, each client encrypts a $d \times d$ matrix using LHE. In GSB^+16, the first step is computed using multiplication triplets generated by the CSP, which is faster than NWI^+13. However, now the clients cannot collude with the CSP, which is similar to the model we consider in the client-aided setting.

[7] The number of encryptions and decryptions in the LHE-based generation is $O(|B| + d)$. As $|B|$ is fixed to 128, its running time does not grow strictly linearly with d, as reflected in Table 2.

NWI + 13和GSB + 16中的协议可以分解为两个步骤。 在第一步中，d×d矩阵XT×X被安全地构造，其定义了线性系统。 在第二步中，使用乱码电路实现Cholesky算法或其变体。 在NWI + 13的第一步中，每个客户端使用LHE加密d×d矩阵。 在GSB + 16中，第一步是使用CSP生成的乘法三元组计算的，它比NWI + 13快。 但是，现在客户端无法与CSP串通，这与我们在客户端辅助设置中考虑的模型类似。

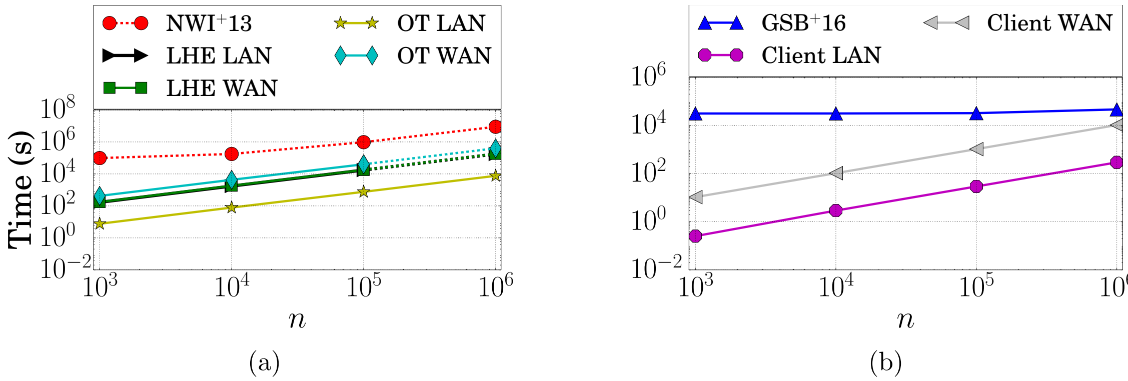


Figure 11: Efficiency comparison with prior work. Figures are in log-log scale, $d = 500$, $|B| = 128$ for our schemes.

图11：与先前工作的效率比较。 数字以对数对数标度表示，d = 500，| B | 我们的计划是128。

使用乱码电路，NWI + 13实现Cholesky算法，而GSB + 16实现CGD，一种近似算法。

Using garbled circuits, NWI$^+$13 implements the Cholesky algorithm while GSB$^+$16 implements CGD, an approximation algorithm.

For comparison, we use the numbers reported in [20, Table 1, Figure 6]. As the performance of the first step of NWI$^+$13 is not reported in the table, we implement it using Paillier's encryption [37] with batching, which is the same as used in the protocol of NWI$^+$13. For the first step in GSB$^+$16, we use the result of the total time for two clients only in [20, Table 1] with $d = 500$, which is the fastest[8]; for the second step in GSB$^+$16, we use the result for CGD with 15 iterations in [20, Figure 6] with $d = 500$. We sum up the running time of our offline and online phase, and sum up the running time of the first and the second step in NWI$^+$13 and GSB$^+$16, and report the total running time of all parties in all the schemes.

为了比较，我们使用[20，表1，图6]中报告的数字。 由于表中没有报告NWI + 13的第一步的表现，我们使用Paillier的加密[37]来实现它与批处理，这与NWI + 13的协议中使用的相同。 对于GSB + 16的第一步，我们仅在[20，表1]中使用两个客户端的总时间结果，其中d = 500，这是最快的8; 对于GSB + 16的第二步，我们使用[20，图6]中的15次迭代的CGD结果，其中d = 500.我们总结了我们的o ffl ine和在线阶段的运行时间，并总结了运行时间。 NWI + 13和GSB + 16的第一步和第二步，并报告所有方案中各方的总运行时间。

In Figure 11a, we compare the performance of the scheme in NWI$^+$13 and our schemes with OT-based and LHE-based multiplication triplets generation, executed in both LAN and WAN settings. As shown in the figure, the performance is improved significantly. For example, when $n = 100,000$ and $d = 500$, even our LHE-based protocol in both LAN and WAN settings has a $54\times$ speedup. The OT-based protocol is $1270\times$ faster in the LAN setting and $24\times$ faster in the WAN setting. We could not execute the first step of NWI$^+$13 for $n \geq 10,000$ and the dotted line in the figure is our extrapolation [9].

