Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 165 (2019) 349–355

Procedia Computer Science

www.elsevier.com/locate/procedia

INTERNATIONAL CONFERENCE ON RECENT TRENDS IN ADVANCED COMPUTING 2019, ICRTAC 2019

Performance Analysis of Distributed and Federated Learning Models on Private Data

Kunal Chandiramani, Dhruv Garg*, Maheswari N

Vellore Institute of Technology, Vandalur Kelambakkam Road, Chennai, 600127, India

Abstract

There has been significant research in privacy-related aspects of machine learning and large scale data processing. In traditional methods of training a model, data is gathered at a centralized machine where training on the entire data takes place. This has led to a major problem of not only scalability but also of preserving the anonymity and privacy of sensitive user data. As a consequence, there has been extensive work done towards distributed machine learning. In more recent times, Federated Learning has gained a lot of traction. This is because of the features that make it highly suitable to train models collaboratively while preserving the privacy of sensitive data. In this paper, we compare basic machine learning, distributed machine learning, and federated learning by modelling on the Fashion MNIST dataset. Our results show that federated learning model not only maintains privacy but is also fast, and allows deployment at scale- even with low compute, mobile devices.

Keywords: Data privacy; Machine Learning; Distributed Machine Learning; Federated Learning; Artificial Intelligence

1. Introduction

Machine Learning (ML) and Artificial Intelligence (AI) provide systems the capability to learn and improve models from experience without being explicitly programmed. These technologies have become of widespread use in recent times: they have been employed in many different domains and in numerous applications within those domains. The range of their applicability is wide, from basic binary classification in a simple data-set to sophisticated recommendation systems suggesting products or services to the end-users. Moreover, there has been exponential growth in user data and the avenues to harness it. This data is being leveraged with a goal to enhance the user's knowledge and productivity.

Although machine learning applications are widely used, they face a *major issue of data privacy and policy*. Many applications require data of its own users: user location, search queries, browsing the history, call logs, bank

* *Corresponding author. Tel.: +91-9969017594*
E-mail address: dhruvshekhar.garg2016@vitstudent.ac.in

1877-0509
10.1016/j.procs.2020.01.039

transactions, products bought, videos watched, etc., are all recorded and stored. To train a model, this data is uploaded to a centralized server where machine learning algorithms are applied to extract patterns. This establishes a cycle: new data acquired is used to build more sophisticated and better performing models. This problem exacerbates when a user's data falls into wrong hands. Recent work has stated that an anonymized data-set can also be attacked in terms of privacy. Shui Yu [9] cites that from a data-set containing movement data (spatio-temporal points) of 1,500,000 people, models were able to identify people with approximately 95 percent accuracy. It must also be noted here, that the concern around data privacy is not as recent as it is thought to be. The community has anticipated this and has been investing significant effort in privacy aware learning [10] and privacy-preserving data publishing [11]. There have also been new technologies like blockchain which is resistant to modification of data. Moreover, blockchain technology builds upon cryptography, which is a powerful tool for privacy protection. A major drawback though for cryptographic techniques is that, in a world of mobile devices, limited computing power cannot possibly support intensive encryption and decryption [9].

To further examine this *dual-problem* of maintaining privacy while computing large data-sets, we compare the existing model training methods (basic and distributed) with a recent model training method (federated learning). We build a neural network which plays the role of a classifier in all the three training mechanisms. Thus, while the underlying model being trained is the same, the way in which the parameter updations occur is different. This difference helps us to evaluate the models on parameters of time taken to train and the accuracy of the finally trained model. While the basic and distributed methods of training are widely known, federated learning is a more recent category of distributed machine learning, which is able to train on a large corpus of decentralized data while preserving its confidentiality.

Federated Learning is a flourishing research topic which was first proposed by Google [12,13,14] in 2016. The humongous volumes of data can be very useful in training models, but new regulations on data localization and privacy prevent companies from using it freely. Federated learning tries to solve this problem by providing a privacy-preserving mechanism to train models on decentralized data. Instead of collecting data into a central server, the model is sent to each end device for training. In this way, federated learning has the potential to disrupt todays dominant model of centralized computing.

