Neural Networks 121 (2020) 484–496

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

# Privacy-enhanced multi-party deep learning

Maoguo Gong *, Jialun Feng, Yu Xie

*School of Electronic Engineering, Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education, Xidian University, Xi'an, Shaanxi Province 710071, China*

ARTICLE INFO

*Article history:*
Received 5 March 2019
Received in revised form 1 August 2019
Accepted 3 October 2019
Available online 11 October 2019

*Keywords:*
Privacy
Multi-party deep learning
Differential privacy
Homomorphic encryption
Privacy budget

ABSTRACT

In multi-party deep learning, multiple participants jointly train a deep learning model through a central server to achieve common objectives without sharing their private data. Recently, a significant amount of progress has been made toward the privacy issue of this emerging multi-party deep learning paradigm. In this paper, we mainly focus on two problems in multi-party deep learning. The first problem is that most of the existing works are incapable of defending simultaneously against the attacks of honest-but-curious participants and an honest-but-curious server without a manager trusted by all participants. To tackle this problem, we design a privacy-enhanced multi-party deep learning framework, which integrates differential privacy and homomorphic encryption to prevent potential privacy leakage to other participants and a central server without requiring a manager that all participants trust. The other problem is that existing frameworks consume high total privacy budget when applying differential privacy for preserving privacy, which leads to a high risk of privacy leakage. In order to alleviate this problem, we propose three strategies for dynamically allocating privacy budget at each epoch to further enhance privacy guarantees without compromising the model utility. Moreover, it provides participants with an intuitive handle to strike a balance between the privacy level and the training efficiency by choosing different strategies. Both analytical and experimental evaluations demonstrate the promising performance of our proposed framework.



## 1. Introduction

In recent years, deep learning has achieved impressive success in many fields (e.g., computer vision Camuñas-Mesa, Serrano-Gotarredona, Ieng, Benosman, & Linares-Barranco, 2018, natural language processing Devlin, Chang, Lee, & Toutanova, 2018, speech processing Zhang, Li, Jin, & Choe, 2015 and human -machine games Silver et al., 2016, etc.), and its performance has approached or even exceeded human in a wide range of applications, such as in face recognition (Cao, Wang, Gao, & Li, 2018; Huang, Dai, Ren, & Lai, 2017), handwritten digit recognition (Liang & Hu, 2015; Wang, Xu, Yang, & Zurada, 2018) and playing Go (Silver et al., 2017). A prerequisite for these successes is the massive data available for model training. However, massive data collection from multiple sources may present privacy issues. On the one hand, data may contain both identity information and private information. For example, a photo contains both faces and elements which may suggest religious beliefs. On the other hand, it is illegal to share sensitive data, like medical data stored by medical institutions. Once these kinds of data are collected centrally, the data may be stored permanently, and used for the purpose which the original data owner is unaware of. Generally, these private or sensitive data are scattered across multiple research institutions or companies, and they are unwilling or unable to share data with each other. If researchers want to train a deep learning model with high accuracy, they can only perform on their own data. Nevertheless, data owned by an institution may be very limited (e.g., a school clinic), which will result in an over-fitting deep learning model.

Therefore, there raises an urgent demand for multi-party deep learning, where multiple participants jointly train a deep learning model to achieve common objectives without revealing their own training data. The existing literature on multi-party deep learning (Phong et al., 2018; Shokri & Shmatikov, 2015; Zhang et al., 2017) shares the same paradigm: all participants share a common deep neural network architecture and common training objectives. Each participant independently trains the local model on their private data to obtain gradients, and then uploads the gradients to a pre-set central server to update the weights of the global model. After that, each participant downloads the updated weights from the central server to update the local model. Loop the above processes until the common objectives are achieved. Although the above paradigm makes it impossible for an attacker who is curious about private data to access the data directly, private data can still be indirectly revealed through certain technologies. For instance, Hitaj, Ateniese, and Perez-Cruz (2017)

* Corresponding author.
*E-mail address:* gong@ieee.org (M. Gong).

https://doi.org/10.1016/j.neunet.2019.10.001
0893-6080/

**Table 1**
Comparison of existing privacy-preserving multi-party deep learning frameworks.

| Framework | Potential information leakage to server | Potential information leakage to other participants | Need a manager trusted by all participants |
|---|---|---|---|
| Shokri and Shmatikov (2015) | Yes | No | No |
| Phong, Aono, Hayashi, Wang, and Moriai (2018) | No | Yes | No |
| Zhang, Ji, Wang, and Wang (2017) | No | No | Yes |
| Ours | No | No | No |

employed a generative adversarial network to deceive a victim into releasing his private information. Phong et al. (2018) restored the training data of a victim through his uploaded gradients. The local model is reverse-engineered to reveal the private data of a victim (Pathak, Rane, & Raj, 2010). To tackle the problem of indirect leakage, the existing works mainly adopt two types of schemes. One is differential privacy, which reduces the leakage of private information for any single data by adding noise, where the amount of noise and the privacy level is controlled by a parameter privacy budget $\epsilon$. The other is homomorphic encryption, which can perform algebraic operations in ciphertext. Specifically, Shokri and Shmatikov (2015) presented a method to selectively upload and download gradients, and utilized a differential privacy technique to add noise to the uploaded gradients. Zhang et al. (2017) applied threshold secret sharing method (i.e., the global model updates only when the number of uploaded participants reaches a certain threshold). Besides, local differential privacy techniques and multiplicative homomorphic encryption are applied under the control of a manager to ensure the security of the training process. Phong et al. (2018) proposed to apply additive homomorphic encryption to ensure that private information is not leaked from the central server.

In spite of the achievements of privacy preserving in multi-party deep learning, there still exist two problems. The first problem is that the existing works are incapable of defending simultaneously against the attacks of honest-but-curious participants and an honest-but-curious server without a manager that all participants trust. In the framework of Shokri and Shmatikov (2015), private information may be leaked to an honest-but-curious central server (Phong et al., 2018). Under the framework of Zhang et al. (2017), since the secret key is kept by the central server, a manager that all participants trust is required. Otherwise, private information may be leaked from the central server. Besides, in the framework of Phong et al. (2018), there is no guarantee that private information will not be revealed to an honest-but-curious participant. Table 1 shows the comparison of existing privacy-preserving multi-party deep learning frameworks. The other problem is that existing frameworks consume high total privacy budget when applying differential privacy for preserving privacy, which leads to a high risk of privacy leakage (Abadi et al., 2016). The existing frameworks like Shokri and Shmatikov (2015) and Zhang et al. (2017) allocated fixed privacy budget at each iteration. Since deep learning algorithms typically require numerous iterations, the consumption of total privacy budget is generally high. However, in the early stages of training, random weights are far from optimal, and the gradients are usually large, which makes it possible to allocate less privacy budget (i.e., add more noise to the gradients). To the best of our knowledge, the work related to the dynamic allocation of privacy budget for deep learning is only proposed in Lee and Kifer (2018) for the centralized scenarios. However, it is not suitable for the scenarios of multi-party deep learning, since it updates weights only when the gradients with noise can reduce the objective function value. It is worth noting that this method is greedy, which is easy to fall into the poor local optimum. In addition, it is obviously unreasonable for each participant to reduce the objective function value at each iteration. Thus, it is necessary for a novel privacy-preserving multi-party deep learning framework to fix these problems.

Motivated by the above observations, we design a privacy-enhanced multi-party deep learning framework with two characteristics. For one thing, private information is neither leaked among participants nor leaked from a central server without a manager that all participants trust. For another, our framework consumes less total privacy budget without sacrificing utility. In order to defend simultaneously against honest-but-curious participants and an honest-but-curious server, two technologies are utilized. Firstly, the sparse vector technique (Dwork, Roth, et al., 2014; Shokri & Shmatikov, 2015), a differential privacy technique that combines gradients selection with noise addition on gradients, is applied. It does not expose the entire gradients, thereby reducing the risk of private information leakage and ensuring that private information is not leaked to an honest-but-curious participant. Next, the Paillier algorithm (Paillier, 1999), an efficient homomorphic encryption scheme, is applied to ensure that private information is not leaked to an honest-but-curious server. In order to tackle the problem of consuming high total privacy budget that corresponds to differential privacy, three strategies for allocating privacy budget at each epoch (in a uniform, exponential, or logarithmic manner) are proposed. By dynamically allocating privacy budget, more noise is added in the early stages of training to reduce the consumption of privacy budget, thereby further reducing the risk of private information leakage. In the later stages of training, the original noise level is restored to ensure the model utility does not decrease. Besides, it provides participants with an intuitive handle to strike a balance between privacy level and training efficiency by choosing different strategies.

In a nutshell, our main contributions are as follows:

(1) We design a privacy-enhanced multi-party deep learning framework that integrates differential privacy and homomorphic encryption to prevent potential privacy leakage to other participants and central server without requiring a manager that all participants trust.

(2) In order to alleviate the problem of consuming high total privacy budget, we propose three strategies for dynamically allocating privacy budget at each epoch to reduce the consumption of privacy budget, providing stronger privacy guarantees without sacrificing model utility.

(3) We analytically and experimentally evaluate our framework on two benchmark datasets. The results demonstrate the effectiveness and strong privacy guarantees of our framework.

The remainder of this paper is organized as follows. The preliminaries of work is discussed in Section 2. In Section 3, we present our privacy-enhanced multi-party deep learning framework in detail. The empirical evaluations are provided in Section 4 and the conclusion is illustrated in Section 5.