在图11a中，我们比较了NWI + 13中的方案的性能以及我们的方案与基于OT和基于LHE的乘法三元组生成，在LAN和WAN设置中执行。 如图所示，性能显着提高。 例如，当n = 100,000且d = 500时，即使我们在LAN和WAN设置中基于LHE的协议也具有54倍的加速。基于OT的协议在LAN设置中快1270倍，在WAN设置中快24倍。 对于n≥10,000，我们无法执行NWI + 13的第一步，图中的虚线是我们的推断

We further compare the performance of the scheme in GSB$^+$16 and our scheme with client-generated multiplication triplets in Figure 11b, as they are both secure under the assumption that servers and clients do not collude. As shown in the figure, when $n = 100,000$ and $d = 500$, our scheme has a $31\times$ speedup in WAN setting and a $1110\times$ speedup in LAN setting. As the figure is in log-log scale, the larger slope of the growth of the running time for our schemes does not mean we will be slower eventually with large enough n. It means that the relative speedup is decreasing, but, in fact, the absolute difference between the running time of our scheme and GSB$^+$16 keeps increasing.

[8] For $n = 1,000,000, d = 500$, since the data point is missing in [20, Table 1], we extrapolate assuming a quadratic complexity in d.

[9] The running time of our scheme using OT-based offline in the WAN setting for $n = 100,000$ and $n = 1,000,000$, using LHE-based offline in LAN and WAN for $n = 1,000,000$ are also estimated (dotted in the figure). The running time using OT-based offline in LAN for $n = 1,000,000$ is from real execution, though the number was not reported in Table 2 due to page limit. Similarly, we were also able to run the client-aided offline for $n = 1,000,000$.

我们进一步比较了图11b中GSB + 16和我们的方案与客户生成的乘法三元组的方案的性能，因为它们在服务器和客户端不串通的假设下都是安全的。 如图所示，当n = 100,000且d = 500时，我们的方案在WAN设置中具有31倍的加速和在LAN设置中具有1110倍的加速。 由于图像是对数对数尺度，我们的方案运行时间增长的较大斜率并不意味着我们最终会因为足够大的n而变慢。 这意味着相对加速正在减少，但实际上，我们的方案运行时间与GSB + 16之间的绝对差异在不断增加。