The remaining parts of this manuscript are organized as follows: the literature review in Section 2, followed by the methodology of the three classifiers in Section 3; the experimental results are discussed in Section 4, and the conclusion and discussions are described in Section 5.

2. Literature Survey

In this section, we have discussed papers related to distributed machine learning, federated learning, and privacy-preserving machine learning. One of the most important parts in research is reading different existing literature's implementation, advantages, and disadvantages. We observed that federated learning is a recent topic which has few existing literature's and articles. A lot of work has already been done in privacy-preserving learning and distributed machine learning algorithms. For example, ref [15,16] proposed to train a decision tree algorithm over vertically partitioned data while safeguarding the sensitive private data. Furthermore, Vidya and Chris also developed privacy preserving K-means clustering [19], privacy preserving Naive Bayes classifier [17] and safe association mining guidelines [18] over vertically partitioned data. Ref [20,21] in their literature proposed Support Vector Machine (SVM) Classification on vertically and horizontally partitioned data. Ref [22] delineated gradient descent methods on private data, ref [23] suggested safe multi-party linear regression and classification protocols. All the above works operated on secure multi-party commutation (SMC) [24,25] for privacy assurance. A more detailed description of the literature related to this paper are described below.

In February 2019, Qiang Yang et al. [1] in their literature discussed the major challenges artificial intelligence is facing right now. The first concern was about the isolation of data, and the second concern was on improving the data privacy and security situation. To solve these challenges, they proposed a secure federated learning platform, which consisted of vertical federated learning, horizontal federated learning, and federated transfer learning.

In 2018, Mohammad et al. [2] provided an introduction to the intersection of machine learning and privacy communities by proposing techniques which can be used to protect data. In their paper, they have focused on one major issue: for training a machine learning model data is still being uploaded to a centralized server (cloud) daily. This is likely to violate a users privacy.

In 2019, Keith Bonawitz et al. [3] described how federated learning could be used to train large corpus of decentralized data. They described the resulting high-level sketch of federated learning. In addition to this, they also elucidated the major challenges that distributed learning faces right now, their solutions and the future direction.

In 2017, Virginia Smith et al. [4] proposed a multi-task federated learning system which enabled various sites to complete distinct tasks, while sharing knowledge and safeguarding security. In addition to this, their model was shown to mitigate issues such as high communication cost, fault tolerance and stragglers.

In 2016, Nicolas Papernot et al. [5], drew attention to attacks on machine learning systems and the defences used to counter them. They described a comprehensive threat model for machine learning: not just the algorithm in isolation, but the entire data pipeline. Their paper lists desirable properties to improve the security and privacy of models. Finally, they showed that there are tensions between model complexity, accuracy, and resilience that must be adjusted for the environments in which they are used.

In 2017, Keith Bonawitz et al. [6], designed a robust protocol for secured aggregation of high-dimensionality data. The work leverages a secure multi-party computation (MPC) system to compute and update the model weights/parameter from the individual users' devices. They claimed that their protocol is suitable for use in a federated learning setting by aggregating user-provided model updates for a deep neural network. Added benefits of the protocol also include low communication overhead, resilience against failures.

In 2014, Raphael Bost et al. [7], re-iterated the fact that it is imperative on those building models from the data should maintain the confidentiality of both - the data and the classifier. They state that the feature vector x and the model w should be kept a secret from one or all of the stakeholders involved. They developed three major privacy-preserving classification protocols - hyper-plane decision, Naive Bayes classifier, and Decision Tree classifier.

In 2016, Dimitra Kamarinou et al. [8], provided an analysis of the implications of using machine learning algorithms for profiling of individuals concerning the European Union General Data Protection Regulation (GDPR). The paper explored the data protection rights and obligations, the consequences of the development of machine learning systems, and how personal data is collected and used. They not only present the challenges but also benefits that fair and lawful processing might offer.