## 2. Preliminaries

### 2.1. Deep learning

Deep learning (Goodfellow, Bengio, Courville, & Bengio, 2016; Mahmud, Kaiser, Hussain, & Vassanelli, 2018) aims to automatically learn multiple levels of abstract representations of data. A

deep neural network (DNN) typically consists of multiple layers, with hundreds of millions of adjustable weights. The goal is to minimize the objective function by adjusting these weights. Due to the complex structure of DNN, deep learning models are most probably to be optimized by stochastic gradient descent (SGD) (Zinkevich, Weimer, Li, & Smola, 2010) or its variants (Bukovsky & Homma, 2017; Chang, Lin, & Zhang, 2018), where the training process is a process from randomized weights to fine-tuned weights.

Stochastic gradient descent (Zinkevich et al., 2010) selects a subset (called a mini-batch) of samples from training data to compute the gradients based on the loss of the objective function through the back-propagation (Rumelhart, Hinton, & Williams, 1986) procedure, and then updates the weights by averaging the gradients at each iteration. The processes can be formulated as:

$$w := w - \alpha * \frac{\sum_{i=1}^{m} \nabla_w{}^i}{m} \tag{1}$$

where $w$ represents the weights, $\alpha$ is the learning rate, $m$ is the size of a mini-batch, and $\nabla_w{}^i$ denotes the gradients computed by sample $i$. An example of a participant's local model training is as follows: The model begins with random weights or the weights downloaded from a central server. In the early stages of training, the SGD is applied and a high learning rate is set to speed up the update process. In the later stages of training, a low learning rate is set to fine tune the weights to converge. Note that our privacy-enhanced multi-party deep learning framework can be easily embedded into a model optimized with gradient descent, meaning that our model is widely available for almost all deep learning models.

#### 2.1.1. Distributed stochastic gradient descent

Distributed stochastic gradient descent (Dean et al., 2012) assumes that multiple participants train local models on their own private data to obtain local gradients simultaneously and independently. After each round of local training, they share their local gradients to a central server to jointly update a global model.

It should be noted that the local gradients of each participant are not gradients in the ordinary sense, but represent the results of the updated weights minus the original weights. We use $\Delta w^i$ to represent the local gradients of participant $i$, ${w_{\text{new}}}^i$ to denote the updated weights that may be trained over several mini-batches, and ${w_{\text{old}}}^i$ to indicate the original weights before the update. The calculation of local gradients is formulated as:

$$\Delta w^i = {w_{\text{new}}}^i - {w_{\text{old}}}^i \tag{2}$$

Specifically, at each epoch, $N$ participants simultaneously train replicas of the neural network over their own private data to obtain corresponding local gradients $\Delta w^i$ and send them to the central server to update the global weights $w_{\text{global}}$. The central server then updates the global weights by:

$$w_{\text{global}} := w_{\text{global}} + \sum_{i=1}^{\tau} \Delta w^i \tag{3}$$

where $\tau$ represents the size of a group of participants. In particular, $\tau = 1$ corresponds to an asynchronous protocol that $w_{\text{global}}$ is updated as long as a participant has uploaded the gradients; $\tau = N$ corresponds to a synchronous protocol that $w_{\text{global}}$ is updated only when all participants have uploaded the gradients.

### 2.2. Attack model

Typically, an attack model is required to verify the reliability of the framework. Like most other privacy-preserving multi-party deep learning frameworks (e.g., Phong et al., 2018; Shokri & Shmatikov, 2015; Zhang et al., 2017), we assume an attack model with the following characteristics: (1) The manager that all participants trust does not exist. Thus, private information cannot be disclosed to the manager if a manager is required. (2) Participants are honest-but-curious. It means that each participant operates in strict accordance with the predetermined processes, but is curious about the private information of other participants. Furthermore, participants may collude with each other in an attempt to expose the private information of one of the participants. (3) Central server is also honest-but-curious, which may steal private information such as model parameters and uploaded gradients.

We note that the above attack model is realistic. Normally, we can hardly find a manager trusted by all participants because it is difficult to judge whether a manager is trustworthy. In addition, for small companies or small medical institutions, they have strong willingness to collaborate for obtaining a model with higher accuracy than a model trained only based on their own data. Thus, they are only curious about the data of other participants and do not deliberately undermine the training processes. Moreover, the central server may be rented from someone other than participants. The owner of the central server will not break the protocol but may be curious about what private task is being trained or may steal the model parameters.

### 2.3. Differential privacy

Differential privacy (Dwork, 2008) constitutes a strong privacy standard for publishing an aggregated result without revealing too much private information about any single item in individual dataset. It is defined in terms of adjacent datasets. Two datasets are adjacent if they differ only in one single data record (i.e., one data record is in one dataset and absent in the other). In our framework, we utilize differential privacy to ensure that the private information of any data for each participant is not revealed too much when aggregated in central server.

**Definition 1** ($\epsilon$-*differential Privacy (Dwork, 2008)*)**.** A randomized algorithm $A$ satisfies $\epsilon$-differential privacy for any output $O$ of $A$ and for any two datasets $D$, $D'$ differing at most a single item, if and only if $A$ fulfills:

$$Pr[A(D) \in O] \leq e^{\epsilon} Pr[A(D') \in O] \tag{4}$$

In Definition 1, the privacy budget $\epsilon$ controls the extent to which the output distribution of two adjacent datasets may differ. A smaller value of $\epsilon$ leads to the two distributions being more similar, meaning that it is more difficult to distinguish which one of two adjacent datasets the single data record is in. Therefore, the privacy budget $\epsilon$ is generally used to measure and control the privacy level of a randomized algorithm $A$. A smaller value of $\epsilon$ provides a stronger privacy guarantee.

In practice, a general paradigm for converting a real-valued function $f$ to a randomized algorithm $A$ that satisfies $\epsilon$-differential privacy is via additive noise. For example, the Laplace noise mechanism (Dwork, McSherry, Nissim, & Smith, 2006) is defined by

$$A(D) = f(D) + Lap(\frac{\Delta f}{\epsilon}) \tag{5}$$

where $\Delta f$ represents the sensitivity of $f$, which is defined as $\max_{D,D'} ||f(D) - f(D')||_1$. $Lap(\frac{\Delta f}{\epsilon})$ represents a random variable sampled from the Laplace distribution with zero mean and scale $\frac{\Delta f}{\epsilon}$. The mechanism shows that the amount of noise is relevant to the privacy budget $\epsilon$ and the sensitivity $\Delta f$. A smaller value of $\epsilon$ brings more noise when $\Delta f$ is fixed. In our case, $f$ computes local gradients and shares the gradients to a central server. Thus, the

noise is added to the gradients. An example of adding noise to the gradients to ensure $\epsilon$-differential privacy can be formulated as

$$\Delta w' = \Delta w + Lap(\frac{\Delta f}{\epsilon}) \tag{6}$$

where $\Delta f$ is the absolute distance between maximum value of the gradient and minimum value of the gradient. Since the value of gradient may be infinite, the gradient is clipped within a range (e.g., [−0.001,0.001]) to reduce the amount of noise. In our experiment, we assume the same sensitivity for all weights (i.e., $\Delta f = 0.001 - (-0.001) = 0.002$), but this is not required, and each weight may have a different sensitivity. Since $\Delta f$ is fixed, if the privacy budget $\epsilon$ allocated is less, more noise will be added to the gradients, which is more likely to reduce the utility of the model.

Differential privacy also facilitates modular design and analysis of differentially private algorithms, due to its property of composability. If all the components of an algorithm satisfy differential privacy, so is their composition.

**Theorem 1** (*Sequential Composition Theorem (McSherry & Talwar, 2007)*). *Assume that a set of differential privacy randomized algorithms $A = \{A_1, A_2, \ldots, A_m\}$ are executed sequentially on an entire dataset. If $A_i$ satisfies $\epsilon_i$-differential privacy and the random processes of any two algorithms are independent, then the randomized algorithms A satisfy $\sum_{i=1}^{m} \epsilon_i$-differential privacy.*

**Proof.** According to the definition of $\epsilon$-differential privacy (4) described above,

$$\begin{aligned} Pr[A(D) \in O] &= \prod_{i=1}^{m} Pr[A_i(D) \in O_i] \\ &\leq \prod_{i=1}^{m} (e^{\epsilon_i} \times Pr[A_i(D') \in O_i]) \\ &= e^{\sum_{i=1}^{m} \epsilon_i} \times \prod_{i=1}^{m} (Pr[A_i(D') \in O_i]) \\ &\leq e^{\sum_{i=1}^{m} \epsilon_i} \times Pr[A(D') \in O] \quad \square \end{aligned}$$

Currently, sequential composition is the most common and straightforward way to analyze the privacy budget consumption of a differentially private algorithm. When deep learning is integrated with differential privacy, the iterative training of deep learning is equivalent to sequentially executing multiple identical randomized algorithms that satisfy $\epsilon$-differential privacy. If a participant consumes the same privacy budget $\epsilon$ per epoch, the total privacy budget consumed for training $n$ epochs is $n*\epsilon$. Since $n$ is generally large in deep learning, the total privacy budget is high, leading to a high risk of privacy leakage.

Therefore, instead of consuming the same privacy budget per epoch like existing works, we propose three strategies for allocating privacy budget at each epoch to reduce the consumption of the total privacy budget thereby further reducing the risk of private information leakage, providing stronger privacy guarantees.

### 2.4. Homomorphic encryption

Homomorphic encryption (Rivest, Adleman, & Dertouzos, 1978) allows particular algebraic operations to be performed directly on ciphertext, and the result is still in an encrypted form. After decrypting this result, it matches the result of performing the same operations on plaintext. Since operations are performed on ciphertext, no information will be revealed if there is no secret key for decryption. Therefore, in our framework, we utilize homomorphic encryption to prevent uploaded local gradients and model parameters from being leaked when updating global weights and interacting with a central server.