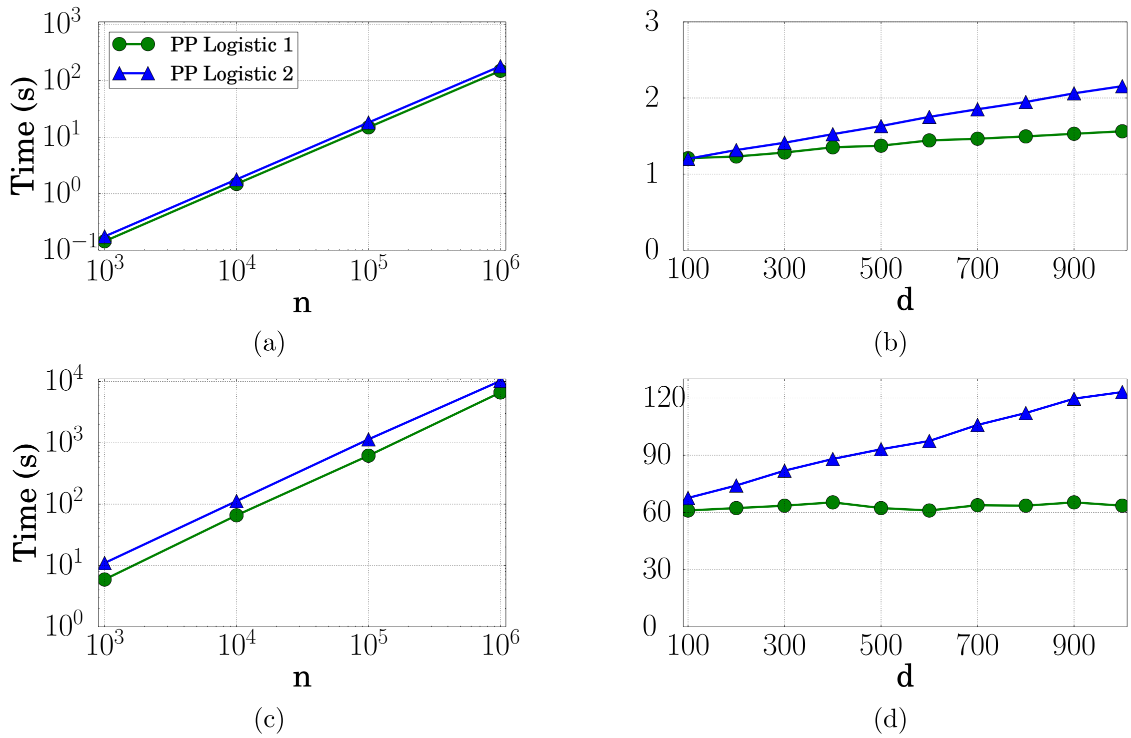


Figure 12: Online cost of privacy preserving logistic regression in the standard and client-aided setting. $|B|$ is set to 128. Figure (a), (b) are for LAN network and Figure (c), (d) are for WAN network. Figure (a) and (c) are in log-log scale, $d = 784$. Figure (b) and (d) are in regular scale, $n = 10,000$.

图12：在标准和客户端辅助设置中保护隐私的在线成本。| B | 设置为128.图（a），（b）用于LAN网络，图（c），（d）用于WAN网络。图（a）和（c）是对数对数标度，d = 784.图（b）和（d）是常规标度，n = 10,000。

The reason why the cost of NWI$^+$13 and GSB$^+$16 are so high when n is small is that the size of the garbled circuit to solve the linear system only depends on d. Even if there is only 1 data sample, the time of the second step for $d = 500$ is around 90,000s in NWI$^+$13 and 30,000s in GSB$^+$16.

当n很小时，NWI + 13和GSB + 16的成本如此之高的原因是解决线性系统的乱码电路的大小仅取决于d。即使只有1个数据样本，d = 500的第二步时间在NWI + 13中约为90,000s，在GSB + 16中约为30,000s

Note that the gap between our scheme and prior work will become even larger as d increases, as the running time is linear in d in our schemes and quadratic or cubic in the two prior schemes. In addition, all the numbers reported for the two prior work were obtained on a network with 1 Gbps bandwidth [20] which is close to our LAN setting. Indeed, the garbled circuit introduces a huge communication and storage overhead. As reported in [20, Figure 4], the garbled circuits for $d = 500$ in both schemes have more than 10^{11} gates, which is 3000GB. The communication time to transfer such a circuit would be at least 330000s on a WAN network, which implies the speedup for our scheme could be more significant in the WAN setting.

请注意，随着d的增加，我们的方案与先前工作之间的差距将变得更大，因为在我们的方案中d的运行时间是线性的，而在两个先前的方案中是二次或三次的。此外，报告的两个先前工作的所有数字都是在带宽为1 Gbps的网络上获得的[20]，这与我们的LAN设置很接近。实际上，乱码电路引入了巨大的通信和存储开销。如[20，图4]所述，两种方案中d = 500的乱码电路都有超过1011个门，即3000GB。在WAN网络上传输此类电路的通信时间至少为330000，这意味着我们的方案在WAN设置中的加速可能更为显着。

Finally, NWI$^+$13 only supports horizontally partitioned data, where each client holds one or multiple rows of the data matrix; GSB$^+$16 only supports vertically partitioned data with $2 \sim 5$ clients, where each client holds one entire column of the data. Our schemes can support arbitrary partitioning of the data. Besides, the offline phase of our protocols is data independent. The servers and the clients can start the offline phase with basic knowledge on the bounds of the dataset size, while the bulk of the computation in the two prior work need to be performed after obtaining the data.

最后，NWI + 13仅支持水平分区数据，其中每个客户端保存一行或多行数据矩阵; GSB + 16仅支持具有2~5个客户端的垂直分区数据，其中每个客户端保存一整列数据。我们的方案可以支持数据的任意分区。此外，我们协议的o ffl ine阶段与数据无关。服务器和客户端可以使用关于数据集大小边界的基本知识启动o ffl ine阶段，而两个先前工作中的大部分计算需要在获得数据之后执行。

	Protocol 1		Protocol 2	
	Offline	Online	Offline	Online
RELU	290,000s*	4239.7s	14951.2s	10332.3s
Square	320,000s*	653.0s	16783.9s	4260.3s

Table 4: Performance of privacy preserving neural networks training on MNIST in LAN setting. $n = 60,000$, $d = 784$.