3. Classifier Models

In this section of the paper, we have described our implementation as a comparative study between *three classifiers* - the basic machine learning classifier, the distributed machine learning classifier, and the federated learning classifier.

For comparison, we trained the models on the *Fashion-MNIST* data-set using Keras and TensorFlow. The Fashion-MNIST data-set contains 60,000 training and 10,000 testing images. The images are gray-scale images of dimensions 28x28. Each of these images include a picture of different clothes and garments, and are labeled from 1 to 10. The Fashion-MNIST data-set was created to serve as a direct drop-in replacement to the traditional MNIST data-set.

3.1. Basic Machine Learning classifier

In this section, we describe the implementation of the basic machine learning classifier using a Neural Network. The methodology of the model can be divided into four important stages: (1) data pre-processing, (2) defining the Neural Network architecture, (3) training the model, and (4) testing the model. The detailed working of the basic machine learning classifier is shown in Figure.1.

The first and foremost step for training any model is importing the required libraries. In this case they are TensorFlow, Keras, Matplotlib, Numpy, Pandas and time. Once we have imported all the required libraries, we load the Fashion-MNIST data-set, normalize the data and split it into two different sets: one set for training the model and another set for testing the model.

After the data pre-processing step, we define the architecture of our neural network model.

- The *first* layer is the flatten type layer. The main function of this layer is to convert our image (a matrix) into a vector, that is, to take the rows of the matrix and place them one after another.
- The *second* layer is a dense layer or fully connected layer, which means all the neurons of the current layer are connected to all the neurons of the previous layer. This layer consists of 128 neurons and it uses rectified linear unit function (ReLU) as its activation function.

- The *third* and the final layer is also a fully connected layer. This layer consists of neurons equal to the number of classes in Fashion-MNIST data-set (i.e. 10) and it uses softmax as its activation function.

Once we have defined our Neural Networks architecture, the next step is training our model. To train our model, we configure TensorFlow session, which in this case is *tf.train.MonitoredTrainingSession*. Once we create our session, we run it in a while loop to iterate and train the model, until it encounters the stop command. After training our model, we test our model to check how accurate the model is and how well it has generalized on the data given.

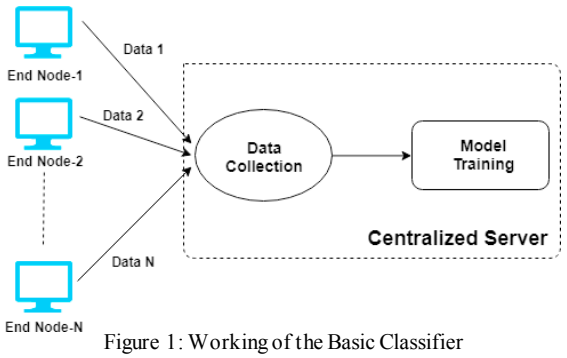


Figure 1: Working of the Basic Classifier

3.2. Distributed Machine Learning classifier

With large data-sets, the computationally complex algorithms often exceed the main memory available. Since traditional machine learning techniques are not suitable for many modern data-sets (due to large to compute requirements from a single computer), implementations moved towards the distributed machine learning model. Distributed machine learning algorithms evolved with the aim to handle very large data sets. They have had the potential to be efficient and scalable algorithms with respect to accuracy and computation. For our distributed machine learning classifier, we made modifications to the previously described basic classifier implementation, and adapted the model to train in a distributed manner. In ideal implementations, distributed machine learning would work among several different devices. However, for simplicity, we simulated the distributed machine learning algorithm to work on different ports of same computer: instead of having the IP addresses of other devices, we utilized different *localhost* ports. This way we could simulate scenarios having several devices when we launched different threads of execution on different ports of the same device.

