CROWD-ML: A LIBRARY FOR PRIVACY-PRESERVING MACHINE LEARNING ON SMART DEVICES

Jihun Hamm, Jackson Luken, Yani Xie

The Ohio State University, Columbus, OH 43210, USA

ABSTRACT

When user-generated data such as audio and video signals are used to train machine learning algorithms, users' privacy must be considered before the learned model is released. In this work, we present an open-source library for privacy-preserving machine learning framework on smart devices. The library allows Android and iOS devices to collectively learn a common classifier/regression model from distributed data with differential privacy, using a variant of minibatch stochastic gradient descent method. The library allows researchers and developers to easily implement and deploy customized tasks that use on-device sensors to collect sensitive data for machine learning.

Index Terms— differential privacy, machine learning, smart device, crowdsourcing, distributed optimization

1. INTRODUCTION

Smart and connected devices are becoming increasingly pervasive in daily life, including smartphones, wearables, smart home appliance, smart meters, connected cars, surveillance cameras, and environmental sensors. Can we benefit from analyzing the massive data generated from such devices, without storing the data centrally nor breaching users' privacy? More specifically, can machine learning be outsourced to a crowd of smart devices with a strong privacy guarantee?

In this paper, we present an open-source library[1] for privacy-preserving machine learning framework on smart devices. The library allows Android and iOS devices to learn a common classifier/regression model with differential privacy by solving the distributed optimization problem:

$$\min_{w \in \mathcal{W}} f(w) = \min_{w \in \mathcal{W}} \frac{1}{M} \sum_{i=1}^{M} f_i(w), \qquad (1)$$

where w is the common parameter vector, and $f_i(w)$ is the empirical risk of device i:

$$f_i(w) = \frac{1}{|Z^i|} \sum_{(x,y) \in Z^i} l(h(x;w), y). \qquad (2)$$

Here Z^i is the i.i.d. training data set of device i, and we assume the datasets $Z^1, ..., Z^M$ from M devices are also i.i.d.

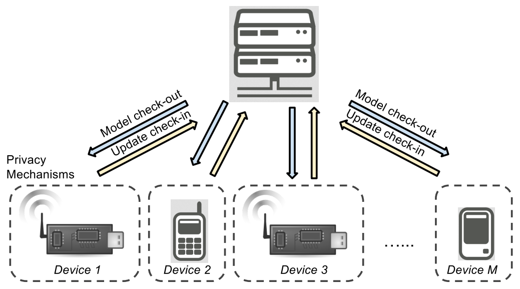

Fig. 1. Distributed machine learning on smart devices

Iterative optimization of such objectives distributed over multiple agents has been studied by many researchers (e.g., [1, 2, 3, 4].) In this paper, we focus on a semi-synchronous model which consists of multiple devices and a central server/coordinator with bounded communication delays (see Fig. 1.) We present a variant of Stochastic Gradient Descent (SGD) optimization which uses device-side and server-side minibatches and also allows local weight update on devices. Each device computes and transmits the weight vector w after a number of local updates, and the server stores the average of all w's from multiple devices. The simplicity of gradient-based optimization allows such a procedure to performed on smart devices with limited resource.

Recently, many researchers have incorporated privacy mechanisms into stochastic gradient descent-based optimization [5, 6] and/or solving distributed learning problems [7, 8, 9, 10], using differential privacy as a quantifiable measure of privacy [11, 12, 13]. An appropriate choice of mechanism depends on several factors–objective functions (e.g., ERM with strictly-convex, differentiable, and smooth loss function [14]), granularity of privacy (e.g., sample, person, or group) and adversary assumption (e.g., other devices, server, or the public.) In this library, we use additive noise to sanitize communication from a device to a server. We provide basic sensitivity analysis of the proposed method in Sec. 3.

Although there has been a lot of research in distributed optimization and learning, including a similar learning framework for smart devices [4], Crowd-ML is unique in that it is

This research was supported in part by Google IoT Research Tech Awards Pilot and Google Faculty Research Award.

[1]`https://github.com/jihunhamm/Crowd-ML`

an open-source library with emphasis on privacy. The optimization algorithm in this library is also more general and flexible than our previous paper [9].

2. DISTRIBUTED MACHINE LEARNING

2.1. Losses and classifiers

In the library we implement the following basic losses/classifiers:

	$l(h(x;w),y)$
Logistic Regress.	$\log(1+e^{-yw^Tx})+0.5\lambda\|w\|^2$
Softmax	$-w_y^Tx+\log\sum_k e^{w_k^Tx}+0.5\lambda\sum_k\|w_k\|^2$
Soft-margin SVM	$\max\{0,1-y(w^Tx+b)\}+0.5\lambda\|w\|^2$

In addition, we also implement a multi-layer neural network, with the choice of sum-squared-error or softmax at the top layer, and the choice of activation functions from tanh, sigmoid, and ReLU.