表4：在LAN设置中对MNIST进行隐私保护神经网络训练的性能。 n = 60,000， d = 784。

逻辑回归实验

6.2 Experiments for Logistic Regression

在本节中，我们将审查隐私保护逻辑回归协议的实验结果。由于该协议不需要任何额外的乘法三元组，因此离线阶段具有与线性回归完全相同的成本。

In this section, we review experimental results for our privacy preserving logistic regression protocol. Since this protocol does not require any additional multiplication triplets, the offline phase has the exact same cost as linear regression.

As shown in Figure 12, our privacy preserving logistic regression introduces some overhead on top of the linear regression. Specifically, in Figure 12a, when $n = 1,000,000$ and $d = 784$, our protocol 1 using OT-based or LHE-based multiplication triplets takes 149.7s in the online phase. This overhead is introduced purely by the extra garbled circuit to compute our logistic function. The fact that a small additional garbled circuit introduces a 7× overhead, serves as evidence that the running time would be much larger if the whole training was implemented in garbled circuits. Our protocol 2, using client-generated multiplication triplets, takes 180.7s as no extra multiplication triplet is used in logistic regression and the garbled circuit is an additive overhead, no matter which type of multiplication triplet is used. The training time grows linearly with both n and d, as presented in Figure 12a and 12b.

如图12所示，我们的隐私保护逻辑回归在线性回归之上引入了一些开销。具体来说，在图12a中，当n = 1,000,000且d = 784时，我们使用基于OT或基于LHE的乘法三元组的协议1在在线阶段需要149.7秒。这个开销纯粹由额外的乱码电路引入，以计算我们的逻辑功能。一个小的附加乱码电路引入7倍开销的事实证明，如果整个训练在乱码电路中实现，则运行时间会大得多。我们的协议2使用客户端生成的乘法三元组，需要180.7秒，因为在逻辑回归中没有使用额外的乘法三元组，并且乱码电路是一个加性开销，无论使用哪种类型的乘法三元组。训练时间随n和d线性增长，如图12a和12b所示。

Figure 12c and 12d shows the result on a WAN network. The time spent on the interactions is still the dominating factor. When $n = 1,000,000$ and $d = 784$, it takes around 6623s for our first protocol, and 10213s for the second. Compared to privacy preserving linear regression, one extra interaction and extra communication for the garbled circuit is added per iteration. We can also increase the mini-batch size $|B|$ to balance the computation and interactions and improve the performance. We omit the result due to page limit.

图12c和12d显示了WAN网络上的结果。花在互动上的时间仍然是主导因素。当n = 1,000,000且d = 784时，我们的第一个协议需要大约6623秒，而第二个协议需要10213秒。与隐私保护线性回归相比，每次迭代添加一个额外的交互和用于乱码电路的额外通信。我们还可以增加小批量| B |平衡计算和交互并提高性能。由于页面限制，我们省略了结果。

To further show the scalability of our system, we run the online part of our privacy preserving logistic regression on the Gisette dataset with 5000 features and up to 1,000,000 samples on a LAN network. It takes 268.9s using our first protocol and 623.5s using the second one. The trained model can reach an accuracy of 97.9% on the testing dataset.

为了进一步显示我们系统的可扩展性，我们在Gisette数据集上运行隐私保护逻辑回归的在线部分，具有5000个功能，在LAN网络上最多1,000,000个样本。使用我们的第一个协议需要268.9秒，使用第二个协议需要623.5秒。经过训练的模型可以在测试数据集上达到97.9%的准确度。

We are not aware of any prior work in this security model with an implementation. We are the first to implement a scalable system for privacy preserving logistic regression.

我们不了解此安全模型中的任何先前工作与实现。我们是第一个实现隐私保护逻辑回归的可扩展系统。

6.3 Experiments for Neural Networks

神经网络的实验

We also implemented our privacy preserving protocol for training an example neural network on the MNIST dataset. The neural network has two hidden layers with 128 neurons in each layer. We experiment with both the RELU and the square function as the activation function in the hidden layers and our proposed alternative to softmax function in the output layer. The neural network is fully connected and the cost function is the cross entropy function. The labels are represented as *hot vectors* with 10 elements, where the element indexed by the digit is set to 1 while others are 0s. We run our system on a LAN network and the performance is summarized in Table 4. $|B|$ is set to 128 and the training converges after 15 epochs.

