Privacy-Preserving Deep Learning

Reza Shokri
The University of Texas at Austin
shokri@cs.utexas.edu

Vitaly Shmatikov
Cornell Tech
shmat@cs.cornell.edu

ABSTRACT

Deep learning based on artificial neural networks is a very popular approach to modeling, classifying, and recognizing complex data such as images, speech, and text. The unprecedented accuracy of deep learning methods has turned them into the foundation of new AI-based services on the Internet. Commercial companies that collect user data on a large scale have been the main beneficiaries of this trend since the success of deep learning techniques is directly proportional to the amount of data available for training.

Massive data collection required for deep learning presents obvious privacy issues. Users' personal, highly sensitive data such as photos and voice recordings is kept indefinitely by the companies that collect it. Users can neither delete it, nor restrict the purposes for which it is used. Furthermore, centrally kept data is subject to legal subpoenas and extra-judicial surveillance. Many data owners—for example, medical institutions that may want to apply deep learning methods to clinical records—are prevented by privacy and confidentiality concerns from sharing the data and thus benefitting from large-scale deep learning.

In this paper, we design, implement, and evaluate a practical system that enables multiple parties to jointly learn an accurate neural-network model for a given objective without sharing their input datasets. We exploit the fact that the optimization algorithms used in modern deep learning, namely, those based on stochastic gradient descent, can be parallelized and executed asynchronously. Our system lets participants train independently on their own datasets and selectively share small subsets of their models' key parameters during training. This offers an attractive point in the utility/privacy tradeoff space: participants preserve the privacy of their respective data while still benefitting from other participants' models and thus boosting their learning accuracy beyond what is achievable solely on their own inputs. We demonstrate the accuracy of our privacy-preserving deep learning on benchmark datasets.

Categories and Subject Descriptors

Security and privacy [**Software and application security**]: Domain-specific security and privacy architectures

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813687.

Keywords

Privacy; Neural networks; Deep learning; Gradient Descent

1 Introduction

Recent advances in deep learning methods based on artificial neural networks have led to breakthroughs in long-standing AI tasks such as speech, image, and text recognition, language translation, etc. Companies such as Google, Facebook, and Apple take advantage of the massive amounts of training data collected from their users and the vast computational power of GPU farms to deploy deep learning on a large scale. The unprecedented accuracy of the resulting models allows them to be used as the foundation of many new services and applications, including accurate speech recognition [24] and image recognition that outperforms humans [26].

While the utility of deep learning is undeniable, the same training data that has made it so successful also presents serious privacy issues. Centralized collection of photos, speech, and video from millions of individuals is ripe with privacy risks. First, companies gathering this data keep it forever; users from whom the data was collected can neither delete it, nor control how it will be used, nor influence what will be learned from it. Second, images and voice recordings often contain accidentally captured sensitive items—faces, license plates, computer screens, the sound of other people speaking and ambient noises [44], etc. Third, users' data kept by companies is subject to subpoenas and warrants, as well as warrantless spying by national-security and intelligence outfits.

Furthermore, the Internet giants' monopoly on "big data" collected from millions of users leads to their monopoly on the AI models learned from this data. Users benefit from new services, such as powerful image search, voice-activated personal assistants, and machine translation of webpages in foreign languages, but the underlying models constructed from their collective data remain proprietary to the companies that created them.

Finally, in many domains, most notably those related to medicine, the sharing of data about individuals is not permitted by law or regulation. Consequently, biomedical and clinical researchers can only perform deep learning on the datasets belonging to their own institutions. It is well-known that neural-network models become better as the training datasets grow bigger and more diverse. Due to not being able to use the data from other institutions when training their models, researchers may end up with worse models. For example, data owned by a single organization (e.g., a particular medical clinic) may be very homogeneous, producing an overfitted model that will be inaccurate when used on other inputs. In this case, privacy and confidentiality restrictions significantly reduce utility.

Our contributions. We design, implement, and evaluate a practical system for collaborative deep learning that offers an attractive tradeoff between utility and privacy. Our system enables multiple participants to learn neural-network models on their own inputs, without sharing these inputs but benefitting from other participants who are concurrently learning similar models.

Our key technical innovation is the selective sharing of model parameters during training. This parameter sharing, interleaved with local parameter updates during stochastic gradient descent, allows participants to benefit from other participants' models without explicit sharing of training inputs. Our approach is independent of the specific algorithm used to construct a model for a particular task. Therefore, it can easily accommodate future advances in neural-network training without changing the core protocols.

Selective parameter sharing is effective because stochastic gradient descent algorithms underlying modern neural-network training can be parallelized and run asynchronously. They are robust to unreliable parameter updates, race conditions, participants dropping out, etc. Updating a small fraction of parameters with values obtained from other participants allows each participant to avoid local minima in the process of finding optimal parameters. Parameter sharing can be tuned to control the tradeoff between the amount of information exchanged and the accuracy of the resulting models.

We experimentally evaluate our system on two datasets, MNIST and SVHN, used as benchmarks for image classification algorithms. The accuracy of the models produced by the distributed participants in our system is close to the centralized, privacy-violating case where a single party holds the entire dataset and uses it to train the model. For the MNIST dataset, we obtain 99.14% accuracy (respectively, 98.71%) when participants share 10% (respectively, 1%) of their parameters. By comparison, the maximum accuracy is 99.17% for the centralized, privacy-violating model and 93.16% for the non-collaborative models learned by participants individually. For the SVHN dataset, we achieve 93.12% (89.86%) accuracy when participants share 10% (1%) of their parameters. By comparison, the maximum accuracy is 92.99% for the centralized, privacy-violating model and 81.82% for the non-collaborative models.

Even without additional protections, our system already achieves much stronger privacy, with negligible utility loss, than any existing approach. Instead of directly revealing all training data, the only leakage in our system is indirect, via a small fraction of neural-network parameters. To minimize even this leakage, we show how to apply differential privacy to parameter updates using the sparse vector technique, thus mitigating privacy loss due to both parameter selection (i.e., choosing which parameters to share) and shared parameter values. We then quantitatively measure the tradeoff between accuracy and privacy.

2 Related Work

2.1 Deep learning

Deep learning is the process of learning nonlinear features and functions from complex data. Surveys of deep-learning architectures, algorithms, and applications can be found in [5, 16]. Deep learning has been shown to outperform traditional techniques for speech recognition [23,24,27], image recognition [30,45], and face detection [48]. A deep-learning architecture based on a new type of rectifier activation functions is claimed to outperform humans when recognizing images from the ImageNet dataset [26].

Deep learning has shown promise for analyzing complex biomedical data related to cancer [13, 22, 32] and genetics [15, 56]. The training data used to build these models is especially sensitive from the privacy perspective, underscoring the need for privacy-preserving deep learning methods.

Our work is inspired by recent advances in parallelizing deep learning, in particular parallelizing stochastic gradient descent on GPU/CPU clusters [14], as well as other techniques for distributing computation during neural-network training [1, 39, 59]. These techniques, however, are not concerned with privacy of the training data and all assume that a single entity controls the training.

2.2 Privacy in machine learning

The existing literature on privacy protection in machine learning mostly targets conventional machine learning algorithms, as opposed to deep learning, and addresses three objectives: privacy of the data used for learning a model or as input to an existing model, privacy of the model, and privacy of the model's output.

Techniques based on secure multi-party computation (SMC) can help protect intermediate steps of the computation when multiple parties perform collaborative machine learning on their proprietary inputs. SMC has been used for learning decision trees [33], linear regression functions [17], association rules [50], Naive Bayes classifiers [51], and k-means clustering [28]. In general, SMC techniques impose non-trivial performance overheads and their application to privacy-preserving deep learning remains an open problem.

Techniques that protect privacy of the model include privacy-preserving probabilistic inference [38], privacy-preserving speaker identification [36], and computing on encrypted data [3, 6, 55]. By contrast, our objective is to collaboratively train a neural network that can be used privately and independently by each participant.

Differential privacy [19] is a popular approach to privacy-preserving machine learning. It has been applied to boosting [21], principal component analysis [10], linear and logistic regression [8, 57], support vector machines [41], risk minimization [9, 53], and continuous data processing [43]. Recent results show that a noisy variant of stochastic gradient descent achieves optimal error for minimizing Lipschitz convex functions over ℓ_2-bounded sets [4], and that randomized "dropout," used to prevent overfitting, cal also strengthen the privacy guarantee in a simple 1-layer neural network [29]. To the best of our knowledge, none of the previous work addressed the problem of collaborative deep learning with multiple participants using distributed stochastic gradient descent.