The Paillier algorithm (Paillier, 1999) is an additive homomorphic encryption scheme based on decisional composite residuosity problem. The problem is considered to be computationally intractable in the field of cryptography. Therefore, in the absence of the secret key for decryption, it is hard to crack the ciphertext encrypted by the Paillier algorithm through powerful computation capacity to obtain the corresponding plaintext. The Paillier algorithm initiates the generation of the public key *pk* and secret key *sk* with a key size, wherein the key size determines the range of plaintext that can be encrypted. A larger key size provides a larger encoding range but results in lower efficiency. In the form of ciphertext, the Paillier algorithm can perform the homomorphic addition which is defined as

$$\text{Enc}(m_1) + \text{Enc}(m_2) = \text{Enc}(m_1 + m_2) \tag{7}$$

where $\text{Enc}(\cdot)$ represents the encryption operation, $m_1$, $m_2$ denote two plaintext messages. Different from fully homomorphic encryption (Gentry & Boneh, 2009) which can perform arbitrary algebraic operations on ciphertext, the Paillier algorithm can only perform homomorphic addition operations on ciphertext, but it is much more efficient. Thus, it can reduce the time overhead of applying homomorphic encryption, especially for deep learning training that requires numerous iterations. Furthermore, it supports any number of homomorphic additions, meaning that it does not cause incorrect decryption due to excessive homomorphic additions.

## 3. Our framework

### 3.1. Overview

The high-level architecture of our privacy-enhanced multi-party deep learning framework is illustrated in Fig. 1, with $N$ participants and a central server. The central server can be managed by a pre-set program or an honest-but-curious manager. Overall, $N$ participants independently and simultaneously perform their local learning processes on their limited local private data, and the central server aggregates the local gradients contributed by the participants. The goal of the training is to obtain a model with higher model utility than the model trained only on their own local data, without revealing the private information of their local private data. The dotted box in Fig. 1 details the learning process for one of the participants and the central server. Each participant trains a replica of the deep learning model architecture agreed by all participants to obtain local gradients. After applying a series of privacy-preserving techniques and our proposed strategies for dynamically allocating privacy budget, local gradients are uploaded to the central server for the update of the global weights. The global weights are then downloaded from the central server to update the local model. The above learning processes loop until the model converges. In Table 2, we describe all components of our framework in detail.

### 3.2. Participants

Participants are primarily responsible for independently training local deep learning models on their private data and uploading their gradients to the central server after applying a series of privacy-preserving techniques, thereby contributing to the global model while preventing private information leakage. Algorithm 1 shows the procedure of local learning for each participant. All

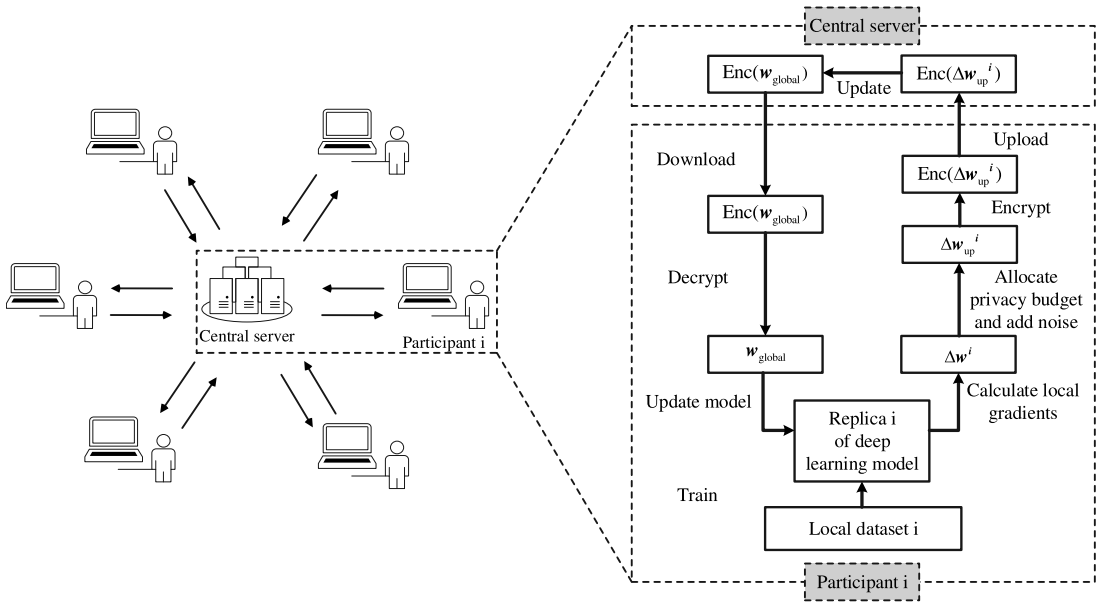



**Fig. 1.** High-level architecture of our privacy-enhanced multi-party deep learning framework, with multiple participants and a central server. The learning process for one of the participants and the central server is detailed in the dotted box.

**Table 2**
The common notations used in this paper.

| Notations | Explanation |
|---|---|
| $N$ | Number of participants |
| $\theta_{\text{up}}$ | Uploaded proportion |
| $\epsilon_{\max}, \epsilon_{\min}$ | Maximum and minimum privacy budget per epoch |
| $\gamma$ | The number of epochs to reach $\epsilon_{\max}$ |
| $w^i$ | Local weights of participant $i$ |
| $\Delta w^i$ | Local gradients of participant $i$ |
| $\Delta {w_{\text{up}}}^i$ | Uploaded gradients of participant $i$ |
| $w_{\text{global}}$ | Global weights |
| $c$ | Current epoch |
| $\epsilon^c$ | The privacy budget allocated in the $c$th epoch |
| $\tau$ | The threshold of the number of participants to update |
| $\text{Enc}(\cdot)$ | Encryption operation |

participants first jointly generate the public key $pk$ and the secret key $sk$ for the homomorphic encryption algorithm, in which the secret key $sk$ for decryption cannot be accessed except for the participants. In addition, all participants jointly decide the common strategy for allocating privacy budget which we detail in Section 3.2.2. In the $c$th epoch, the latest encrypted global weights $\text{Enc}(w_{\text{global}})$ are downloaded from the central server and decrypted to the global weights $w_{\text{global}}$. The corresponding local weights $w^i$ are then replaced with $w_{\text{global}}$ to update the local model, preventing over-fitting due to only training on local data. After that, the participants independently train the local models on their local data and compute the local gradients via (2).

After obtaining the local gradients, a series of privacy-preserving techniques are applied to prevent the leakage of private information. Instead of exposing entire local gradients, participants only select $\theta_{\text{up}}$ proportion of gradients to decrease the private information leakage. Furthermore, noise that satisfies $\epsilon$-differential privacy is added to the selected gradients to prevent the indirect leakage to the honest-but-curious participants, wherein $\epsilon^{(c)}$ is allocated by our proposed strategies which we detail in Section 3.2.2. The way of selecting and adding noise we choose in this paper is the sparse vector technique, since it can simultaneously mitigate the potential leakage of selecting and sharing gradients. After selecting gradients and adding noise, the upload gradients $\Delta {w_{\text{up}}}^i$ are obtained and then encrypted to the

**Algorithm 1** Procedure of local learning for participant $i$

**Input:** keys $pk$ and $sk$, uploaded proportion $\theta_{\text{up}}$, maximum and minimum privacy budget $\epsilon_{\max}$, $\epsilon_{\min}$ per epoch, the strategy for allocating privacy budget among (8) (9) (10)
**Output:** $w^i$

```
Initialize local weights w^i, total privacy budget ε_sum = 0 and current epoch c = 0
while common objectives are not achieved do
    download Enc(w_global) from the central server
    decrypt Enc(w_global) to obtain w_global with sk and replace the corresponding w^i
    train local model and compute local gradients Δw^i
    allocate privacy budget ε^(c) at the cth epoch based on the strategy
    select θ_up proportion of Δw^i and add noise that satisfies ε^(c)-differential privacy to the selected gradients to obtain Δw_up^i
    ε_sum = ε_sum + ε^(c)
    encrypt Δw_up^i to obtain Enc(Δw_up^i) with pk
    upload Enc(Δw_up^i) to the central server
    c = c + 1
end while
```

encrypted uploaded gradients $\text{Enc}(\Delta {w_{\text{up}}}^i)$. Finally, $\text{Enc}(\Delta {w_{\text{up}}}^i)$ is uploaded to the central server for the update of the global model.

### 3.2.1. Homomorphic encryption

Since our framework only performs the homomorphic addition on ciphertext, we just need to choose a homomorphic encryption algorithm that supports additive homomorphism. The homomorphic encryption algorithm we choose in this paper is the Paillier algorithm due to its efficiency and its ability to support any number of homomorphic additions. However, the Paillier algorithm only supports the calculation of non-negative integers, but the weights and gradients in deep learning are generally in the form of positive and negative floating point numbers. Therefore, they need to be encoded as non-negative integers before encryption and decoded into the original form after decryption. We can first encode floating point numbers as integers

through multiplying the large number (e.g., $10^6$) based on the pre-set precision and rounding the results. To deal with positive and negative integers, we can divide the encoding space (e.g., $\{0, 1, 2, \ldots, n-1\}$) into two halves. The first half is used to encode positive integers, and the second half is used to encode negative integers by adding $n$. The opposite operations are performed when decoding.