我们还实施了隐私保护协议，用于在MNIST数据集上训练示例神经网络。神经网络有两个隐藏层，每层有128个神经元。我们尝试将RELU和square函数作为隐藏层中的激活函数，并且我们提出了输出层中softmax函数的替代方案。神经网络完全连接，成本函数是交叉熵函数。标签表示为具有10个元素的热向量，其中由数字索引的元素设置为1，而其他元素为0。我们在LAN网络上运行我们的系统，性能总结在表4中。| B | 设置为128，训练在15个时期后收敛。

如表所示，当使用RELU功能时，我们的第一个协议的在线阶段需要4239.7s，而使用OT的离线阶段需要大约2.9×105s。当使用平方函数时，在线阶段的性能显着提高，因为大多数乱码电路被秘密共享值上的乘法所取代。特别是，我们的第一个协议的在线阶段只需要653.0秒。o ffl ine阶段的运行时间增加，显示两个阶段之间的交易。使用客户端辅助乘法三元组，o ffl ine阶段进一步减少到大约1.5×104s，在线阶段的开销。

As shown in the table, when RELU function is used, the online phase of our first protocol takes 4239.7s, while the offline phase using OT takes around $2.9 \times 10^5 s$. When the square function is used, the performance of the online phase is improved significantly, as most of the garbled circuits are replaced by multiplications on secret shared values. In particular, it only takes 653.0s for the online phase of our first protocol. The running time of the offline phase is increased, showing a trade-off between the two phases. Using client-aided multiplication triplets, the offline phase is further reduced to about 1.5×10^4s, with an overhead on the online phase.

由于大量的交互和高通信，WAN设置的神经网络训练尚不实用。为了在神经网络中执行一轮前向和后向传播，使用RELU功能在线阶段需要30.52秒，并且使用基于LHE的方法，相位阶段需要大约2200秒。总运行时间在轮数中是线性的，在这种情况下约为7000。

Due to high number of interactions and high communication, the neural network training on WAN setting is not yet practical. To execute one round of forward and backward propagation in the neural network, the online phase takes 30.52s using RELU function and the offline phase takes around 2200s using LHE-based approach. The total running time is linear in the number of rounds, which is around 7000 in this case.

在准确性方面，我们的协议训练模型使用RELU可以达到93.4%，使用square函数可以达到93.1%。在实践中，还有其他类型的神经网络可以达到更好的准确性。例如，卷积神经网络被认为更适合于图像处理任务。在这样的神经网络中，神经元没有完全连接，数据和系数之间的内积被2-D卷积所取代。原则上，我们也可以支持这种神经网络，因为可以使用加法和乘法来计算卷积。然而，使用安全计算中的快速傅里叶变换等技术来改善性能是一个有趣的开放性问题。尝试各种MPC友好的激活是研究的另一种途径。

In terms of the accuracy, the model trained by our protocol can reach 93.4% using RELU and 93.1% using the square function. In practice, there are other types of neural networks that can reach better accuracy. For example, the *convolutional* neural networks are believed to work better for image processing tasks. In such neural networks, the neurons are not fully connected and the inner product between the data and the coefficients is replaced by a 2-D convolution. In principle, we can also support such neural networks, as the convolution can be computed using additions and multiplications. However, improving the perfomance using techniques such as Fast Fourier Transform inside secure computation is interesting open questions. Experimenting with various MPC-friendly activations is another avenue for research.

6.4 Experiments for predictions.

表5总结了预测的成本。我们使用来自MNIST的评估数据集的样本，其中d = 784.我们使用的神经网络与6.3节中描述的相同。它有2个隐藏层，每个层有128个神经元，并输出一个包含10个元素的热矢量。我们假设模型在两台服务器之间保密。如表中所示，在线阶段非常快，这对延迟关键应用程序是有利的，因为可以独立于数据预先计算o ffl ine阶段。此外，由于矢量化，当并行进行多个预测时，时间会线性增长。

Table 5 summarizes the cost of predictions. We use samples from the evaluation dataset of MNIST with $d = 784$. The neural network we use is the same as described in Section 6.3. It has 2 hidden layers with 128 neurons each, and outputs a hot vector with 10 elements. We assume that the models remain secret shared between the two servers. As shown in the table, the online phase is extremely fast, which benefits latency critical applications as the offline phase can be precomputed independently of the data. In addition, because of vectorization, the time grows sublinearly when making multiple predictions in parallel.

	k	Linear (ms)		Logistic (ms)		Neural (s)	
		Online	Offline	Online	Offline	Online	Offline
LAN	1	0.20	2.5	0.59	2.5	0.18	4.7
	100	0.22	51	3.9	51	0.20	13.8
WAN	1	72	620	158	620	0.57	17.8
	100	215	2010	429	2010	1.2	472

Table 5: Online and offline performances for privacy preserving predicition. $d = 784$. The neural network is the same as the one in Section 6.3.