Aggregation of independently trained neural networks using differential privacy and secure multi-party computation is suggested in [37]. Unfortunately, averaging neural-network parameters does not necessarily result in a better model.

Unlike previously proposed techniques, our system achieves all three privacy objectives in the context of collaborative neural-network training: it protects privacy of the training data, enables participants to control the learning objective and how much to reveal about their individual models, and lets them apply the jointly learned model to their own inputs without revealing the inputs or the outputs. Our system achieves this at a much lower performance cost than cryptographic techniques such as secure multi-party computation or homomorphic encryption and is suitable for deployment in modern large-scale deep learning.

3 Deep Learning

Deep learning aims to extract complex features from high-dimensional data and use them to build a model that relates inputs to outputs (e.g., classes). Deep learning architectures are usually constructed as multi-layer networks so that more abstract features are computed as nonlinear functions of lower-level features. We mainly focus on supervised learning, where the training inputs are labeled with correct classes, but in principle our approach can also be used for unsupervised, privacy-preserving learning, too.

Multi-layer neural networks are the most common form of deep learning architectures. Figure 1 shows a typical neural network with two hidden layers. Each node in the network models a neu-

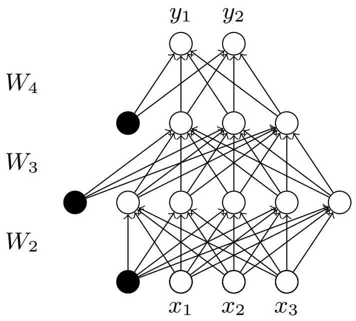

Figure 1: A neural network with two hidden layers. Black circles represent the bias nodes. Matrices W_k contain the weights used in computing the activation functions at each layer k.

ron. In a typical multi-layer network, each neuron receives the output of the neurons in the previous layer plus a bias signal from a special neuron that emits 1. It then computes a weighted average of its inputs, referred to as the total input. The output of the neuron is computed by applying a nonlinear activation function to the total input value. The output vector of neurons in layer k is $\mathbf{a}_k = f(W_k\,\mathbf{a}_{k-1})$, where f is an activation function and W_k is the weight matrix that determines the contribution of each input signal. Examples of activation functions are hyperbolic tangent $f(z) = (e^{2z} - 1)(e^{2z} + 1)^{-1}$, sigmoid $f(z) = (1 + e^{-z})^{-1}$, rectifier $f(z) = max(0, z)$, and softplus $f(z) = log(1 + e^z)$. If the neural network is used to classify input data into a finite number of classes (each represented by a distinct output neuron), the activation function in the last layer is usually a softmax function $f(z_j) = e^{z_j} \cdot (\sum_k e^{z_k})^{-1}, \forall j$. In this case, the output of each neuron j in the last layer is the relative score or probability that the input belongs to class j.

In general, the values computed in higher layers represent more abstract features of the data. The first layer is composed of the raw features extracted from the data, e.g., the intensity of colors in each pixel in an image or the frequency of each word in a document. The outputs of the last layer correspond to the abstract answers produced by the model. If the neural network is used for classification, these abstract features also represent the relation between input and output. The nonlinear function f and the weight matrices determine the features that are extracted at each layer. The main challenge in deep learning is to automatically learn from training data the values of the parameters (weight matrices) that maximize the objective of the neural network (e.g., classification accuracy).

Learning network parameters using gradient descent. Learning the parameters of a neural network is a nonlinear optimization problem. In supervised learning, the *objective function* is the output of the neural network. The algorithms that are used to solve this problem are typically variants of *gradient descent* [2]. Simply put, gradient descent starts at a random point (set of parameters for the neural network), then, at each step, computes the gradient of the nonlinear function being optimized and updates the parameters so as to decrease the gradient. This process continues until the algorithm converges to a local optimum.

In a neural network, the gradient of each weight parameter is computed through *feed-forward* and *back-propagation* procedures. Feed-forward sequentially computes the output of the network given the input data and then calculates the error, i.e., the difference between this output and the true value of the function. Back-propagation propagates this error back through the network and computes the contribution of each neuron to the total error. The gradients of individual parameters are computed from the neurons' activation values and their contribution to the error.

Stochastic gradient descent (SGD). The gradients of the parameters can be averaged over all available data. This algorithm, known as *batch* gradient descent, is not efficient, especially if learning on a large dataset. *Stochastic* gradient descent (SGD) is a drastic simplification which computes the gradient over an extremely small subset (mini-batch) of the whole dataset [58]. In the simplest case, corresponding to maximum stochasticity, one data sample is selected at random in each optimization step.

Let $\mathbf{w}$ be the flattened vector of all parameters in a neural network, composed of $W_k, \forall k$. Let E be the *error function*, i.e., the difference between the true value of the objective function and the computed output of the network. E can be based on L^2 norm or cross entropy [34]. The back-propagation algorithm computes the partial derivative of E with respect to each parameter in $\mathbf{w}$ and updates the parameter so as to reduce its gradient. The update rule of stochastic gradient descent for a parameter w_j is

$$w_j := w_j - \alpha \frac{\partial E_i}{\partial w_j} \quad (1)$$

where α is the *learning rate* and E_i is computed over the mini-batch i. We refer to one full iteration over all available input data as an *epoch*.

Note that each parameter in vector $\mathbf{w}$ is updated independently from other parameters. We will rely on this property when designing our system for privacy-preserving, collaborative stochastic gradient descent in the rest of this paper. Some techniques set the learning rate adaptively [18] but still preserve this independence.

4 Distributed Selective SGD

The core of our approach is a distributed, collaborative deep learning protocol that relies upon the following observations: (i) updates to different parameters during gradient descent are inherently independent, (ii) different training datasets contribute to different parameters, and (iii) different features do not contribute equally to the objective function. Our *Selective Stochastic Gradient Descent (Selective SGD or SSGD)* protocol achieves comparable accuracy to conventional SGD but involves updating 1 or even 2 orders of magnitude fewer parameters in each learning iteration.

Selective parameter update. The main intuition behind selective parameter update is that during SGD, some parameters contribute much more to the neural network's objective function and thus undergo much bigger updates during a given iteration of training. The gradient value depends on the training sample (mini-batch) and varies from one sample to another. Moreover, some features of the input data are more important than others, and the parameters that help compute these features are more crucial in the process of learning and undergo bigger changes.

In selective SGD, the learner chooses a fraction of parameters to be updated at each iteration. This selection can be completely random, but a smart strategy is to select the parameters whose current values are farther away from their local optima, i.e., those that have a larger gradient. For each training sample i, compute the partial derivative $\frac{\partial E_i}{\partial w_j}$ for all parameters w_j as in SGD. Let S be the indices of θ parameters with the largest $\frac{\partial E_i}{\partial w_j}$ values. Finally, update the parameter vector $\mathbf{w}_S$ in the same way as in (1), so the parameters not in S remain unchanged. We refer to the ratio of θ over the total number of parameters as the *parameter selection rate*.

Distributed collaborative learning. Distributed selective SGD assumes two or more participants training independently and concurrently. After each round of local training, participants asyn-

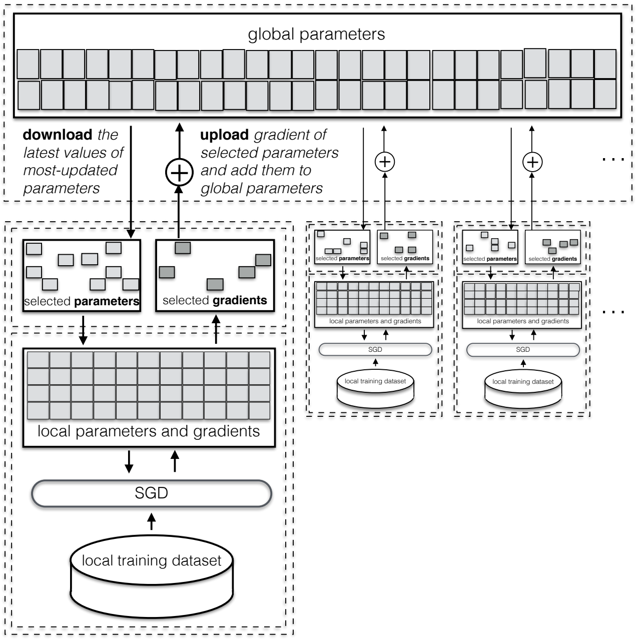
global parameters
download *the latest values of most-updated parameters*
upload *gradient of selected parameters and add them to global parameters*
selected **parameters**
selected **gradients**
local parameters and gradients
SGD
local training dataset
. . .