#### 3.2.2. Strategies for allocating privacy budget

Existing multi-party deep learning approaches that combine with differential privacy add a fixed amount of noise per epoch or reduce noise by applying a local differential privacy mechanism. However, they neglect the characteristics of deep learning. The training process of deep learning is a process from randomized weights to fine-tuned weights. In addition, the aggregation of gradients with additive noise from multiple participants can reduce noise to some extent. Therefore, adding more noise in the early stage of training can still perform well, and the original amount of noise is restored in the later stage of training to achieve high accuracy. The possibility of adding more noise means that the privacy budget can be reduced.

Inspired by these findings, three strategies are proposed for allocating privacy budget at each epoch. At the beginning of training, the minimum privacy budget $\epsilon_{\min}$ is allocated for adding more noise to reduce the consumption of the privacy budget, providing stronger privacy guarantee. Then the allocation of the privacy budget is gradually increased to the maximum privacy budget $\epsilon_{\max}$ to reduce the noise to the original level, so as not to decrease model utility. Specifically, the first strategy is that the privacy budget rises in a uniform manner, implying that the amount of noise is reduced uniformly to the lowest level, which is formulated as:

$$\epsilon^{(c)} = \min\{\epsilon_{\min} + c * \frac{\epsilon_{\max} - \epsilon_{\min}}{\gamma}, \epsilon_{\max}\} \tag{8}$$

The second strategy is that the privacy budget rises in an exponential manner, implying that the amount of noise is reduced slowly in the early stages, and reduced rapidly in the later stages, which is formulated as:

$$\epsilon^{(c)} = \min\{\epsilon_{\min} + (e^c - 1) * \frac{\epsilon_{\max} - \epsilon_{\min}}{e^\gamma - 1}, \epsilon_{\max}\} \tag{9}$$

The third is that the privacy budget rises in a logarithmic manner, implying that the amount of noise is reduced rapidly in the early stages, and reduced slowly in the later stages. The formula is as follows:

$$\epsilon^{(c)} = \min\{\epsilon_{\min} + \ln(c * \frac{(e^{(\epsilon_{\max} - \epsilon_{\min})} - 1)}{\gamma} + 1), \epsilon_{\max}\} \tag{10}$$

$\epsilon^{(c)}$ denotes the privacy budget at the $c$th epoch; $\epsilon_{\min}$ and $\epsilon_{\max}$ denote the minimum and maximum privacy budget at each epoch; $\gamma$ denotes the number of epochs required to reach $\epsilon_{\max}$, which controls the speed where the privacy budget rises. The smaller value of $\gamma$, the faster the privacy budget rises. In particular, when $c = 0$, $\epsilon = \epsilon_{\min}$; when $c = \gamma$, $\epsilon = \epsilon_{\max}$; when $c > \gamma$, $\epsilon = \epsilon_{\max}$, which means keeping the lowest noise training after $\gamma$ epochs until convergence. The value of $\epsilon_{\min}$ is recommended as $\epsilon_{\max}/10$, for the reasons that if the value of $\epsilon_{\min}$ is too large, there is almost no difference from not performing the strategies for allocating privacy budget. If the value of $\epsilon_{\min}$ is too small, it may lead to the poor training due to too much noise. It is noted that only one parameter $\gamma$ of our strategies needs to be determined by the participants. If $\gamma$ is too small, the noise level will quickly return to its original lowest level (i.e., allocate $\epsilon_{\max}$ at each epoch), resulting in less privacy budget savings. If $\gamma$ is too large, the number of epochs required to reach the lowest noise level is too much, which in turn increases the total privacy budget. The determination of $\gamma$ requires a trial to estimate the number of epochs required on a similar shared dataset or at local training. Then $\gamma$ is set to several epochs or a dozen of epochs before this number of epochs, since it needs several epochs to fit the model in the lowest noise level to achieve optimality.

### 3.3. Central server

**Algorithm 2** Procedure of global update for central server

```
Initialize encrypted global weights Enc(w_global)
Set an empty waiting list, threshold τ and t = 0
while common objectives are not achieved do
    event receive a download signal from participant i
        sent Enc(w_global) to participant i
    end event
    event receive an upload signal from participant i
        if i not in the waiting list then
            t = t + 1
            if t < τ then
                append i to the waiting list
                store Enc(Δw_up^i)
            else
                update Enc(w_global) via (11)
                empty the waiting list and set t = 0
            end if
        end if
    end event
end while
```

The central server is primarily responsible for updating and storing $\mathrm{Enc}(w_{\text{global}})$ which is available for participants to download without leaking any private information. Algorithm 2 shows the procedure of global update for the central server. The initialization of $\mathrm{Enc}(w_{\text{global}})$ can be uploaded by a participant, or set to all zero. When the central server receives a download signal from a participant, it provides the latest $\mathrm{Enc}(w_{\text{global}})$ that the participant can download entirely or selectively. When the central server receives an upload signal from participant $i$, if the number of participants who have uploaded is less than the threshold $\tau$, $\mathrm{Enc}(\Delta w_{\text{up}}{}^i)$ is stored and $i$ is added to the waiting list to prevent participant $i$ from uploading again, in case the gradients contributed by a participant are updated twice in an epoch. If the number of participants who have uploaded is equal to $\tau$, the waiting list will be emptied and $\mathrm{Enc}(w_{\text{global}})$ is updated by:

$$\mathrm{Enc}(w_{\text{global}}) := \mathrm{Enc}(w_{\text{global}}) + \sum_{i=1}^{\tau} \mathrm{Enc}(\Delta w_{\text{up}}{}^i) \tag{11}$$

The parameter $\tau$ controls the trade-off between the model utility and the training efficiency. A larger value of $\tau$ results in higher model utility, but lower training efficiency. In particular, $\tau = 1$ corresponds to an asynchronous protocol with the highest training efficiency but the lowest model utility; $\tau = N$ corresponds to a synchronous protocol with the highest model utility but the lowest training efficiency. Participants can choose an appropriate $\tau$ in advance, according to the actual situation. High $\tau$ can be chosen when high model utility is required or when all participants have similar computation capacity. Notice that in the central server, private information such as global weights and local gradients uploaded by participants are in ciphertext. Therefore, private information will not be disclosed to the honest-but-curious central server.

### 3.4. Analysis

In this part, we analyze the effectiveness, efficiency and privacy of our framework.

*3.4.1. Effectiveness*

We analyze the effectiveness of our strategies for allocating privacy budget at each epoch, which reduce the consumption of total privacy budget and do not decrease model utility despite adding more noise to the gradients. We first illustrate some characteristics of deep learning. The training process of deep learning is a process from randomized weights to fine-tuned weights. Besides, the optimization of deep neural networks is non-convex, with multiple local optimal solutions. In the early stages of training, the weights are randomly initialized, which are extremely far away from local optimal solutions, and the potential optimization space is large. In addition, the values of gradients are usually large at the early stages of training. In this circumstance, if we add more noise to the gradients, most of updated directions of the weights are not changed. Even if some updated directions may be reversed, they will be corrected during the aggregation of gradients with multiple participants or corrected in the next update. Furthermore, noise makes the search range of weights wider rather than directly toward the gradient, which may reach a potentially better local optimal solution. For instance, a differential privacy mechanism is used to prevent over-fitting while strengthening privacy guarantee in Jain, Kulkarni, Thakurta, and Williams (2015). Therefore, although more noise is added in the early stages of training, the training can substantially perform well and update toward the local optimal solution. According to the sequential composition theorem discussed in Section 2.3, for example, both training ten epochs with high noise that corresponds to $\epsilon = 1$ and training one epoch with low noise that corresponds to $\epsilon = 10$ consume the same total privacy budget. Since the early stages of training with high noise can still perform well, there is no need to train ten epochs with high noise, achieving the same accuracy as the one epoch of training with low noise, thereby saving the privacy budget and enhancing the privacy guarantees. In the later stages of training, when it is about to converge to a local optimum, a slight change of weights will result in a large change in the final output. Besides, training with high noise causes the network weights to oscillate around the local optimal solution, making it difficult to converge. In this case, if we gradually increase the privacy budget (i.e., reduce noise) to the maximum privacy budget, the model can reach the local optimum to the same extent as training with low noise, so as not to decrease the model utility. In the previous analysis, the premise that the strategy is effective is that the training can substantially perform well. Some potential pitfalls may lead to our proposed strategies or convergence failure. For example, if the maximum privacy budget $\epsilon_{max}$ or the uploaded proportion $\theta_{\text{up}}$ is too small, it may fail to converge due to too much noise. Although the strategies of adding more noise at the early stages of training may slow down convergence, they can reduce the privacy budget to provide stronger privacy guarantees while ensuring that the model utility does not decrease.

*3.4.2. Efficiency*

We analyze the efficiency of our framework from three perspectives. From the perspective of each participant, almost all privacy-preserving operations we apply, such as gradients selection, noise addition, encryption and decryption, can be designed to be executed in parallel to speed up the operations, since these operations are performed independently for each weight or for each gradient. Therefore, the privacy-preserving operations have little impact on the training efficiency of each participant.

From the perspective of central server, since the homomorphic encryption algorithm we apply (i.e., the Paillier algorithm) only supports homomorphic addition, which is much more efficient than those that support arbitrary algebraic operations, our framework deliberately performs operations on ciphertext only with additions to reduce the time overhead of homomorphic encryption. In addition, the update of global weights with homomorphic additions can be performed independently for each weight, so it can also be designed to be executed in parallel to speed up the update.