表5：用于隐私保护预测的在线和其他表现。d = 784.神经网络与6.3节中的神经网络相同。

6.5 Microbenchmarks 微基准测试

In addition to evaluating the end-to-end performance of our system, we present microbenchmarks to show the effect of our major optimizations individually in this section.

除了评估我们系统的端到端性能之外，我们还提供微基准测试，以在本节中单独显示我们主要优化的效果。

		Total (OT)	Total (LHE)	GC
LAN	$k = 1000$	0.028s	5.3s	0.13s
	$k = 10,000$	0.16s	53s	1.2s
	$k = 100,000$	1.4s	512s	11s
WAN	$k = 1000$	1.4s	6.2s	5.8s
	$k = 10,000$	12.5s	62s	68s
	$k = 100,000$	140s	641s	552s

Table 6: Comparison of our decimal multiplication and the fixed-point multiplication using garbled circuit.

表6：使用乱码电路的十进制乘法和固定点乘法的比较。

Arithmetic on shared decimal numbers. Table 6 compares the performance of our new scheme for decimal multiplications with that of fixed-point multiplication using garbled circuit. We run k multiplications in parallel and compare the total time of our scheme (online + OT-based offline and online + LHE-based offline) to GC-based fixed-point multiplication, with a 16-bit integer part and a 16-bit decimal part. As shown in the table, our OT-based approach is faster than garbled circuits by a factor of $5-8\times$ on LAN, and a factor of $4-5\times$ on WAN networks. The typical number of multiplications in parallel, needed to train on our datasets is close to 100,000. Though our LHE-based approach is much slower than using garbled circuits on LAN networks, and is comparable on WAN networks, we will show in the next microbenchmark that the LHE-based approach benefits the most from vectorization, which makes it even faster than our OT-based approach on WAN networks.

共享十进制数的算术。 表6比较了我们的十进制乘法新方案与使用乱码电路的固定点乘法方案的性能。 我们并行运行k次乘法，并将我们的方案（在线+基于OT和在线+基于LHE的o ffl ine）的总时间与基于GC的固定点乘法进行比较，具有16位整数部分和16位 小数部分。如表中所示，我们的基于OT的方法比局域网上的乱码电路快5-8倍，而WAN网络则高4-5倍。 在我们的数据集上训练所需的并行乘法的典型数量接近100,000。 虽然我们的基于LHE的方法比在LAN网络上使用乱码电路要慢得多，并且在WAN网络上具有可比性，但我们将在下一个微基准测试中表明基于LHE的方法在矢量化方面受益最多，这使得它比我们的更快 基于OT的WAN网络方法。

Note that if the client-aided offline phase is used, the speedup is more significant. Typically it only takes milliseconds in total for $k = 10,000$. However, as the security model is weakened when using client-aided multiplication triplets, we did not compare it directly with the fixed-point multiplication.

请注意，如果使用客户端辅助阶段，则加速更为显着。 通常，对于k = 10,000，总共只需要几毫秒。 但是，由于在使用客户端辅助乘法三元组时安全模型被削弱，我们没有直接将其与固定点乘法进行比较。

	d	Online	Online Vec	OT	OT Vec	LHE	LHE Vec
LAN	100	0.37ms	0.22ms	0.21s	0.05s	67s	1.6s
	500	1.7ms	0.82ms	1.2s	0.28s	338s	5.5s
	1000	3.5ms	1.7ms	2.0s	0.46s	645s	10s
WAN	100	0.2s	0.09s	14s	3.7s	81s	2s
	500	0.44s	0.20ss	81s	19s	412s	6.2s
	1000	0.62s	0.27s	154s	34s	718s	11s

Table 7: Speedup from vectorization. $|B| = 128$.

表7：来自矢量化的加速。 jBj = 128。

Vectorization. Table 7 shows the speedup gained from vectorization. As a basic operation in our training, we need to multiply a shared $|B| \times d$ matrix and a $d \times 1$ vector. $|B|$ is set to 128 and d varies from 100 to 1000. In the unoptimized version, we compute the inner product between the vector and each row of the matrix; in the vectorized version, we compute the matrix-vector multiplication. As shown in Table 7, the online time is improved by around $2\times$. The OT-based offline phase is improved by $4\times$. The LHE-based offline phase is improved by $41-66\times$ and it is faster than the OT-based offline phase on WAN networks because of the vectorization.