Figure 2: High-level architecture of our deep learning system. An abstract model of the parameter server, which maintains global values for the parameters, is depicted at the top.

chronously share with each other the gradients they computed for some of the parameters. Each participant fully controls which gradients to share and how often. The sum of all gradients computed for a given parameter determines the magnitude of the global descent towards the parameter's local optima ("local" here refers to the space of parameter values and does not mean being limited to a single participant). Participants thus benefit from each other's training data—without actually seeing this data!—and produce much more accurate models that they would have been able to learn in isolation, limited to their own training data.

Participants can exchange gradients directly, or via a trusted central server, or even use secure multi-party computation to exchange them "obliviously," emulating the functionality of a trusted server that hides the origin of each update. For the purposes of this discussion, we assume an abstraction of a central server to which participants asynchronously upload the gradients. The server adds all gradients to the value of the corresponding parameter. Each participant downloads a subset of the parameters from the server and uses them to update his local model. The download criterion for a given parameter can be the frequency or recency of updates or the moving average of gradients added to that parameter.

5 System Architecture

5.1 Overview

Figure 2 illustrates the main components and protocols of our collaborative deep learning system. We assume that there are N participants, each of which has a local private dataset available for training. All participants agree in advance on a common network architecture and common learning objective. We assume the existence of a parameter server which is responsible for maintaining the latest values of parameters available to all parties. This parameter server is an abstraction, which can be implemented by an actual server or emulated by a distributed system.

α	Learning rate of stochastic gradient descent
θ_d, θ_u	Fraction of parameters selected for download and upload
γ	Bound on gradient values shared with other participants
τ	Threshold for gradient selection

Table 1: List of meta-parameters

```
Choose initial parameters $\mathbf{w}^{(i)}$ and learning rate $\alpha$.
Repeat until an approximate minimum is obtained:
    1. Download $\theta_d \times |\mathbf{w}^{(i)}|$ parameters from server and replace the
       corresponding local parameters.
    2. Run SGD on the local dataset and update the local parameters
       $\mathbf{w}^{(i)}$ according to (1).
    3. Compute gradient vector $\Delta\mathbf{w}^{(i)}$ which is the vector of
       changes in all local parameters due to SGD.
    4. Upload $\Delta\mathbf{w}_S^{(i)}$ to the parameter server, where $S$ is the set
       of indices of at most $\theta_u \times |\mathbf{w}^{(i)}|$ gradients that are selected
       according to one of the following criteria:
         • largest values: Sort gradients in $\Delta\mathbf{w}^{(i)}$ and upload $\theta_u$
           fraction of them, starting from the biggest.
         • random with threshold: Randomly subsample the gra-
           dients whose value is above threshold $\tau$.
       The selection criterion is fixed for the entire training.
```

Figure 3: Pseudocode of DSSGD for participant i.

Each participant initializes the parameters and then runs the training on his own dataset. The system includes a *parameter exchange protocol* that enables participants to upload the gradients of selected neural-network parameters to the parameter server and download the latest parameter values at each local SGD epoch. This allows participants to (i) independently converge to a set of parameters and, critically, (ii) avoid overfitting these parameters to a single participant's local training dataset. Once the network is trained, each participant can independently and privately evaluate it on new data, without interacting with other participants.

In the following, we describe all components of our system in detail. Table 1 lists the *meta-parameters* of our system. These parameters control the collaborative learning process, as opposed to the actual neural-network parameters that are being learned.

5.2 Local training

We assume that each participant i maintains a local vector of neural-network parameters, $\mathbf{w}^{(i)}$. The parameter server maintains a separate parameter vector $\mathbf{w}^{(global)}$. Each participant can initialize his local parameters randomly or by downloading their latest values from the parameter server.

Each participant then trains the neural network using the standard SGD algorithm, iterating over his local training data over many epochs. There need not be any coordination between different participants during their local training. They influence each other's training indirectly, via the parameter server.

Figure 3 presents the pseudocode of the distributed selective SGD (DSSGD) algorithm. DSSGD is run independently by every participant and consists of five steps in each learning epoch. First, the participant downloads a θ_d fraction of parameters from the server and overwrites his local parameters with the downloaded values. He then runs one epoch of SGD training on his local dataset. This training can be done on a sequence of mini-batches; a mini-batch is the set of randomly chosen training data points of size M.

In the third step, the participant computes $\Delta\mathbf{w}^{(i)}$, the vector of changes in all parameters in step 2, i.e., for every parameter j, the old $w_j^{(i)}$ value is subtracted from the new $w_j^{(i)}$ value after the latest epoch of local SGD. We refer to $\Delta w_j^{(i)}$ as the gradient of parameter j over one epoch of local SGD.[1] $\Delta\mathbf{w}^{(i)}$ values reflect how much each parameter has to change to more accurately model the local dataset of the ith participant. This information is exactly what other participants need to incorporate in order to avoid overfitting.

There are several ways to choose which gradients to share at the end of each local epoch. Participants need to agree on the criterion and use it consistently throughout DSSGD. We assume that at most θ_u fraction of parameters can be selected for upload at each epoch.

We consider two selection criteria. The first method is to select exactly θ_u fraction of values, picking big values that significantly contribute to the gradient descent algorithm. The other method is to select a random subset of values that are larger than threshold τ. Since the number of gradients that are greater than τ may be smaller than the θ_u fraction of parameters, fewer gradients will be shared. This might slow down convergence but this selection criterion is closer to the sparse vector technique that we use when extending our system with differential privacy (see Section 7.2).

Before uploading the selected gradients $\Delta\mathbf{w}^{(i)}$, their values are truncated into the $[-\gamma, \gamma]$ range. To prevent these values from leaking too much information about the training data, random noise can also be added as described in Section 7. In short, the participant updates $\Delta\mathbf{w}^{(i)}$ with $\mathsf{bound}(\Delta\mathbf{w}^{(i)}, \gamma)$ and adds some random noise to it before uploading it. In Section 7, we explain how to set the range and randomness parameters and discuss their effect on SGD.

5.3 Parameter server

The parameter server initializes the parameter vector $\mathbf{w}^{(global)}$ and then handles the participants' upload and download requests. Figure 4 shows the server's pseudocode. When someone uploads gradients, the server adds the uploaded $\Delta\mathbf{w}_j$ value to the corresponding global parameters and updates the meta-data and the update counter $stat_j$ for each parameter j. To increase the weight of more recently updated parameters, the server can periodically multiply the counter by a decay factor β, i.e., $\mathbf{stat} := \beta \cdot \mathbf{stat}$. These statistics are used during download, when participants obtain from the server the latest values of the parameters with the largest $stat$ values. Each participant decides what fraction of these parameters to download by setting θ_d.

5.4 Why distributed selective SGD works

Our distributed SSGD achieves achieving almost the same accuracy as conventional, privacy-violating SGD for the same reason why SGD is successful in general: *stochasticity* of the learning process. Updating local parameters with a subset of global parameters during training increases the stochasticity of local SGD. This plays an essential role in preventing local SGD from overfitting to its small local dataset. When training alone, each participant is susceptible to falling into local optima. Overwriting locally learned parameters with values learned by other participants, who train on different datasets, helps each participant escape local optima and enables them to explore other values, resulting in more accurate models.