From the overall view, $N$ participants independently and simultaneously perform their local learning processes on their local data, which is equivalent to splitting a large dataset into $N$ parts and distributing them to $N$ machines for training (i.e., data parallelism). Compared with the distributed SGD (Dean et al., 2012), our framework attaches a series of privacy-preserving techniques. Among them, the running time of gradients selection and noise addition are negligible compared to the local training. The increase in time overhead may be caused by the strategies for allocating privacy budget at each epoch and the Paillier algorithm, which we experimentally evaluate in Section 4.

*3.4.3. Privacy*

We demonstrate our framework provides strong privacy guarantees under the attack model described in Section 2.2. Under the attack of an honest-but-curious participant, the uploaded gradients aggregated by $\tau$ participants can be determined via subtracting the latest $\text{Enc}(w_{\text{global}})$ from the $\text{Enc}(w_{\text{global}})$ at the next epoch and then decrypting the result. When $\tau > 1$, the uploaded gradients consist of at least the gradients of the two participants. Thus, the gradients of one of the participants are hard to be computed from the summed results. Even if $\tau = 1$ or the honest-but-curious participant colludes with other participants to calculate a victim's uploaded gradients, it is still difficult to reveal the victim's private information, since ${\Delta w_{\text{up}}}^i$ is only a small proportion of $\Delta w^i$ and contains noise that satisfies differential privacy. Furthermore, the strategies for allocating privacy budget at each epoch reduce the total privacy budget, which provide stronger privacy guarantees. Therefore, our framework ensures that private information is not leaked to honest-but-curious participants.

Under the attack of an honest-but-curious central server, all private information received from participants is encrypted by the Paillier algorithm, such as $\text{Enc}({\Delta w_{\text{up}}}^i)$ and $\text{Enc}(w_{\text{global}})$. Furthermore, the update process is performed directly on ciphertext. Due to the fact that central server cannot access the secret key *sk*, the central server can neither reveal the private information of participants nor steal the model weights. Even if the central server colludes with a participant to obtain *sk* to decrypt a victim's $\text{Enc}({\Delta w_{\text{up}}}^i)$, it is still difficult to reveal the victim's private information since ${\Delta w_{\text{up}}}^i$ is just a small proportion of $\Delta w^i$ and contains noise that satisfies with differential privacy. Therefore, our framework ensures that private information is not leaked to an honest-but-curious central server.

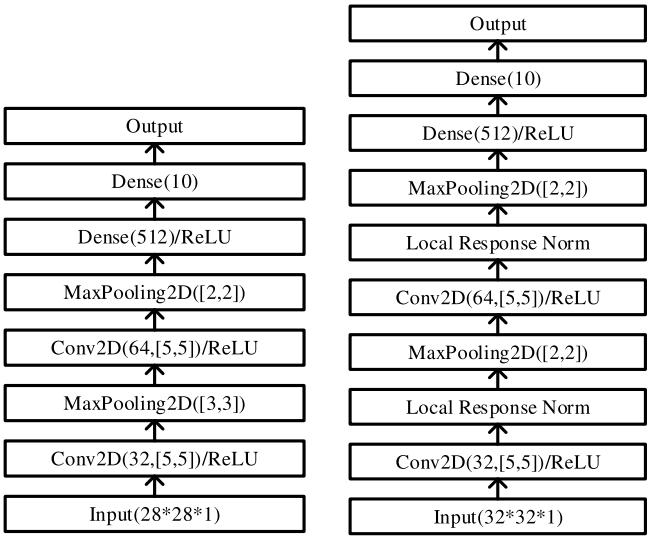
## 4. Experimental evaluation

### 4.1. Datasets

To demonstrate the performance of our framework, we experimentally evaluate on two benchmark datasets.

- MNIST[1] (LeCun, Bottou, Bengio, & Haffner, 1998) is the dataset of handwritten digits, formatted as $28 \times 28$ grayscale images, with 60,000 training samples and 10,000 testing samples. The digits have been size-normalized and centered. In our experiments, each participant owns 1% of the entire training dataset (i.e., 600 training samples).

[1] http://yann.lecun.com/exdb/mnist.




(a) MNIST (b) SVHN

**Fig. 2.** Deep learning architectures used for MNIST and SVHN datasets.

- SVHN[2] (Netzer et al., 2011) is the dataset of house numbers in Google Street View, formatted as $32 \times 32 \times 3$ RGB images, with approximately 630,000 samples. In our experiments, we convert RGB images into grayscale images, and normalize them by subtracting the mean and dividing by the standard deviation. In order to simulate the limited local data of each participant, we select 100,000 training samples and 10,000 testing samples. Analogously, in our experiments, each participant owns 1% of the entire training dataset (i.e., 1000 training samples).

### 4.2. Experiments setup

We implement and evaluate our framework in Python using TensorFlow 1.7.0 (Abadi et al., 2016) and a library for the Paillier algorithm [3] on an Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10 GHz server with GPU Tesla P40. We consider two deep learning architectures with high performance on MNIST and SVHN datasets during centralized training, as shown in Fig. 2, where Fig. 2a shows a typical two-layer convolutional neural network (CNN) (LeCun et al., 1998) and Fig. 2b shows a two-layer CNN with local response normalization layers (LRN) (Krizhevsky, Sutskever, & Hinton, 2012).

We evaluate our framework with three metrics: the model utility, the privacy level and the training efficiency. Specifically, the model utility is measured by the classification accuracy of the model on the testing samples. The privacy level is measured by the consumption of the total privacy budget (i.e., a smaller value of total privacy budget provides stronger privacy guarantees). The training efficiency is measured by the number of training epochs required for model convergence and the running time of the Paillier algorithm.

We set up two baselines for the comparison with high noise training and low noise training. The first is the accuracy of the model trained only on one participant's local data (i.e., the standalone training). The other is the accuracy of the model trained on the data centralized from all participants (i.e., $N = 20, 40, 60$). Neither of two baselines applies privacy-preserving operations (i.e., $\theta_{up} = 1$ and no noise), since there is only one data owner in both cases. Then we experimentally evaluate the three strategies for allocating privacy budget discussed in Section 3.2.2, and compare them with the framework of Shokri and Shmatikov (2015) (i.e., allocate fixed privacy budget at each epoch). In our experiments, we vary the number of participants $N$ among 20, 40 and 60. In order to facilitate the statistics of participants' total privacy budget, we adopt a synchronous protocol (i.e., $\tau = N$). The uploaded proportion $\theta_{up}$ is set to 10% to simulate uploading only a small proportion of gradients, reducing the risk of private information leakage. The common objective is the convergence of the global model. The learning process stops when the accuracy of the global model does not rise for three consecutive epochs. Other parameters we set are based on the performance of the local training. For MNIST, we set $\gamma = 10$, mini-batch to 10 and learning rate to $1e^{-3}$. For SVHN, we set $\gamma = 15$, mini-batch to 32 and learning rate to $1e^{-1}$. The parameter analysis for $\theta_{up}$, $\gamma$ and $N$ is set in the last subsection of the experimental evaluation. All the experiments run over 10 random combinations of $N$ sub-datasets from 100 total sub-datasets.

### 4.3. Experiments results

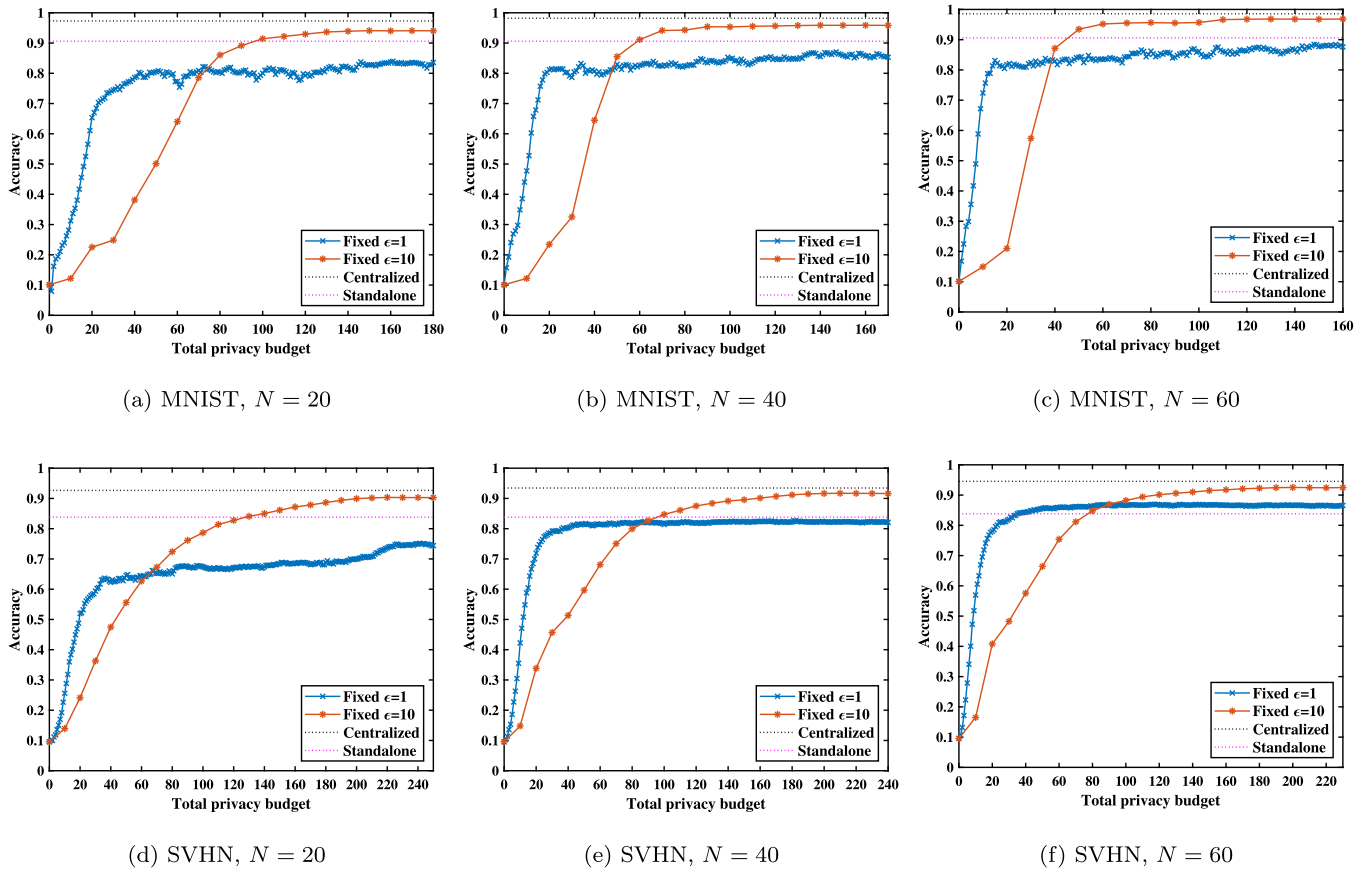
#### 4.3.1. Results for low noise training and high noise training