矢量。 表7显示了从矢量化获得的加速比。 作为训练中的基本操作，我们需要乘以共享| B |×d矩阵和d×1矢量。 | B | 设置为128，d在100到1000之间变化。在未优化的版本中，我们计算向量和矩阵的每一行之间的内积; 在矢量化版本中，我们计算矩阵向量乘法。 如表7所示，在线时间提高了约2倍。 基于OT的相位相位提高了4倍。 基于LHE的离线阶段提高了41-66倍，并且由于矢量化，它比WAN网络上基于OT的离线阶段更快。

新的逻辑函数。我们将计算新逻辑函数的成本与使用10次多项式近似逻辑函数进行比较。对于多项式近似，我们使用我们的十进制乘法方案，并使用Horner规则使用9次连续乘法计算它。表8显示了128次并行功能评估的运行时间（仅用于分摊网络延迟的影响）。如表中所示，除非在LAN网络中使用客户端辅助乘法三元组，这会削弱安全模型，否则我们的新逻辑函数比使用多项式逼近(3.5 - 1511×)快得多

	New Logistic	Poly Total Client-aided	Poly Total OT	Poly Total LHE
LAN	0.0045s	0.0005s	0.025s	6.8s
WAN	0.2s	0.69s	2.5s	8.5s

Table 8: Performance of our new logistic function and polynomial approximation.

表8：我们的新逻辑函数和多项式近似的性能。

New logistic function. We compare the cost of calculating our new logistic function with approximating the logistic function using a degree 10 polynomial. For the polynomial approximation, we use our scheme for decimal multiplications and compute it using 9 sequential multiplications using the Horner's rule. Table 8 shows the running time for 128 parallel evaluations of the function (just to amortize the effect of network delay). As shown in the table, unless using client-aided multiplication triplets in LAN networks, which weakens the security model, our new logistic function is dramatically faster than using polynomial approximation ($3.5 - 1511\times$).

Acknowledgements

We thank Jing Huang from Visa Research for helpful discussions on machine learning, and Xiao Wang from University of Maryland for his help on the EMP toolkit. The work was partially supported by NSF grants #1514261 and #1526950.

References

[1] Arcene data set. `https://archive.ics.uci.edu/ml/datasets/Arcene`. Accessed: 2016-07-14.

[2] Eigen library. `http://eigen.tuxfamily.org/`.

[3] EMP toolkit. `https://github.com/emp-toolkit`.

[4] Gisette data set. `https://archive.ics.uci.edu/ml/datasets/Gisette`. Accessed: 2016-07-14.

[5] GMP library. `https://gmplib.org/`.

[6] MNIST database. `http://yann.lecun.com/exdb/mnist/`. Accessed: 2016-07-14.

[7] NTL library. `http://www.shoup.net/ntl/`.

[8] Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 308–318.

[9] Aono, Y., Hayashi, T., Trieu Phong, L., and Wang, L. Scalable and secure logistic regression via homomorphic encryption. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (2016), ACM, pp. 142–144.

[10] Asharov, G., Lindell, Y., Schneider, T., and Zohner, M. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the ACM CCS 2013* (2013).

[11] Bellare, M., Hoang, V. T., Keelveedhi, S., and Rogaway, P. Efficient garbling from a fixed-key blockcipher. In *Security and Privacy (SP), 2013 IEEE Symposium on* (2013), IEEE, pp. 478–492.

[12] Bellare, M., Hoang, V. T., and Rogaway, P. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 784–796.

[13] Bunn, P., and Ostrovsky, R. Secure two-party k-means clustering. In *Proceedings of the 14th ACM conference on Computer and communications security* (2007), ACM, pp. 486–497.

[14] Canetti, R. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on* (2001), IEEE, pp. 136–145.

[15] Chaudhuri, K., and Monteleoni, C. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems* (2009), pp. 289–296.

[16] Damgard, I., Geisler, M., and Kroigard, M. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography 1*, 1 (2008), 22–31.

[17] Demmler, D., Schneider, T., and Zohner, M. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS* (2015).

[18] Du, W., and Atallah, M. J. Privacy-preserving cooperative scientific computations. In *csfw* (2001), vol. 1, Citeseer, p. 273.

[19] Du, W., Han, Y. S., and Chen, S. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *SDM* (2004), vol. 4, SIAM, pp. 222–233.