Our distributed SSGD does not make any assumptions about which parameters need to be updated by other participants, nor about the update rate. Some participants may undergo a higher number of updates, due to better computation and throughput capabilities. Some participants may fail to upload their selected parameters due to network errors or other failures. They may also overwrite each other's updates due to asynchronous access to the parameter server. Not only do race conditions not cripple our distributed SSGD, in fact they contribute to its success by increasing stochasticity. Stochasticity due to asynchronous parameter update is known to be effective for training accurate deep neural networks [14]. This is also consistent with regularizing techniques that randomly corrupt neurons [47] or input data [52] during training in order to avoid overfitting.

```
Choose initial global parameters $\mathbf{w}^{(global)}$.
Set vector $\mathbf{stat}$ to all zero.
EVENT: A participant uploads gradients $\Delta\mathbf{w}_S$.
    • For all $j \in S$:
        – Set $\mathbf{w}^{(global)} := \mathbf{w}^{(global)} + \Delta\mathbf{w}_j$
        – Set $stat_j := stat_j + 1$
EVENT: A participant downloads $\theta$ parameters.
    • Sort $\mathbf{stat}$, and let $I_\theta$ be the set of indices for $\mathbf{stat}$ elements
      with largest values.
    • Send $\mathbf{w}_{I_\theta}^{(global)}$ to the participant.
```

Figure 4: Pseudocode of DSSGD on the server.

5.5 Parameter exchange protocol

DSSGD does not assume that participants follow any particular schedule when uploading their parameters. In our evaluation, we considered the following scenarios.

With *round robin*, participants run SSGD sequentially. Each downloads a fraction of the most updated parameters from the server, runs local training, and uploads selected gradients; the next participant follows in fixed order. With *random order*, participants download, learn, and upload in random order, but access to the server is atomic, i.e., participants lock it before reading and release the lock after writing. With *asynchronous*, participants do not coordinate. While one participant is training on a set of parameters, others may update them on the server before training finishes.

6 Evaluation

6.1 Datasets and learning objectives

We evaluate our system on two major datasets used as benchmarks in the deep-learning literature. The first is the MNIST dataset [31] of handwritten digits formatted as 32x32 images, normalized so that the digits are located at the center of the image. The dataset[2] is composed of 60,000 training examples and 10,000 test examples.

The second is the SVHN dataset [35] of house numbers obtained from Google's street view images. The images are 32x32, with 3 floating point numbers containing the RGB color information of each pixel (that we convert to YUV). Each image is centered around a digit, but many of the images contain some distractors at the sides. The dataset[3] contains 600,000 training images, from which we use 100,000 for training and 10,000 as test examples. Table 2 summarizes how many training and test examples we use.

We normalize the datasets by subtracting the average and dividing by the standard deviation of data samples in their training sets. The size of the input layer of neural networks for MNIST and

[1] Usually *gradient* refers to the change in a parameter after a single mini-batch training, but here we generalize it to one epoch of training over several mini-batches.

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

[3] http://ufldl.stanford.edu/housenumbers

```
nn.Sequential {
  [input -> (1) -> ... -> (7) -> output]
  (1): nn.Reshape(1024)
  (2): nn.Linear(1024 -> 128)
  (3): nn.ReLU
  (4): nn.Linear(128 -> 64)
  (5): nn.ReLU
  (6): nn.Linear(64 -> 10)
  (7): nn.LogSoftMax
}
```

Figure 5: MLP architecture used for MNIST (and for SVHN, with 3072 inputs instead of 1024) in Torch7 nn

```
nn.Sequential {
  [input -> (1) -> ... -> (11) -> output]
  (1): nn.SpatialConvolutionMM
  (2): nn.Tanh
  (3): nn.SpatialMaxPooling
  (4): nn.SpatialConvolutionMM
  (5): nn.Tanh
  (6): nn.SpatialMaxPooling
  (7): nn.Reshape(256)
  (8): nn.Linear(256 -> 200)
  (9): nn.Tanh
  (10): nn.Linear(200 -> 10)
  (11): nn.LogSoftMax
}
```

Figure 6: CNN architecture used for MNIST in Torch7 nn

```
nn.Sequential {
  [input -> (1) -> ... -> (13)-> output]
  (1): nn.SpatialConvolutionMM
  (2): nn.Tanh
  (3): nn.Sequential {
    [input -> (1) -> (2) -> (3)-> output]
    (1): nn.Square
    (2): nn.SpatialAveragePooling
    (3): nn.Sqrt
  }
  (4): nn.SpatialSubtractiveNormalization
  (5): nn.SpatialConvolutionMM
  (6): nn.Tanh
  (7): nn.Sequential {
    [input -> (1) -> (2) -> (3)-> output]
    (1): nn.Square
    (2): nn.SpatialAveragePooling
    (3): nn.Sqrt
  }
  (8): nn.SpatialSubtractiveNormalization
  (9): nn.Reshape(1600)
  (10): nn.Linear(1600 -> 128)
  (11): nn.Tanh
  (12): nn.Linear(128 -> 10)
  (13): nn.LogSoftMax
}
```

Figure 7: CNN architecture used for SVHN in Torch7 nn

SVHN are 1024 and 3072, respectively. The learning objective is to classify the input as one of 10 possible digits, thus the size of the output layer is 10.

6.2 Computing framework

We use Torch7 [11, 49] and Torch7 nn packages.[4] This popular deep-learning library has been used and extended by major Internet companies such as Facebook.[5]

	MNIST	SVHN
train	60,000	100,000
test	10,000	10,000

Table 2: Size of training and test datasets

	MNIST	SVHN
MLP	140,106	402,250
CNN	105,506	313,546

Table 3: Number of neural-network parameters

6.3 Neural network architectures

We use two popular neural network architectures: multi-layer perceptron (MLP) and convolutional neural network (CNN). MLPs are feed-forward neural network architectures in which neurons in each layer are fully connected to the neurons in the next layer. The backpropagation algorithm was initially proposed to learn the parameters of these networks [42]. Figure 1 is an example of an MLP network. CNNs are a special kind of multi-layer neural networks with sparse connectivity [31]. CNNs are widely used for image and video recognition . We provide the exact specifications of our network architectures in Figure 5 (MLP) and Figures 6 and 7 (CNN), all printed using Torch7 nn package. The figures show the activation function used in each layer (e.g., Tanh for tangent hyperbolic, and ReLU for rectifier function), and the connection between layers. Table 3 summarizes the number of parameters.

6.4 Experimental setup

We implemented distributed SSGD with three different parameter exchange protocols—round robin, random order, and asynchronous. The performance of random order was very similar to round robin and thus omitted. We compared all results with two baseline scenarios. The first is *Centralized SGD on the entire dataset*. This is a privacy-violating scenario where all the training data is pooled into one dataset and the network is trained on this dataset using standard stochastic gradient descent. The other scenario is *Standalone SGD*. This is the scenario where participants train solely on their own training data, without any collaboration.

We implemented two criteria for selecting which gradients to upload to the parameter server. With *largest values*, each participant uploads the gradients with the biggest absolute values from the last local training epoch. With *random with threshold*, the participant uploads a random sample of gradients whose values are over a threshold. For *download*, each participant selects the parameters that have undergone the most updates. Other selection criteria, e.g., downloading the parameters that have undergone the biggest change, are also feasible.

In all experiments, the decay factor β for parameter statistics (see Section 5.3) was set to 0.8. We evaluate several settings for the mini-batch size (1 and 32) and for the SGD learning rate[6] ($\alpha = 0.001$ and 0.01) with decay rate $1e^{-7}$. We also vary the number of participants N in each DSSGD scenario between 30, 90, and 150.

We randomly initialize the local training dataset of each participant with 1% of the entire dataset, i.e., 600 data samples for the MNIST scenario and 1000 data samples for the SVHN scenario. The fraction θ_u of parameters selected for sharing in SSGD takes values in $\{1, 0.1, 0.01, 0.001\}$, i.e., $\{15, 141, 1402, 140106\}$ parameters in the case of training an MLP on MNIST (see Table 3). The fraction θ_d of parameters to be downloaded is usually set to 1.

[4] https://github.com/torch/nn

[5] https://github.com/facebook/fblualib

[6] The learning rate and its decay rate are applied during local SGD when training over a new mini-batch. The parameter server does not apply it to the uploaded gradients.