To illustrate the training characteristics of multi-party deep learning integrated with differential privacy, we compare the low noise training with the high noise training. We consume fixed $\epsilon = 10$ privacy budget per epoch to train the deep learning model, which represents the low noise training. Relatively, we consume fixed $\epsilon = 1$ privacy budget per epoch to train the deep learning model, which represents the high noise training.

Fig. 3 evaluates the accuracy versus total privacy budget of high noise training and low noise training for different datasets and different numbers of participants. By observing that the number of points in the curve of $\epsilon = 10$ is much less than that of $\epsilon = 1$, for the reason that training ten epochs with privacy budget $\epsilon = 1$ per epoch consumes the same total privacy budget as training one epoch with privacy budget $\epsilon = 10$ per epoch, according to the Sequential Composition Theorem discussed in Section 2.3. In the early stages of training (e.g., total privacy budget $\epsilon_{sum} < 30$), the accuracy of high noise training rises much faster than that of low noise training with the same privacy budget consumption. In the later stages of training, the accuracy of high noise training becomes oscillating due to the high noise, and the final accuracy is even lower than that of standalone training. Conversely, the accuracy of low noise training rises steadily, achieving much higher accuracy than standalone training. As the number of participants increased, the accuracy rises faster and achieves a higher level, since more participants contribute their local gradients. For instance, it is shown in Fig. 3f that it can achieve higher accuracy than standalone training even in the case of the high noise training.

In order to reduce the consumption of privacy budget without compromising model utility, we combine the advantages of low noise training and high noise training. High noise training is used in the early stages of training to reduce privacy budget consumption, thereby enhancing privacy guarantees. Low noise training is used in the later stages of training to achieve high accuracy. For example, in the case of Fig. 3a, we can switch the high noise training to the low noise training when $\epsilon_{sum} = 30$. However, this approach is difficult to determine when to switch and extremely increases the time overhead of training. Therefore, we propose more general strategies for allocating privacy budget to gradually increase the privacy budget in a uniform, logarithmic, and exponential manner.

[2] http://ufldl.stanford.edu/housenumbers.
[3] https://github.com/n1analytics/python-paillier.




(a) MNIST, $N = 20$ (b) MNIST, $N = 40$ (c) MNIST, $N = 60$

(d) SVHN, $N = 20$ (e) SVHN, $N = 40$ (f) SVHN, $N = 60$

**Fig. 3.** The accuracy versus total privacy budget of low noise training and high noise training with various numbers of participant $N$ on MNIST and SVHN. The points on the curve represent the result after each epoch.

*4.3.2. Results for strategies for allocating privacy budget*

To demonstrate the effectiveness of our strategies for allocating privacy budget, the framework of Shokri and Shmatikov (2015) that allocates fixed privacy budget at each epoch (i.e., fixed $\epsilon = 10$ per epoch) is set as a baseline for comparison. Three types of strategies for allocating privacy budget have been discussed in Section 3.2.2. Accordingly, $\epsilon_{\max}$ is set to 10 and $\epsilon_{\min}$ is set to 1.

Fig. 4 depicts the accuracy versus total privacy budget for different datasets, different numbers of participants, and different strategies for allocating privacy budget. The accuracy of three strategies rises faster than the accuracy of the baseline with the same total privacy budget consumption. Among them, the strategy in an exponential manner rises fastest. The strategy in a uniform manner rises the second fastest and the strategy in a logarithmic manner rises the slowest. After convergence, three strategies consume less total privacy budget than the baseline and achieve almost the same accuracy as the baseline. As the number of participants increases, it still maintains these rising characteristics. Besides, the accuracy rises faster and eventually converges to a higher level. Moreover, by observing that our strategies are robust to different datasets.

Table 3 details the accuracy, the total privacy budget consumption and the number of training epochs after convergence. From the perspective of accuracy, three strategies can still achieve almost the same accuracy as the low noise training, though more noise is added during the early stages of training, which means that our strategies do not decrease model utility. From the perspective of total privacy budget consumption, the larger the number of participants, the less the total privacy budget each participant consumes, since the contribution of more participants makes the model converge faster. Compared with the baseline, the strategy in an exponential manner economizes the most privacy budget, about 15.81%–21.73%. The strategy in a uniform manner economizes the second most privacy budget, about 13.10%–17.44%. The strategy in a logarithmic manner economizes the least privacy budget, about 6.33%–8.75%. From the perspective of the number of training epochs that corresponds to the training efficiency, compared with the baseline, the strategy in an exponential manner incurs the most time overhead, about 1.28–1.39 times. The strategy in a uniform manner incurs the second most time overhead, about 1.11–1.17 times. The strategy in a logarithmic manner hardly incurs the time overhead, about 1.00–1.06 times.

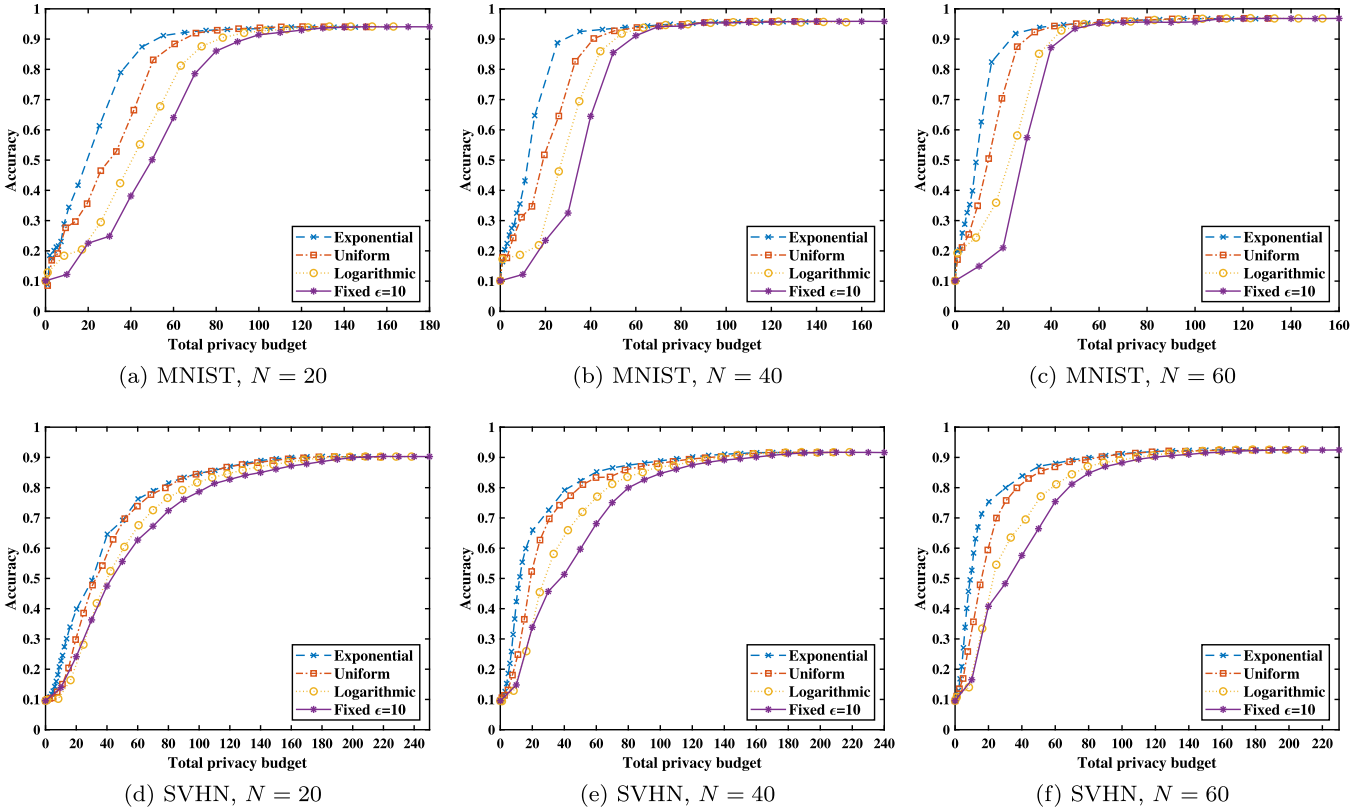
Obviously, we provide an intuitive handle to strike a balance between privacy level and training efficiency by choosing different strategies to allocate privacy budget. Economizing more privacy budget incurs more time overhead, but provides stronger privacy guarantees. For the situation that requires high privacy level, the exponential strategy for allocating privacy budget is a good choice. For the situation with limited computation capacity, we can choose the logarithmic strategy for allocating privacy budget. In general, the uniform strategy for allocating privacy budget is recommended, for the reason that it economizes more privacy budget with less time overhead.