2.2. Optimization

We use a variant of the standard stochastic gradient descent

$$w(t+1)=w(t)-\eta(t)\nabla f_i(w(t)) \qquad (3)$$

to solve (1). Firstly, we use both device-side and server-side minibatches. Device-side minibatch plays an important role in reducing the number of communication rounds. In an environment where communication is costly, increasing the device-side minibatch size B_d allows more work to be performed on each device lowering the frequency of communication. Server-side minibatch plays a different role. Server-side minibatch lowers the variance of the noisy weights from multiple devices and also mitigates negative effects of communication delays [3]. Secondly, we also allow each device to perform self-updates of (3) and sends the current parameters w to the server, instead of sending the gradient and relying on the server to perform the update (3) [2]. Similar to device minibatch, local updates can lower the communication frequency and keep a device improving its model when it is off-line.

Several commonly learning rate types are implemented, including $\eta(t)=c_0$, $\eta(t)=c_0/t$, $\eta(t)=c_0/\sqrt{t}$, and two adaptive methods – AdaGrad [15] and RMSprop [16].

We summarize the optimization procedure in Alg. 1. Note that there are two iteration counters each for device-side and server-side routines: number of minibatches processed at the device (t_d) and the server (t_s), number of local weight updates at the device (s_d), and number of weights received from devices (s_s). The learning rate can depend on all four counters. In Alg. 1 we show a simple rate that depends only on local counters, e.g., $\eta(S_d(t_d-1)+s_d)$. Using local counters as oppose to global counters makes little difference if all devices communicate with a server with a similar frequency.

[2]The latter is equivalent to the former with $S_d=1$ local update where the server receives $w(t)-\eta(t)\nabla f_i(w(t))$, assuming the server knows $\eta(t)$.

Algorithm 1 Optimization for Crowd-ML

```
Device Routine
  Init: read loss type, noise type, max iter $T_d$, local update
  num $S_d$, device minibatch size $B_d$, from the server
  for $t_d = 1,...,T_d$ do
    Check out weights $w$ and $t_s$ from the server
    for $s_d = 1,...,S_d$ do
      Collect $B_d$ i.i.d. samples $Z=\{(x_i,y_i)\}$
      Compute avg gradient $g=\frac{1}{|Z|}\sum_{z\in Z}\nabla_w f(z)$
      Update $w \leftarrow w-\eta(S_d(t_d-1)+s_d)\,g$
    end for
    Add noise $w \leftarrow w+\xi$ from (6)
    Check in $w$ and $t_s$ to the server
  end for
Server Routine
  for $t_s = 1,...,T_s$ do
    Set $W=\{\}$
    for $s_s = 1,...,B_s$ do
      (Asynchronously) receive a check-out request and
      send $\hat{w}$ and $t_s$
      Receive a check-in request from a device
      if Device $t_s$ matches server $t_s$ then
        Receive $w$ and append: $W \leftarrow W \cup \{w\}$
      end if
    end for
    Update $\hat{w} \leftarrow \frac{1}{|W|}\sum_{w\in W} w$
  end for
```

2.3. Perturbation by additive noise

Additive noise is sampled independently and added to the parameters to be transmitted from a device to a server. Two commonly used distributions are Gaussian and Laplace-like (6) distributions which we implemented in the library. Given a privacy budget ϵ and the max number of communication T_d, a simple mechanism is to use i.i.d. noise for each communication. However, noise can be chosen to decrease geometrically [8], decrease proportionally to the learning rate (see Sec. 3), or sampled from a random process in which the noise sequence is not independent [17]. Deciding an optimal noise sequence with the least utility loss is still an open problem.