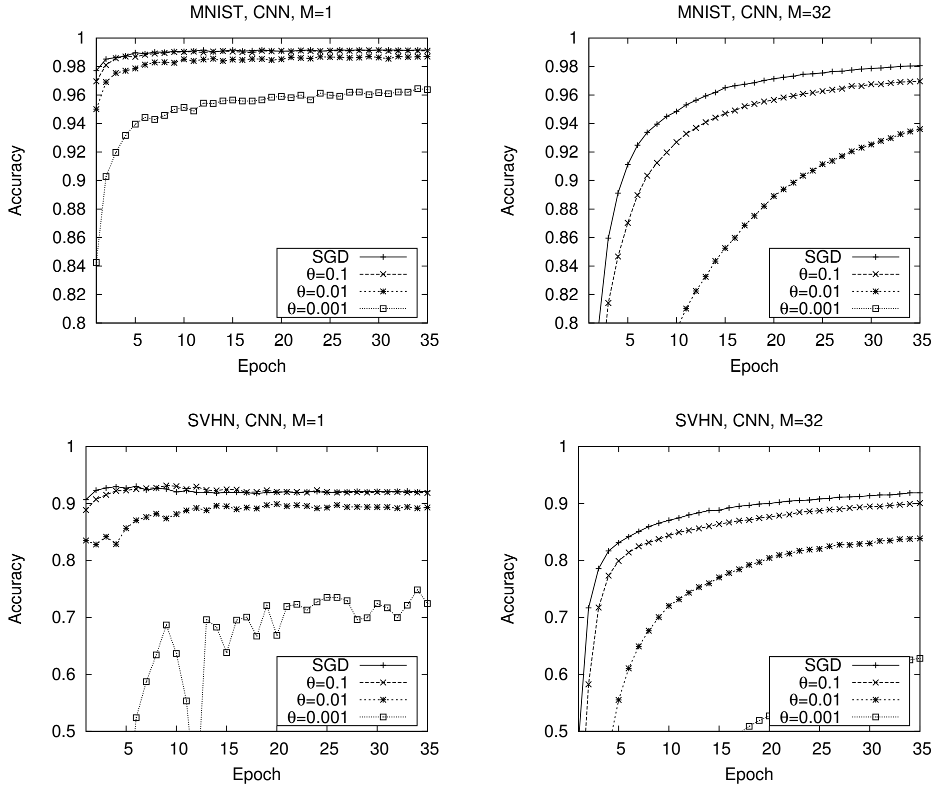

Figure 8: Convergence of SSGD for different mini-batch sizes. The legends show the fraction of parameters selected for sharing at each gradient descent step (with 1, SSGD is equivalent to SGD).

	SGD	0.1	0.01	0.001	Standalone
MNIST, CNN	0.9917	0.9914	0.9871	0.9645	0.9316
SVHN, CNN	0.9299	0.9312	0.8986	0.7481	0.8182

	SGD	0.1	0.01	0.001	Standalone
MNIST, MLP	0.9810	0.98	0.9707	0.9171	0.8832
SVHN, MLP	0.8476	0.8394	0.7833	0.6542	0.5136

Table 4: Maximum accuracy achieved by SSGD for CNN and MLP network architectures and different parameter sharing rates. The results are compared with standalone accuracy. Mini-batch size is 1.

6.5 Results for selective SGD

To show the effectiveness of our approach compared to conventional stochastic gradient descent, we evaluate the accuracy of SSGD and SGD when training a convolutional neural network (CNN) on the MNIST and SVHN datasets. Figure 8 compares SGD and SSGD for different values of meta-parameters (mini-batch size and the fraction of shared gradients). In general, participants can choose the values for the meta-parameters by training on a calibration dataset, e.g., a public dataset that has no privacy implications.

These results confirm the intuition behind SSGD: by sharing only a small fraction of gradients at each gradient descent step, we can achieve almost the same accuracy as SGD. Furthermore, the overall behavior of SGD with and without selective parameter sharing is similar. Setting mini-batch size to 1 achieves high stochasticity throughout the training process and converges very quickly, but also causes fluctuation in some curves. Figure 8 shows accuracy trajectories up to epoch 35; beyond this, we can potentially achieve slightly higher accuracy as shown in Table 4. SSGD can achieve even higher accuracy than SGD because updating only a small fraction of parameters at each epoch acts as a regularization technique which avoids overfitting by preventing the neural network weights from jointly "remembering" the training data (this concept is described in [47]). When mini-batch size is set to 32, convergence is slower but smoother, due to applying the average of gradients over many training data points during gradient descent.

6.6 Results for distributed selective SGD

Figure 9 presents the best accuracy we obtain when running DSSGD on MNIST and SVHN for different neural network architectures, parameter exchange protocols, and fractions of shared parameters. The x-axis is the fraction of shared parameters (θ_u); the y-axis is the accuracy, i.e., the fraction of correctly classified data samples in the test set. We set the download rate θ_d to 1, the learning rate α to 0.001, and mini-batch size to 32.

In each plot, we show the best accuracy for centralized (maximum utility, minimum privacy) and standalone (minimum utility, maximum privacy) SGD. Both are independent of the x-axis since there is no parameter sharing in either. These two scenarios are our baselines. Comparing the accuracy of distributed SSGD with the baselines reflects the tradeoff between utility and privacy. This gap depends on the network architecture and reflects that CNN takes more advantage the training data vs. MLP. Moreover, in our setting, the gap is affected by the complexity of classification and the fact each participant has 1000 data samples in the case of SVHN dataset and 600 data samples in the case of MNIST dataset.

Our results show that *any* cooperation, even when sharing only 1 percent of parameters, results in higher accuracy than standalone learning. Distributed SSGD using round robin parameter exchange results in the highest accuracy, almost equal to centralized SGD. The reason is its similarity to SSGD (see Figure 8). The price paid

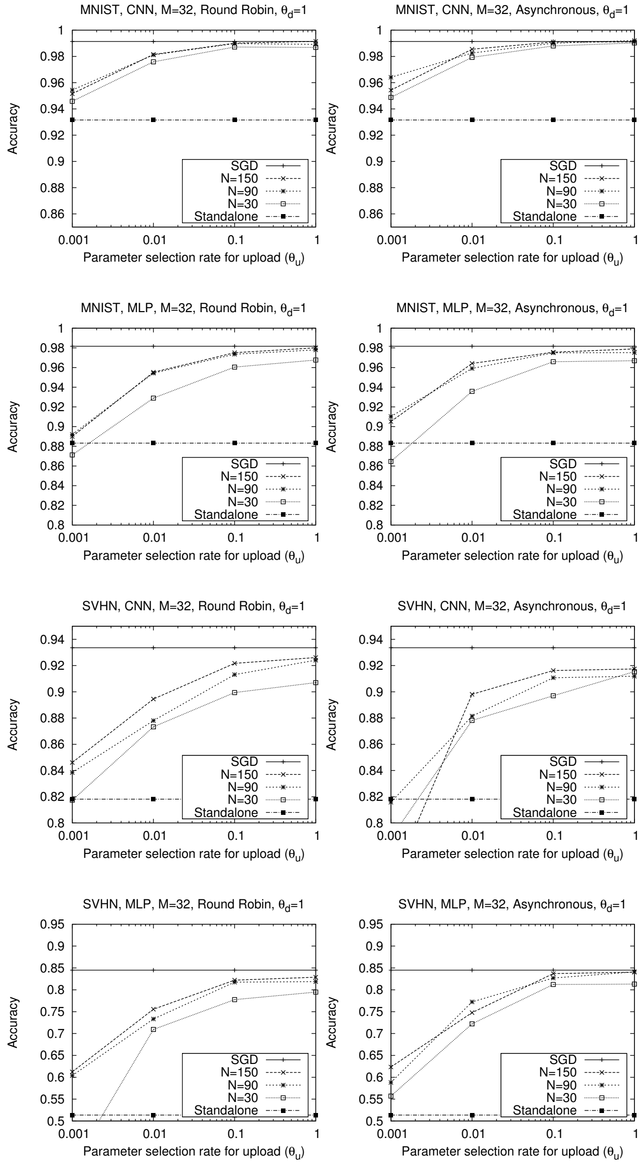

Figure 9: Accuracy of distributed SSGD on the MNIST and SVHN datasets. The legends show the number of participants. "Standalone" means that each participants trains independently on his own data; "SGD" means all training data is pooled for centralized training.

for this accuracy is the speed of learning, which is determined by the slowest participant. The round robin protocol is suitable for scenarios where all participants have similar computation capacity, e.g., biomedical research institutions with dedicated SSGD servers. We do not make any assumptions, however, about how local SGD should run. For example, it can be executed on parallel GPUs to speed up the process. Asynchronous parameter exchange protocol can produce accurate models, too. The key to its success is the inherent randomness and thus high stochasticity of gradient descent, which prevents overfitting. In our implementation, we assumed that each participant may lag behind others and download an outdated set of parameters (those from the previous epoch) with probability 0.5. The promising accuracy of this protocol indicates that DSSGD should work well even with unreliable (e.g., mobile) networks.