*4.3.3. Results for homomorphic encryption*

To illustrate the time overhead of homomorphic encryption in our framework, we experimentally estimate the running time of the Paillier algorithm on each participant and the central server. For each participant, it is necessary to decrypt all the encrypted global weights downloaded from the central server and encrypt the $\theta_{\text{up}}$ proportion of gradients before uploading to the central

**Table 3**
The average accuracy, total privacy budget and the number of training epochs of different strategies for allocating privacy budget at each epoch with various numbers of participant $N$ on MNIST and SVHN over 10 randomly chosen training sets. The number of training epoch is rounded off.

| Dataset | | MNIST | | | SVHN | | |
|---|---|---|---|---|---|---|---|
| Participants $N$ | | $N = 20$ | $N = 40$ | $N = 60$ | $N = 20$ | $N = 40$ | $N = 60$ |
| Accuracy | Fixed $\epsilon = 10$ (Shokri & Shmatikov, 2015) | 94.09 | 95.87 | 96.83 | **90.35** | 91.70 | 92.53 |
| | Logarithmic (ours) | **94.16** | 95.78 | **96.91** | 90.31 | **91.76** | **92.61** |
| | Uniform (ours) | 94.12 | **95.88** | 96.87 | 90.28 | 91.69 | 92.49 |
| | Exponential (ours) | 93.99 | 95.74 | 96.78 | 90.34 | 91.62 | 92.51 |
| Total privacy budget | Fixed $\epsilon = 10$ (Shokri & Shmatikov, 2015) | 182.00 | 171.00 | 160.00 | 252.00 | 242.00 | 231.00 |
| | Logarithmic (ours) | 166.08 | 156.08 | 147.08 | 233.28 | 222.28 | 211.28 |
| | Uniform (ours) | 153.50 | 142.50 | 132.50 | 219.00 | 209.00 | 198.00 |
| | Exponential (ours) | **148.23** | **136.23** | **125.23** | **212.23** | **201.23** | **190.23** |
| Training epoch | Fixed $\epsilon = 10$ (Shokri & Shmatikov, 2015) | **18** | **17** | **16** | **25** | **24** | **23** |
| | Logarithmic (ours) | **18** | **17** | **16** | **25** | **24** | **23** |
| | Uniform (ours) | 20 | 19 | 18 | 29 | 28 | 27 |
| | Exponential (ours) | 23 | 22 | 21 | 34 | 33 | 32 |




(a) MNIST, $N = 20$ (b) MNIST, $N = 40$ (c) MNIST, $N = 60$

(d) SVHN, $N = 20$ (e) SVHN, $N = 40$ (f) SVHN, $N = 60$

**Fig. 4.** The accuracy versus total privacy budget of different strategies for allocating privacy budget at each epoch with various numbers of participant $N$ on MNIST and SVHN. The points on the curve represent the result after each epoch.

**Table 4**
The running time of the Paillier algorithm with various key sizes on MNIST and SVHN.

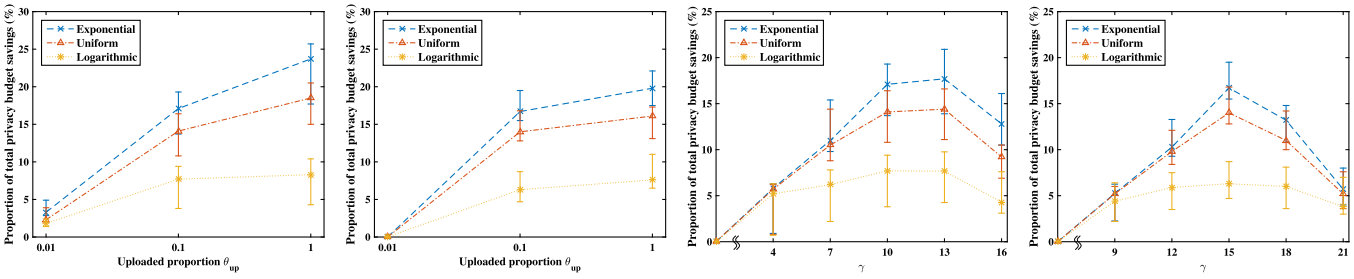
| Dataset | MNIST (encrypt 123 739 gradients, decrypt 1 237 386 weights) | | | SVHN (encrypt 215 649 gradients, decrypt 2 156 490 weights) | | |
|---|---|---|---|---|---|---|
| Key size (bits) | 64 | 128 | 256 | 64 | 128 | 256 |
| Encryption by each participant (s) | 3.41 | 4.26 | 8.83 | 6.08 | 7.76 | 15.64 |
| Decryption by each participant (s) | 12.62 | 15.64 | 33.36 | 20.43 | 26.74 | 55.07 |
| Addition on central server (s) | $0.61\times N$ | $0.64\times N$ | $0.72\times N$ | $1.05\times N$ | $1.16\times N$ | $1.34\times N$ |

server. In our framework, $\theta_{\text{up}}$ is set to 10%. For the central server, it is necessary to perform the homomorphic addition $N$ times to update the encrypted global weights when $N$ participants have uploaded the encrypted local weights. In our experiments, the operations are executed serially, but can be designed to be executed in parallel.

Table 4 evaluates the running time of the Paillier algorithm for various key sizes and different datasets. BY observing that the encryption and decryption time costs are proportional to the key size, while the key size has the least impact on the addition. In addition, the number of network parameters is proportional to the time overhead. Moreover, we can notice that the increase in

**Table 5**
Average accuracy for different strategies, datasets and uploaded proportion $\theta_{up}$ over 10 randomly chosen training sets.

| Dataset | MNIST | | | SVHN | | |
|---|---|---|---|---|---|---|
| Uploaded proportion $\theta_{up}$ | $\theta_{up} = 0.81$ | $\theta_{up} = 0.1$ | $\theta_{up} = 1$ | $\theta_{up} = 0.81$ | $\theta_{up} = 0.1$ | $\theta_{up} = 1$ |
| Fixed $\epsilon = 1$ (Shokri & Shmatikov, 2015) | 62.42 | 83.27 | 91.87 | 30.77 | 74.27 | 85.76 |
| Fixed $\epsilon = 10$ (Shokri & Shmatikov, 2015) | 78.91 | 94.09 | 96.18 | **67.91** | **90.35** | 91.37 |
| Logarithmic (ours) | 79.02 | **94.16** | 96.27 | 67.76 | 90.31 | **91.42** |
| Uniform (ours) | 79.08 | 94.12 | **96.28** | 67.81 | 90.28 | 91.31 |
| Exponential (ours) | **79.27** | 93.99 | 96.15 | 67.78 | 90.34 | 91.22 |




(a) MNIST, $N = 20$, $\gamma = 10$ (b) SVHN, $N = 20$, $\gamma = 10$ (c) MNIST, $N = 20$, $\theta_{up} = 0.1$ (d) SVHN, $N = 20$, $\theta_{up} = 0.1$

**Fig. 5.** Proportion of total privacy budget savings for different strategies, datasets, uploaded proportion $\theta_{up}$, and $\gamma$. The savings are compared to the training with fixed $\epsilon = 10$ per epoch. The errorbar plots depict the average, minimum and maximum values over 10 randomly chosen training sets.

time overhead is primarily decryption and addition, whereas the impact of encryption is minimal, since only a small proportion (i.e., 10%) of gradients are encrypted. For example, in MNIST, we need 1 237 386 decryptions and 1 237 386$*N$ homomorphic additions, but only 123 739 encryptions are required. When $N$ is 60 and key size is 128, for MNIST, it only needs to increase the time overhead of 58.3 s for each epoch, which can be easily accepted compared with training processes.

#### 4.3.4. Parameter analysis

In this subsection, we show the effects of key parameters on the performance of the proposed algorithm, such as $\theta_{up}$ (the uploaded proportion), $\gamma$ (the number of epochs required to reach $\epsilon_{max}$) and $N$ (the number of participants). Table 5 and Figs. 5(a) 5(b) show the effect of $\theta_{up}$ on the accuracy and the proportion of total privacy budget savings compared to the training with fixed $\epsilon = 10$ per epoch. As the $\theta_{up}$ increases, so does the accuracy, since each participant contributes more gradients to the global model. If $\theta_{up}$ is too small, it is even worse than the standalone training. As the $\theta_{up}$ increases, the proportion of total privacy budget savings rises rapidly and then rises slowly, since When $\theta_{up} = 0.01$, our strategies barely economize privacy budget, because the high noise and the small proportional gradients contributed lead to poor training in the early stages (e.g., the accuracy is 62.42% for MNIST and 30.77% for SVHN when $\epsilon = 1$ and $\theta_{up} = 0.01$). With the increase of $\theta_{up}$, the high noise training in the early stages can achieve higher accuracy, enabling more privacy budget savings.