3. PRIVACY ANALYSIS

3.1. Differential privacy

A randomized algorithm that takes data $\mathcal{D}$ as input and outputs a function f is called ϵ-differentially private if

$$P(f(\mathcal{D})\in\mathcal{S})\le e^{\epsilon}P(f(\mathcal{D}')\in\mathcal{S}) \qquad (4)$$

for all measurable $\mathcal{S}\subset\mathcal{T}$ of the output range and for all datasets $\mathcal{D}$ and $\mathcal{D}'$ differing in a single item, denoted by $\mathcal{D}\sim\mathcal{D}'$. That is, even if an adversary knows the whole dataset $\mathcal{D}$

except for a single item, she cannot infer much more about the unknown item from the output f of the algorithm. When an algorithm outputs a real-valued vector $f \in \mathbb{R}^D$, its global sensitivity [12] can be defined as

$$S(f) = \max_{\mathcal{D}\sim\mathcal{D}'} \|f(\mathcal{D}) - f(\mathcal{D}')\| \tag{5}$$

where $\|\cdot\|$ is a norm. An important result from [12] is that a vector-valued output f with sensitivity $S(f)$ can be made ϵ-differentially private by perturbing f with an additive noise vector η whose density is

$$P(\eta) \propto e^{-\frac{\epsilon}{S(f)}\|\eta\|}. \tag{6}$$

3.2. Privacy of Crowd-ML

We compute the sensitivity of each round of communications. Without loss of generality, we assume $\mathcal{D}$ and $\mathcal{D}'$ are different only for the device 1 and the same for the other devices. The private data of device 1 may be arbitrarily different in $\mathcal{D}$ and $\mathcal{D}'$. Suppose device 1 starts from the weight $w(t, 0)$, performs local updates for S_d times

$$w(t, s{+}1) = w(t, s) {-} \eta(t, s)\nabla f_i(w(t, s)),\ s = 0, ..., S_d{-}1, \tag{7}$$

where $w(t, 1), ..., w(t, S_d)$ is the sequence of intermediate weights values. After S_d local updates, we get

$$w(t, S_d) = w(t, 0) - \sum_{s=0}^{S_d-1} \eta(t, s)\nabla f_i(w(t, s)), \tag{8}$$

and therefore the sensitivity of the weight sent to a server $S(w(t, S_d))$ is

$$\begin{aligned} & \max_{D\sim D'} \|w_{\mathcal{D}}(t, S_d) - w_{\mathcal{D}'}(t, S_d)\| \qquad (9) \\ = & \max_{D\sim D'} \left\| \sum_{s=0}^{S_d-1} \eta(t, s)[\nabla f_i(w_{\mathcal{D}}(t, s)) - \nabla f_i'(w_{\mathcal{D}'}(t, s))] \right\|. \end{aligned}$$

For non-adaptive and decreasing rate $\eta(t, s)$, we have

$$\begin{aligned} S(w(t, S_d)) &\le S_d\ \eta(t, 0)\ S(\nabla f_i), \quad \text{where} \qquad (10) \\ S(\nabla f_i) &= \max_{D\sim D'} \|\nabla f_i(w_{\mathcal{D}}(t, s)) - \nabla f_i'(w_{\mathcal{D}'}(t, s))\| \end{aligned}$$

is the sensitivity of the gradient.

Note that if the server is trusted, then the sensitivity can be reduced at least by a factor of $1/B_s$, because an adversary (i.e., another device) only sees an averaged parameter (vector) of B_s devices. Furthermore, if the granularity of privacy we consider is a single sample instead of all N samples of a device, we have an additional $1/N$ factor of reduction in the sensitivity. These reductions in sensitivity can make privacy-preserving learning practical.

3.2.1. Sensitivity of convex loss

The sensitivity (10) can be computed for certain loss functions, such as strongly convex differentiable loss functions with bounded derivatives $|l'(\cdot)| \le 1$ [14]. In this case, the sensitivity is bounded in terms of the convexity coefficient λ (e.g., from regularization), and data diameter $\max\|x\|$:

$$S(w(t, S_d)) \le (2/\lambda)\ S_d\ \eta(t) \max\|x\|. \tag{11}$$

3.2.2. Sensitivity of neural networks

We also consider the sensitivity of a neural network. A simple way to bound (10) in a neural network is to simply clamp each element of a gradient to a range $[-c, c]$ [18] or bound the norm (e.g., l_2) of the gradient [19]. We also present an alternative. Let $o_j = \sigma(n_j)$ be the output of an activation function, $n_j = \sum_i w_{ji}o_i$ the net function, and $\delta_j = \frac{\partial l}{\partial n_j}$ the error. Using backpropagation, the gradient can be expressed as

$$\frac{\partial l}{\partial w_{ji}} = \frac{\partial l}{\partial n_j}\frac{\partial n_j}{\partial w_{ji}} = \delta_j o_i, \ \ \text{where} \tag{12}$$

$$\delta_j = \begin{cases} \frac{\partial l}{\partial n_j}, & \text{(top layer)} \\ \sigma'(o_j)\sum_k w_{kj}\delta_k, & \text{(other layers)} \end{cases}. \tag{13}$$

For a bounded activation function ($|\sigma| \le 1$) with bounded derivatives ($|\sigma'| \le 1$) such as sigmoid, the equation shows that a gradient component $\partial l/\partial w_{ij}$ is basically bounded by the product of weights from the top layer to the current layer. Therefore we can also bound the sensitivity by bounding the range of weight values.