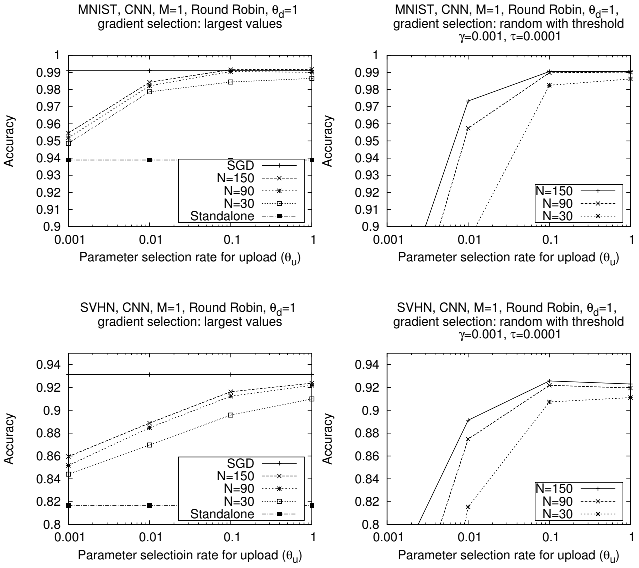

Figure 10: Accuracy of DSSGD for different gradient selection criteria.

We also observe that the number of participants has a lower impact on accuracy than the percentage of shared parameters. This indicates that distributed SSGD does not require very many participants to boost the accuracy.

The number of participants, the rate of parameter updates, and the parameter exchange schedule all influence the communication cost of distributed SSGD. For example, training an MLP model on MNIST dataset with 90 participants with the parameter upload rate of 10% in round-robin schedule requires the server to support $90 \times 14010 \times 32 = 38.5$ Megabytes of parameter uploads during each epoch. With the parameter download rate of 100%, the server needs to support 385 Megabytes of download during each epoch.

All of the above results were obtained assuming each participant shares his largest gradients with the other participants. The other method is to randomly sample from the gradients whose values are above a threshold. Figure 10 compares the accuracy of DSSGD with these two criteria for both MNIST and SVHN datasets. In the "random with threshold" scenario, we first truncate gradient values $\Delta\mathbf{w}$ into the $[-0.001, 0.001]$ range, then go through them in random order, and upload if $\text{abs}(\Delta w_j) \geq \tau$. The neural network architecture (CNN), learning rate ($\alpha = 0.001$), mini-batch size ($M = 1$), and exchange protocol (Round Robin) are the same in all experiments. In the "random with threshold" scenario, fewer than the θ_u fraction of gradients may be uploaded, thus accuracy is sometimes lower. To find an effective value of the threshold τ, participants need to run DSSGD on a public calibration dataset.

Figure 11 shows the convergence of DSSGD for different datasets, learning rates, and number of participants. The upload rate θ_u is 0.1, download rate θ_d is 1, mini-batch size is 32, the parameter exchange protocol is round robin, and the gradient selection criterion is the largest values. These results show that higher learning rate indeed results in faster convergence to maximum accuracy regardless of the number of participants. Therefore, the distributed and selective nature of DSSGD does not change the overall behavior of the gradient descent algorithm.

7 Privacy

Our system aims to address several privacy threats associated with deep learning. First, in conventional deep learning, all training data

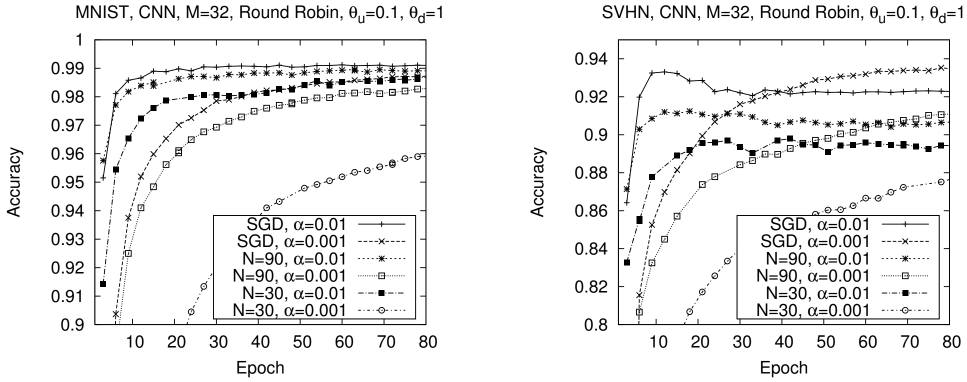

Figure 11: Convergence of DSSGD. The legends show the number of participants N and the learning rate α.

is revealed to a third party (typically, the company performing the learning) and individuals who contributed the data do not have any control over it. Their sensitive information may leak to the company itself, to attackers who compromised the company's data storage, and to law enforcement and intelligence outfits who can access the data via legal and extra-legal means.

Second, in conventional deep learning, data owners have no control over the learning objective (i.e., which model is being trained) and thus no control or even knowledge of what is being inferred from their data. For example, an individual might be willing to share her image for face recognition but not for inferring her location from the background objects.

Third, in conventional deep learning, the learned model is not available directly to data owners. If they want to use it, they must reveal their inputs to the company holding the model, thus exposing them to the same privacy risks as the training data.

Our privacy-preserving deep learning system addresses all of these concerns and aims to protect privacy of the training data, ensure public knowledge of the learning objective, and protect privacy of the data to which the learned model is applied, as well as privacy of the model's output.

The scenarios we consider—for example, collaborative learning of image recognition models between medical institutions—involve participants who are not actively malicious. Therefore, it is reasonable to assume a "passive" adversary model, in which the participants execute the protocol as designed but may attempt to learn or infer sensitive information from other participants' data.

7.1 Preventing direct leakage

While training the model. Unlike conventional deep learning, in our system participants do not reveal their training datasets to anyone, thus ensuring strong privacy of their data. The size and dynamics of local datasets are confidential, and different data samples can be used in each round of SSGD. The participants can also delete their training data at any time.

While using the model. All participants learn the model and thus can use it locally and privately, without any communication with other participants and without revealing the input data or the model's output to anyone. Therefore, in contrast to conventional deep learning, there is absolutely no leakage while using the model.

7.2 Preventing indirect leakage

Participants in our system may indirectly reveal some information about their training datasets via public updates to a fraction of the neural-network parameters during training. Each participant fully controls which gradients to share and may decide not to share particularly sensitive ones. Furthermore, each participant shares only a tiny fraction of his gradients: as we show, even sharing as few as 1% still results in significantly better accuracy than learning just on local data. Even so, we use differential privacy to ensure that parameter updates do not leak too much information about any individual point in the training dataset.

Differential privacy. Our application of differential privacy to parameter updates is inspired by recent work on privacy-preserving empirical risk minimization [4]. In a nutshell, a computation is differentially private if the probability of producing a given output does not depend very much on whether a particular data point is included in the input dataset [19]. For any two datasets D and D' differing in a single item and any output O of function f,

$$\Pr\{f(D) \in O\} \leq exp(\epsilon) \cdot \Pr\{f(D') \in O\}. \qquad (2)$$

The parameter ϵ controls the tradeoff between the accuracy of the differentially private f and how much information it leaks.

In our case, f computes parameter gradients and selects which of them to share with other participants. There are two sources of potential leakage: how gradients are selected for sharing and the actual values of the shared gradients. To mitigate both types of leakage, we use the **sparse vector technique** [20, 25] to (i) randomly select a small subset of gradients whose values are above a threshold, and to (ii) share perturbed values of the selected gradients, all under a consistent differentially private mechanism. This is equivalent to releasing the responses to queries whose value is above a publicly known threshold.