[20] Gascon, A., Schoppmann, P., Balle, B., Raykova, M., Doerner, J., Zahur, S., and Evans, D. Secure linear regression on vertically partitioned datasets.

[21] Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of The 33rd International Conference on Machine Learning* (2016), pp. 201–210.

[22] Gilad-Bachrach, R., Laine, K., Lauter, K., Rindal, P., and Rosulek, M. Secure data exchange: A marketplace in the cloud. Cryptology ePrint Archive, Report 2016/620, 2016. `http://eprint.iacr.org/2016/620`.

[23] Gilad-Bachrach, R., Laine, K., Lauter, K., Rindal, P., and Rosulek, M. Secure data exchange: A marketplace in the cloud.

[24] Guyon, I., Gunn, S., Ben-Hur, A., and Dror, G. Result analysis of the nips 2003 feature selection challenge. In *Advances in neural information processing systems* (2004), pp. 545–552.

[25] Hastie, T., Tibshirani, R., and Friedman, J. The elements of statistical learning – data mining, inference, and prediction.

[26] Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. Extending oblivious transfers efficiently. *Advances in Cryptology-CRYPTO 2003* (2003), 145–161.

[27] Jagannathan, G., and Wright, R. N. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* (2005), ACM, pp. 593–599.

[28] Kamara, S., Mohassel, P., and Raykova, M. Outsourcing multi-party computation. *IACR Cryptology ePrint Archive* (2011), 272.

[29] Kolesnikov, V., and Schneider, T. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming* (2008), Springer, pp. 486–498.

[30] Lindell, Y., and Pinkas, B. Privacy preserving data mining. In *Annual International Cryptology Conference* (2000), Springer, pp. 36–54.

[31] Lindell, Y., and Pinkas, B. A proof of security of yaos protocol for two-party computation. *Journal of Cryptology 22*, 2 (2009), 161–188.

[32] Livni, R., Shalev-Shwartz, S., and Shamir, O. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems* (2014), pp. 855–863.

[33] Malkhi, D., Nisan, N., Pinkas, B., Sella, Y., et al. Fairplay-secure two-party computation system.

[34] Nayak, K., Wang, X. S., Ioannidis, S., Weinsberg, U., Taft, N., and Shi, E. Graphsc: Parallel secure computation made easy. In *2015 IEEE Symposium on Security and Privacy* (2015), IEEE, pp. 377–394.

[35] Nikolaenko, V., Ioannidis, S., Weinsberg, U., Joye, M., Taft, N., and Boneh, D. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 801–812.

[36] Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D., and Taft, N. Privacy-preserving ridge regression on hundreds of millions of records. In *Security and Privacy (SP), 2013 IEEE Symposium on* (2013), IEEE, pp. 334–348.

[37] Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques* (1999), Springer, pp. 223–238.

[38] Peikert, C., Vaikuntanathan, V., and Waters, B. A framework for efficient and composable oblivious transfer. *Advances in Cryptology–CRYPTO 2008* (2008), 554–571.

[39] Sanil, A. P., Karr, A. F., Lin, X., and Reiter, J. P. Privacy preserving regression modelling via distributed computation. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (2004), ACM, pp. 677–682.

[40] Shokri, R., and Shmatikov, V. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 1310–1321.

[41] Slavkovic, A. B., Nardi, Y., and Tibbits, M. M. " secure" logistic regression of horizontally and vertically partitioned distributed databases. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)* (2007), IEEE, pp. 723–728.

[42] Song, S., Chaudhuri, K., and Sarwate, A. D. Stochastic gradient descent with differentially private updates. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE* (2013), IEEE, pp. 245–248.

[43] Vaidya, J., Yu, H., and Jiang, X. Privacy-preserving svm classification. *Knowledge and Information Systems 14*, 2 (2008), 161–178.

[44] Wang, X., Malozemoff, A. J., and Katz, J. Faster two-party computation secure against malicious adversaries in the single-execution setting. Cryptology ePrint Archive, Report 2016/762, 2016. `http://eprint.iacr.org/2016/762`.

[45] Wu, S., Teruya, T., Kawamoto, J., Sakuma, J., and Kikuchi, H. Privacy-preservation for stochastic gradient descent application to secure logistic regression. *The 27th Annual Conference of the Japanese Society for Artificial Intelligence 27* (2013), 1–4.

[46] Yao, A. C. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on* (1982), IEEE, pp. 160–164.

[47] Yu, H., Vaidya, J., and Jiang, X. Privacy-preserving svm classification on vertically partitioned data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2006), Springer, pp. 647–656.