4. EVALUATION

4.1. Platforms

Firebase. We use Google Firebase[3] as a secure and scalable database to store various parameters during optimization. The database keeps three types of information. The first type is task configuration such as loss, noise type and hyperparameters of the algorithm. It is read-only for devices and writable for the server. A device uses this information to initialize its routine. The second type of information is the current state such as weights w and iteration number t. Devices can only check out and cannot check in these values. The third type is the latest weights checked in by devices. Each device can write to its own space, and the server reads the valued when notified of any change in the values.

Server. The server/coordinator is simply another client of the Firebase database that runs the server routine (Alg. 1) as a node.js app written in JavaScript[4]. The script controls the start and stop of training, and monitors the process by

[3]https://firebase.google.com/
[4]https://nodejs.org

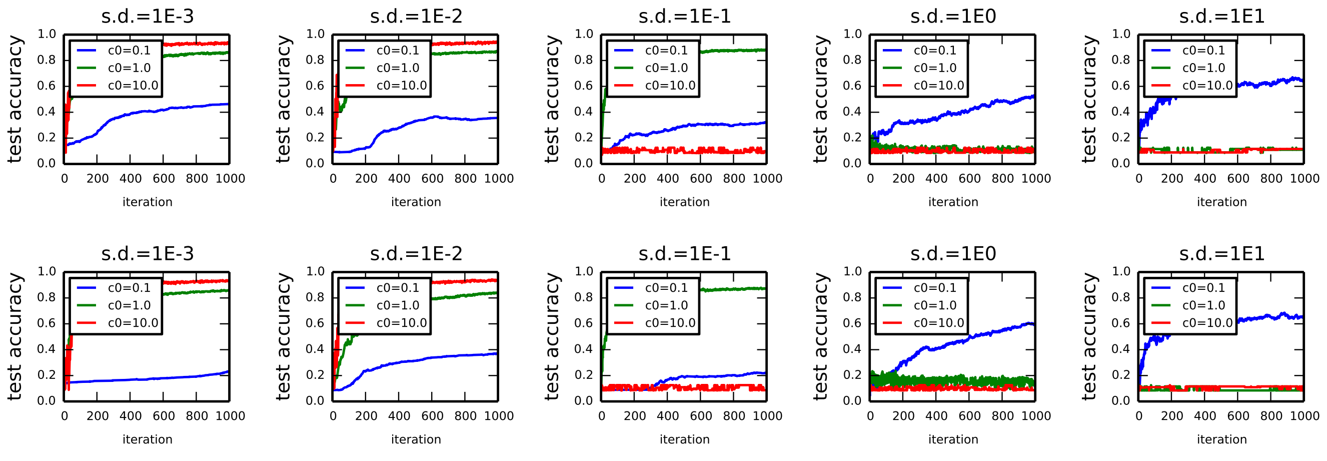

Fig. 2. Test accuracy with MNIST data, with different learning rates ($c_0 = 0.1, 1.0, 10$) and different noise scales (s.d.=1E-3 to 1E1), for Laplace (top row) and Gaussian (bottom row) noise type.

computing accuracy using validation data during training.

Devices. The device routine (Alg. 1) is implemented as an Android app in Java and an iOS app in Objective C, which perform the same function. When the app is launched, it waits for a wi-fi connection to avoid 4G usage, and then tries to log in to the Firebase and initialize itself using the configuration from the Firebase. If successful, the app starts reading local private data from the device, checks out current parameters from the Firebase, computes the gradients/weights, and checks in the result after perturbation by noise.

4.2. Experiments

We test the library with the MNIST dataset [5], with $K = 10$ classes, $N = 60000/1000$ training/testing samples, and the dimensionality reduced to $D = 50$ using PCA.