Let the total privacy budget for each epoch of DSSGD for each participant i be ϵ. We split this budget into c parts, where c is the total number of gradients that we can upload at each epoch (i.e., $c = \theta_u|\Delta\mathbf{w}|$). The budget for each potential upload is then divided into two parts. The first will be spent on checking whether the gradient $\Delta w_j^{(i)}$ of a randomly chosen parameter j is above the threshold τ. The second will be spent on actually releasing (uploading) the gradient if it is above the threshold. We use the Laplacian mechanism to add noise during selection and upload according to the allocated privacy budgets. The noise depends on the privacy budget as well as the sensitivity of the gradient for each parameter. In the following, we assume the same sensitivity Δf for all parameters, but this is not a requirement, and different parameters may have different sensitivities.

Figure 12 presents the pseudocode of differentially private DSSGD. To split ϵ, we follow [20]. $\frac{8}{9}$ of $\frac{\epsilon}{c}$ is devoted to the selection, where part of it is spent on random noise r_w and the other part is spent on random noise r_τ. The remaining $\frac{1}{9}$ is devoted to the released value. Note that r_τ is not re-generated after failed threshold checks. This

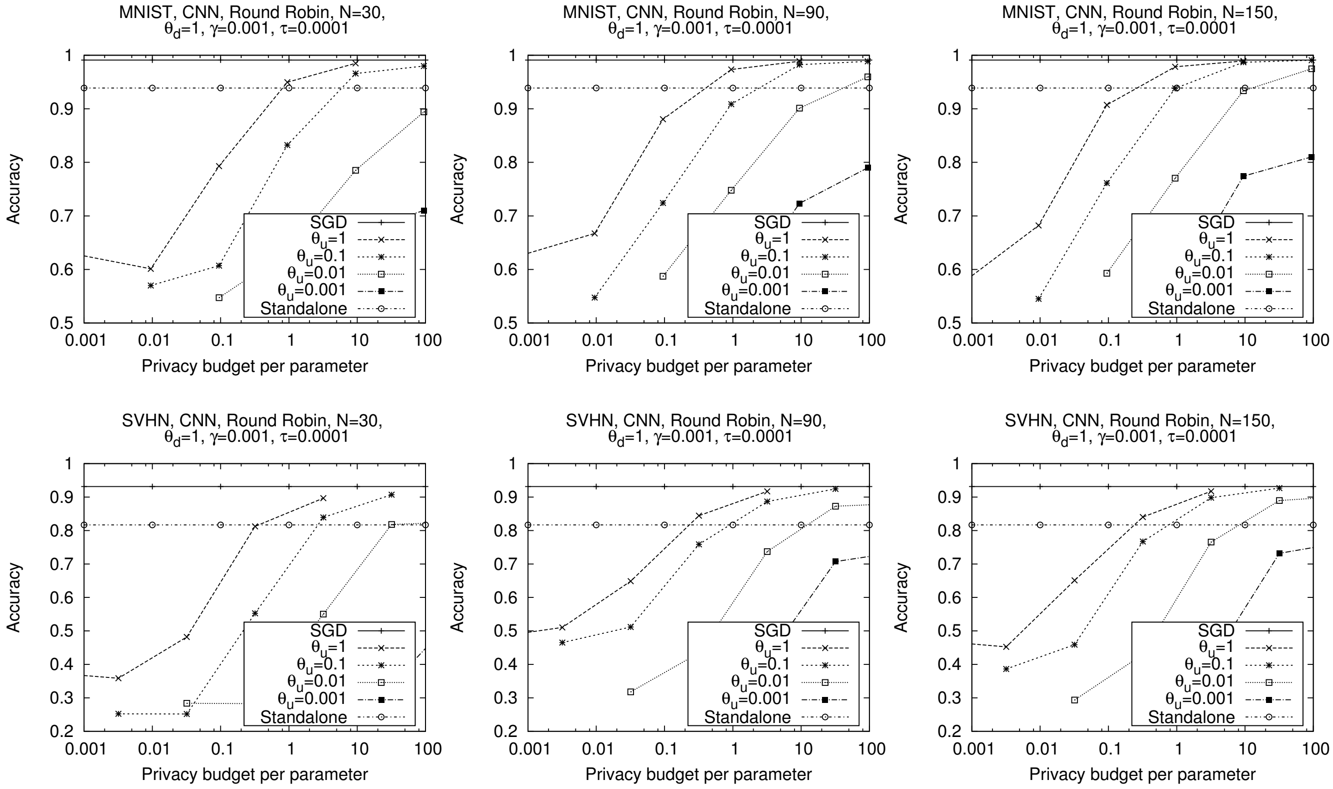

Figure 13: Accuracy of differentially private DSSGD for different datasets, number of participants, fraction of uploaded gradients, and privacy budget. Each subfigure plots the per-parameter privacy budget (i.e., $\frac{\epsilon}{c}$) versus accuracy. The accuracy of SGD and Standalone are plotted for comparison.

```
• Let $\epsilon$ be the total privacy budget for one epoch of participant $i$ running DSSGD, and let $\Delta f$ be the sensitivity of each gradient
• Let $c = \theta_u|\Delta\mathbf{w}|$ be the maximum number of gradients that can be uploaded in one epoch
• Let $\epsilon_1 = \frac{8}{9}\epsilon$, $\epsilon_2 = \frac{2}{9}\epsilon$
• Let $\sigma(x) = \frac{2c\Delta f}{x}$
1. Generate fresh random noise $r_\tau \sim \mathrm{Lap}(\sigma(\epsilon_1))$
2. Randomly select a gradient $\Delta w_j^{(i)}$
3. Generate fresh random noise $r_w \sim \mathrm{Lap}(2\sigma(\epsilon_1))$
4. If $\mathsf{abs}(\mathsf{bound}(\Delta w_j^{(i)}, \gamma)) + r_w \geq \tau + r_\tau$, then
    (a) Generate fresh random noise $r'_w \sim \mathrm{Lap}(\sigma(\epsilon_2))$
    (b) Upload $\mathsf{bound}(\Delta w_j^{(i)} + r'_w, \gamma)$ to the parameter server
    (c) Charge $\frac{\epsilon}{c}$ to the privacy budget
    (d) If number of uploaded gradients is equal to $c$, then Halt
        Else Goto Step 1
5. Else Goto Step 2
```

Figure 12: Pseudocode of differentially private DSSGD for participant i using the sparse vector technique

ensures not only that all shared gradients are differentially private, but also that the privacy "penalty" is not paid for gradients that are too small to be shared with other participants.

Estimating sensitivity. The sensitivity of a function determines how much random noise needs to be added to its output to achieve differential privacy. The (global) sensitivity of f is

$$\Delta f = max_{D,D'}||f(D) - f(D')||. \tag{3}$$

Estimating the true sensitivity of stochastic gradient descent is challenging. Instead, we modify the function so that its output stays within fixed, input-independent bounds and use these bounds to estimate sensitivity: that's the bound function that enforces a $[-\gamma, \gamma]$ range on gradient values that may be shared with other participants (Section 5). This approach may reduce accuracy (although in our case the effect is negligible), but privacy is guaranteed. A similar technique was previously used to enforce privacy of MapReduce computations with untrusted mappers [40].

Limiting the range of values that parameters and gradients can take even improves the training process by helping to avoid overfitting. Some existing regularization techniques already force a bound on the norm of the parameters. Max-norm has been used for collaborative filtering [46] and deep learning [47]. Moreover, gradient values truncated into the $[-\gamma, \gamma]$ range indicate the direction and magnitude of moves during gradient descent. Therefore, small values of γ (implying smaller sensitivity and thus smaller noise and higher accuracy) would influence the learning rate of the algorithm but not whether the optimal solution is achievable. Furthermore, as gradients of multiple participants are aggregated, the gradient descent algorithm can traverse through local optima. We discuss the effect of perturbation on distributed selective SGD below.

The meta-parameter γ is set independently of the training data and thus cannot leak any sensitive information. It can be set by training on a calibration dataset with inputs that are similar to the real inputs but are not privacy-sensitive. We then (over-)estimate the sensitivity of our algorithm as 2γ and truncate the uploaded gradients into the $[-\gamma, \gamma]$ range. This helps mitigate the detrimental effect of very large noise values on the training process.

We expect that global sensitivity estimates can be significantly reduced, resulting in higher accuracy, by ensuring that the norm of all gradients is bounded for each update—either globally, or locally, e.g., across all edges leading to a given neural-network node. In fact, the latter kind of norm-bounding is a known regularization technique. We plan to investigate applications of norm-bounding to differentially private deep learning in future work.