Figs. 5(c) 5(d) show the effect of $\gamma$ on the proportion of total privacy budget savings compared to the training with fixed $\epsilon = 10$ per epoch. As the $\gamma$ increases, the proportion of total privacy budget savings increases first and then decreases. If $\gamma$ is too small, the noise level will quickly return to its original lowest level (i.e. $\epsilon_{max}$), resulting in less privacy budget savings. If $\gamma$ is too large, the number of epochs required to reach the lowest noise level is too much, which in turn increases the total privacy budget. The accuracy is not affected by $\gamma$, since the lowest noise level training will be restored after $\gamma$ epochs.

Table 6 and Fig. 6 show the effect on the accuracy and the proportion of total privacy budget savings under the condition of the same total training sample size for different $N$. Different from the 1% training data per participant in the previous experiments, the total training sample size for different $N$ is the same (i.e., 3%, 1.5%, 1% training data per participant for $N = 20, 40, 60$). As the $N$ increases, the accuracy drops slightly, because the training is harder if the more parts of the training set are divided. When the noise level is lowered, the extent of accuracy degradation is reduced. As the $N$ increases, the proportion of total privacy budget savings increases slightly. That is because more participants make the training converge a little faster, which economizes a little higher proportion of total privacy budget relatively.

## 5. Conclusion and future work

For a multi-party deep learning scenario which is composed of honest-but-curious participants and an honest-but-curious central server, we propose a framework that integrates differential privacy with homomorphic encryption, so that private information is neither leaked among participants nor leaked from the central server. In addition, in order to alleviate the problem of consuming high total privacy budget, three strategies for allocating the privacy budget are presented to further enhance privacy guarantees without sacrificing accuracy. Therefore, it is beneficial for a multi-party deep learning scenario which is in a harsh privacy environment or requires a high privacy level.

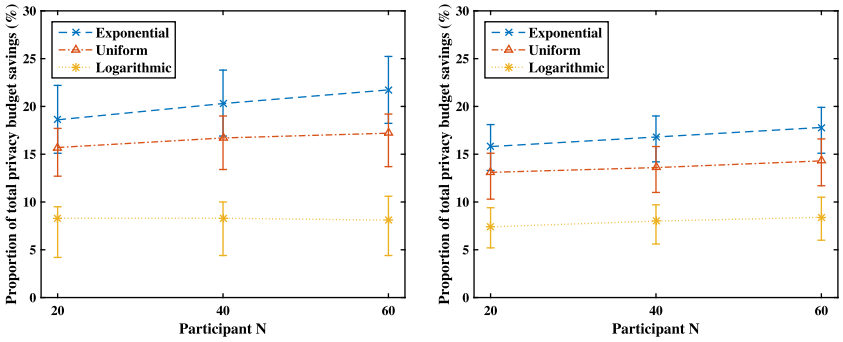
Our framework opens up several directions for further improvement. For instance, instead of the differential privacy techniques based on global sensitivity, the differential privacy techniques based on local sensitivity can be applied to reduce noise without decreasing privacy levels, thereby leading to higher model utility. Besides, instead of the Paillier algorithm, other more efficient homomorphic encryption algorithms can be applied. Moreover, not only the three proposed strategies for allocating the privacy budget, but also other strategies that are more suitable for multi-party deep learning can be applied.

## Acknowledgments

The authors wish to thank the editors and anonymous reviewers for their valuable comments and helpful suggestions which greatly improved the paper's quality. This work was supported by the National key research and development program of China under Grant 2017YFB0802200.

**Table 6**
Average accuracy for different strategies and datasets under the condition of the same total training sample size for different $N$.

| Dataset | MNIST | | | SVHN | | |
|---|---|---|---|---|---|---|
| Participants $N$ | $N = 20$ | $N = 40$ | $N = 60$ | $N = 20$ | $N = 40$ | $N = 60$ |
| Fixed $\epsilon = 1$ (Shokri & Shmatikov, 2015) | 90.58 | 89.28 | 87.74 | 87.24 | 86.64 | 84.46 |
| Fixed $\epsilon = 10$ (Shokri & Shmatikov, 2015) | 96.92 | 96.90 | 96.83 | 92.65 | 92.58 | 92.53 |
| Logarithmic (ours) | 96.97 | **96.93** | **96.91** | 92.71 | **92.65** | **92.61** |
| Uniform (ours) | **96.99** | 96.92 | 96.87 | 92.68 | 92.54 | 92.49 |
| Exponential (ours) | 96.88 | 96.82 | 96.78 | **92.72** | 92.63 | 92.51 |




(a) MNIST, $N = 20$, $\gamma = 10$ (b) SVHN, $N = 20$, $\gamma = 10$

**Fig. 6.** Proportion of total privacy budget savings for different strategies and datasets under the condition of the same total training sample size for different $N$. The savings are compared to the training with fixed $\epsilon = 10$ per epoch. The errorbar plots depict the average, minimum and maximum values over 10 randomly chosen training sets.

## References


Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). Tensorflow: a system for large-scale machine learning. In USENIX symposium on operating systems design and implementation, (pp. 265–283).

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., et al. (2016). Deep learning with differential privacy. In Proceedings of ACM SIGSAC conference on computer and communications security, Vienna, Austria, (pp. 308–318).

Bukovsky, I., & Homma, N. (2017). An approach to stable gradient-descent adaptation of higher order neural units. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(9), 2022–2034.

Camuñas-Mesa, L. A., Serrano-Gotarredona, T., Ieng, S., Benosman, R., & Linares-Barranco, B. (2018). Event-driven stereo visual tracking algorithm to solve object occlusion. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(9), 4223–4237.

Cao, B., Wang, N., Gao, X., & Li, J. (2018). Asymmetric joint learning for heterogeneous face recognition. In: Proceedings of AAAI conference on artificial intelligence. New Orleans, Louisiana, USA, (pp. 6682–6689).

Chang, D., Lin, M., & Zhang, C. (2018). On the generalization ability of online gradient descent algorithm under the quadratic growth condition. *IEEE Transactions on Neural Networks and Learning Systems*.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., et al. (2012). Large scale distributed deep networks. in proceedings of advances in neural information processing systems, Lake Tahoe, Nevada, USA, (pp. 1223–1231).

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Dwork, C. (2008). Differential privacy: A survey of results. In *Conference on Theory & Applications of Models of Computation* (pp. 1–19). Springer.

Dwork, C., McSherry, F., Nissim, K., & Smith, A. (2006). Calibrating noise to sensitivity in private data analysis. In Theory of cryptography conference, New York, NY, USA, (pp. 265–284).

Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, *9*(3–4), 211–407.

Gentry, C., & Boneh, D. (2009). *A fully homomorphic encryption scheme*. Stanford University.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning*. MIT press Cambridge.

Hitaj, B., Ateniese, G., & Perez-Cruz, F. (2017). Deep models under the gan: information leakage from collaborative deep learning. In Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, Dallas, TX, USA, (pp. 603–618).

Huang, K., Dai, D., Ren, C., & Lai, Z. (2017). Learning kernel extended dictionary for face recognition. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(5), 1082–1094.

Jain, P., Kulkarni, V., Thakurta, A., & Williams, O. (2015). To drop or not to drop: Robustness, consistency and differential privacy properties of dropout, arXiv preprint arXiv:1503.02031.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Proceedings of advances in neural information processing systems, Lake Tahoe, Nevada, USA, (pp. 1097–1105).

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lee, J., & Kifer, D. (2018). Concentrated differentially private gradient descent with adaptive per-iteration privacy budget. In *Proceedings of ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 1656–1665). ACM.

Liang, M., & Hu, X. (2015). Recurrent convolutional neural network for object recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, Boston, MA, USA, (pp. 3367–3375).

Mahmud, M., Kaiser, M. S., Hussain, A., & Vassanelli, S. (2018). Applications of deep learning and reinforcement learning to biological data. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(6), 2063–2079.

McSherry, F., & Talwar, K. (2007). Mechanism design via differential privacy. In 48th Annual IEEE symposium on foundations of computer science, Providence, RI, USA, (pp. 94–103).

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In Proceedings of NIPS workshop on deep learning and unsupervised feature learning, Granada, Spain, (pp. 5).

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In Conference on the theory and applications of cryptographic techniques, Prague, Czech Republic, (pp. 223–238).

Pathak, M., Rane, S., & Raj, B. (2010). Multiparty differential privacy via aggregation of locally trained classifiers. In Proceedings of advances in neural information processing systems, Vancouver, British Columbia, Canada, (pp. 1876–1884).

Phong, L., Aono, Y., Hayashi, T., Wang, L., & Moriai, S. (2018). Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, *13*(5), 1333–1345.

Rivest, R. L., Adleman, L., & Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. *Foundations of secure computation*, *4*(11), 169–180.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533.

Shokri, R., & Shmatikov, V. (2015). Privacy-preserving deep learning. In Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, Denver, Colorado, USA, (pp. 1310–1321).

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354.

Wang, J., Xu, C., Yang, X., & Zurada, J. M. (2018). A novel pruning algorithm for smoothing feedforward neural networks based on group lasso method. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(5), 2012–2024.

Zhang, X., Ji, S., Wang, H., & Wang, T. (2017). Private, yet practical, multiparty deep learning. In IEEE 37th international conference on distributed computing systems, Atlanta, GA, USA, (pp. 1442–1452).

Zhang, Y., Li, P., Jin, Y., & Choe, Y. (2015). A digital liquid state machine with biologically inspired learning and its application to speech recognition. *IEEE Transactions on Neural Networks and Learning Systems*, *26*(11), 2635–2649.

Zinkevich, M., Weimer, M., Li, L., & Smola, A. J. (2010). Parallelized stochastic gradient descent. In Proceedings of advances in neural information processing systems, Vancouver, British Columbia, Canada, (pp. 2595–2603).