The first set of tests demonstrates the impact of noise scales and learning rates on convergence and accuracy. We use a two-layer ReLU network with 75 hidden units per layer with a softmax top layer. Other parameters are: max iteration $T = 1000$, learning rate $\eta(t) = c_0/\sqrt{t}$, server batch size $B_s = 1$, device batch size $B_d = 100$, and local update number $S_d = 10$. Two types of noises were tested, Laplace (top row) and Gaussian (bottom row.) We add i.i.d. noise to each communication with varying scale $(10^{-3}, 10^{-2}, 10^{-1}, 1, 10)$. From Fig. 2 we make the following observations. The final accuracy after $T = 1000$ iterations is quite sensitive to the constant c_0 in the learning rate, and the best rate is different at each noise level. A larger rate $c_0 = 10$ works better with low noise (s.d.=10^{-3}) and a smaller rate $c_0 = 0.1$ works better with high noise (s.d.=10.) As the noise scale increases, the accuracy of the best rate decreases, from 0.95 (s.d.=10^{-3}) to 0.65 (s.d.=10). Using Laplace vs Gaussian noise with the same scale does not seem to make a noticeable difference, despite the fact that the privacy guarantees of using Laplace vs Gaussian are quite different [20].

We also demonstrate the library on physical devices. We use 80 Android devices of various manufacturers and Android versions, and solve the MNIST 10 class problem using softmax without hidden layers. Other parameters are: server batch size $B_s = 20$, device batch size $B_d = 100$, no local update $S_d = 0$, and no noise. An accuracy of 0.87 was achieved after $t_s = 100$ iterations, which took about 40–60 minutes in real time. During this time, about 10 out of 80 devices were actively running, and each device made about $t_d = 25$ communications on average with the server. In comparison, when the same task was performed by a single device, the accuracy after $t_d = 25$ communications was about 0.76 compared to 0.87 with multiple devices, which demonstrates the advantage of crowd-sourced learning.

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

5. CONCLUSION

In this paper, we presented an open-source library for privacy-preserving machine learning for Android and iOS devices, which minimizes empirical loss of various types using a variant of stochastic gradient descent that uses device-side and server-side minibatches and allows local weight updates. Communication between a device and a server is sanitized by additive noise to meet differential privacy criteria.

We envision that the library, as an open-source, will evolve to assimilate newest updates in private distributed optimization, and allow non-experts to use the library to deploy interesting learning tasks on smart devices.

6. REFERENCES

[1] J.N. Tsitsiklis, D.P. Bertsekas, and M. Athans, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," in *American Control Conference*, 1984, pp. 484–489.

[2] A. Agarwal and J.C. Duchi, "Distributed delayed stochastic optimization.," in *Proc. Neural Information Processing Systems*, 2011, pp. 873–881.

[3] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction," in *Proc. International Conference on Machine Learning*, 2011.

[4] H.B. McMahan, E. Moore, D. Ramage, et al., "Federated learning of deep networks using model averaging," *arXiv preprint arXiv:1602.05629*, 2016.

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

[6] R. Bassily, A. Smith, and A. Thakurta, "Private empirical risk minimization: Efficient algorithms and tight error bounds," in *Foundations of Computer Science, IEEE 55th Annual Symposium on*. IEEE, 2014, pp. 464–473.

[7] A. Rajkumar and S. Agarwal, "A differentially private stochastic gradient descent algorithm for multiparty classification," in *International Conference on Artificial Intelligence and Statistics*, 2012, pp. 933–941.

[8] Z. Huang, S. Mitra, and N. Vaidya, "Differentially private distributed optimization," in *Proceedings of the 2015 International Conference on Distributed Computing and Networking*. ACM, 2015, p. 4.

[9] J. Hamm, A. Champion, G. Chen, M. Belkin, and D. Xuan, "Crowd-ML: A privacy-preserving learning framework for a crowd of smart devices," in *Proceedings of the 35th IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2015.

[10] J. Hamm, P. Cao, and M. Belkin, "Learning privately from multiparty data," in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 555–563.

[11] C. Dwork and K. Nissim, "Privacy-preserving datamining on vertically partitioned databases," in *Advances in Cryptology*. Springer, 2004, pp. 528–544.

[12] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography*, pp. 265–284. Springer, 2006.

[13] C. Dwork, "Differential privacy," in *Automata, languages and programming*, pp. 1–12. Springer, 2006.

[14] K. Chaudhuri, C. Monteleoni, and A.D. Sarwate, "Differentially private empirical risk minimization," *Journal of Machine Learning Research*, vol. 12, pp. 1069–1109, 2011.

[15] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[16] T. Tieleman and G. Hinton, "Coursera: Neural networks for machine learning, lecture 6.5," 2012.

[17] R. Hall, A. Rinaldo, and L. Wasserman, "Differential privacy for functions and functional data," *Journal of Machine Learning Research*, vol. 14, no. Feb, pp. 703–727, 2013.

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

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

[20] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.