7.3 Experimental results

We evaluate the effect of different values of ϵ (the differential privacy parameter), N (the number of participants), and θ_u (the fraction of uploaded gradients) on the accuracy of neural networks trained using distributed selective SGD with differential privacy.

Figure 13 shows the results and compares them with standalone learning and centralized SGD. We set the bound γ to 0.001 and the threshold τ to 0.0001. As expected, smaller ϵ values (i.e., stronger differential privacy guarantees) result in lower accuracy. However, with many participants and when participants share a large fraction of their gradients, the accuracy of differentially private DSSGD is better than the accuracy of standalone training.

7.4 Oblivious parameter server

Regardless of whether the parameter server is trusted, the privacy guarantees of training data separation and differential privacy still hold. However, to prevent a curious server from linking the updates of each participant, it is possible to design a parameter server that is oblivious to uploaders' identities. For example, participants can anonymously authenticate themselves and the gradients they upload [7]. Scalable anonymous communication protocols with provable security can be used to hide participants' identities [12, 54].

The independence of parameters from each other in distributed SSGD, which is inherent to the underlying stochastic gradient descent algorithm, also enables a completely distributed implementation of the parameter storage system where each participant is responsible for a random subset of the parameters. We leave the detailed design of this scheme to future work.

8 Conclusions

This work is the first step in bringing privacy to a machine learning approach that is revolutionizing AI. We proposed a new distributed training technique, based on selective stochastic gradient descent. Our methodology works for any type of neural network and preserves privacy of participants' training data without sacrificing the accuracy of the resulting models. Therefore, it can help bring the benefits of deep learning to domains where data owners are precluded from sharing their data by confidentiality concerns.

Acknowledgments. We are grateful to Adam Smith for explaining how to apply the sparse vector technique and other differential privacy mechanisms in our setting. This work was partially supported by the NSF grants 1223396 and 1408944, NIH grant R01 LM011028-01 from the National Library of Medicine, and Swiss National Science Foundation postdoctoral fellowship to Reza Shokri.

9 References

[1] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *JMLR*, 15(1):1111–1133, 2014.

[2] M. Avriel. *Nonlinear Programming: Analysis and Methods*. Courier Corporation, 2003.

[3] M. Barni, P. Failla, R. Lazzeretti, A. Sadeghi, and T. Schneider. Privacy-preserving ECG classification with branching programs and neural networks. *Trans. Info. Forensics and Security*, 6(2):452–468, 2011.

[4] R. Bassily, A. Smith, and A. Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *FOCS*, 2014.

[5] Y. Bengio. Learning deep architectures for AI. *Foundations and trends in machine learning*, 2(1):1–127, 2009.

[6] J. Bos, K. Lauter, and M. Naehrig. Private predictive analysis on encrypted medical data. *J. Biomed. Informatics*, 50:234–243, 2014.

[7] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In *CCS*, 2006.

[8] K. Chaudhuri and C. Monteleoni. Privacy-preserving logistic regression. In *NIPS*, 2009.

[9] K. Chaudhuri, C. Monteleoni, and A. Sarwate. Differentially private empirical risk minimization. *JMLR*, 12:1069–1109, 2011.

[10] K. Chaudhuri, A. Sarwate, and K. Sinha. A near-optimal algorithm for differentially-private principal components. *JMLR*, 14(1):2905–2943, 2013.

[11] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A Matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[12] H. Corrigan-Gibbs and B. Ford. Dissent: Accountable anonymous group messaging. In *CCS*, 2010.

[13] A. A. Cruz-Roa, J. E. A. Ovalle, A. Madabhushi, and F. A. G. Osorio. A deep learning architecture for image representation, visual interpretability and automated basal-cell carcinoma cancer detection. In *MICCAI*, 2013.

[14] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. Le, et al. Large scale distributed deep networks. In *NIPS*, 2012.

[15] O. Denas and J. Taylor. Deep modeling of gene expression regulation in an erythropoiesis model. In *Representation Learning, ICML Workshop*, 2013.

[16] L. Deng. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Trans. Signal and Information Processing*, 3, 2014.

[17] W. Du, Y. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *SDM*, volume 4, pages 222–233, 2004.

[18] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12:2121–2159, 2011.

[19] C. Dwork. Differential privacy. In *Encyclopedia of Cryptography and Security*, pages 338–340. Springer, 2011.

[20] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Theoretical Computer Science*, 9(3-4):211–407, 2013.

[21] C. Dwork, G. Rothblum, and S. Vadhan. Boosting and differential privacy. In *FOCS*, 2010.

[22] R. Fakoor, F. Ladhak, A. Nazi, and M. Huber. Using deep learning to enhance cancer diagnosis and classification. In *WHEALTH*, 2013.

[23] A. Graves, A.-R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

[24] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al. Deepspeech: Scaling up end-to-end speech recognition. *arXiv:1412.5567*, 2014.

[25] M. Hardt and G. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *FOCS*, 2010.

[26] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *arXiv:1502.01852*, 2015.

[27] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine*, 29(6):82–97, 2012.

[28] G. Jagannathan and R. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *KDD*, 2005.

[29] P. Jain, V. Kulkarni, A. Thakurta, and O. Williams. To drop or not to drop: Robustness, consistency and differential privacy properties of dropout. *arXiv:1503.02031*, 2015.

[30] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998.

[32] M. Liang, Z. Li, T. Chen, and J. Zeng. Integrative data analysis of multi-platform cancer data with a multimodal deep learning approach. *Trans. Comput. Biology and Bioinformatics*, 12(4):928 – 937, 2015.

[33] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *CRYPTO*, 2000.

[34] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[35] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. In *Deep Learning and Unsupervised Feature Learning, NIPS Workshop*, 2011.

[36] M. Pathak and B. Raj. Privacy-preserving speaker verification and identification using gaussian mixture models. *Trans. Audio, Speech, and Language Processing*, 21(2):397–406, 2013.

[37] M. Pathak, S. Rane, and B. Raj. Multiparty differential privacy via aggregation of locally trained classifiers. In *NIPS*, 2010.

[38] M. Pathak, S. Rane, W. Sun, and B. Raj. Privacy preserving probabilistic inference with Hidden Markov Models. In *ICASSP*, 2011.

[39] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[40] I. Roy, S. T. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for MapReduce. In *NSDI*, 2010.

[41] B. Rubinstein, P. Bartlett, L. Huang, and N. Taft. Learning in a large function space: Privacy-preserving mechanisms for SVM learning. *J. Privacy and Confidentiality*, 4(1):4, 2012.

[42] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In *Neurocomputing: Foundations of research*, pages 673–695. MIT Press, 1988.

[43] A. Sarwate and K. Chaudhuri. Signal processing and machine learning with differential privacy: Algorithms and challenges for continuous data. *Signal Processing Magazine*, 30(5):86–94, 2013.

[44] D. Shultz. When your voice betrays you. *Science*, 347(6221), 2015.

[45] P. Simard, D. Steinkraus, and J. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Document Analysis and Recognition*, 2013.

[46] N. Srebro and A. Shraibman. Rank, trace-norm and max-norm. In *Learning Theory*, pages 545–560. Springer, 2005.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

[48] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014.

[49] Torch7. A scientific computing framework for LuaJIT (torch.ch).

[50] J. Vaidya and C. Clifton. Privacy preserving association rule mining in vertically partitioned data. In *KDD*, 2002.

[51] J. Vaidya, M. Kantarcıoğlu, and C. Clifton. Privacy-preserving naive bayes classification. *VLDB*, 17(4):879–898, 2008.

[52] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.

[53] M. Wainwright, M. Jordan, and J. Duchi. Privacy aware learning. In *NIPS*, 2012.

[54] D. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI*, 2012.

[55] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. Lauter, and M. Naehrig. Crypto-nets: Neural networks over encrypted data. *arXiv:1412.6181*, 2014.

[56] H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, D. Merico, R. K. Yuen, Y. Hua, S. Gueroussov, H. S. Najafabadi, T. R. Hughes, et al. The human splicing code reveals new insights into the genetic determinants of disease. *Science*, 347(6218), 2015.

[57] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett. Functional mechanism: Regression analysis under differential privacy. *VLDB*, 5(11):1364–1375, 2012.

[58] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *ICML*, 2004.

[59] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *NIPS*, 2010.