In this implementation, our goal was to have *3 devices* running in parallel and training the model. Of the 3 devices, *one* was a *parameter server*, which was responsible for saving the global variable states and the other *two* were *workers* who perform the training operations and weight updations. Using the below mentioned flags, we can externally pass the first variables to be initialized. The flags used are:

- *job-name*: a string that denotes a parameter server (*ps*) or a *worker*
- *task-index*: an integer to distinguish the multiple workers and parameter servers
- *ps-hosts*: a string with the port address (or IP addresses) of the parameters server(s)
- *worker-hosts*: a string with the port address (or IP addresses) of the workers

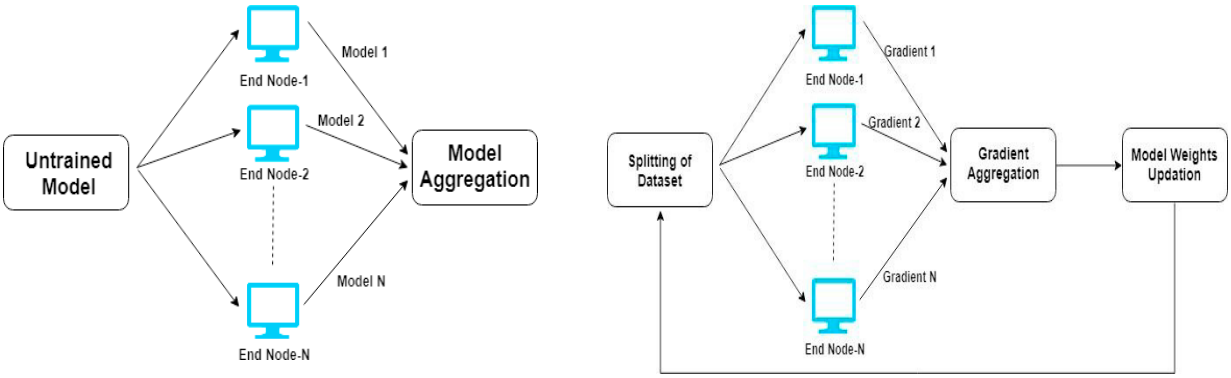
As stated above, the algorithm is using one parameter server and two workers. The script is thus launched thrice, in 3 different terminals. For the parameter server, we set the *job-name* = *ps* and the *task-index* = *0*. Both the worker nodes had *job-name* = *worker* and were differentiated by the *task-index* parameter. One worker had it set to 0 and the other to 1.

Further, we defined a cluster using the Tensor-Flow class *tf.train.ClusterSpec*. As an initialization parameter, the *ClusterSpec* receives a dictionary with the *ps* and *worker keys*. The script running at the parameter server, is continuously listening, and is waiting for the rest to write or read from it. After some time, the parameter server script will block. On passing this step of initialization of parameters, the action shifts exclusively to the worker nodes.

For an equitable training-compute distribution among the workers, we divide the data-set equally between the workers and normalize the data. The distributed classifier works in the following way: each of the worker trains on a

batch of images locally and calculates the gradient that it would need to apply to its weights to reduce the cost. After gradient calculation, it writes the values to the parameter server. Once all the workers (in case of too many workers, a minimum number can be specified by the user) have written their gradients in the parameters server, one of them is selected and behaves like the *chief*. It reads all the gradients, averages them and updates the shared model stored in the parameter server. On receiving the updated gradient average, the chief sends a message to the rest of the workers, to pull the latest model before training on further batches.

As the final step, we define the graph and train the model in a similar way as we did in the basic classifier. The only difference here is that the *tf.device* creates a device-type object and *replica-device-setter* takes care of the synchronization between the devices. It assigns each operation to one of the devices (parameter server, worker(s) or chief). The process of evaluation of the model is the same as the one described in the previous basic machine learning classifier model. Importantly, we must load and maintain regular interval checkpoints, since the trained model ceases to exist once the session is closed. The detailed working of the distributed machine learning classifier is shown in Figure. 2(b).



(a) Federated Classifier. (b) Distributed classifier.

Figure 2: Comparison between working of the Federated and Distributed classifiers

3.3. Federated Machine Learning classifier

In the previous section, we explained the working of the distributed machine learning classifier in which each worker generated a *gradient* for its weights and wrote it to the parameter server. The *chief* worker then reads those gradients from all the workers, averages them and updates the shared model. In the federated learning classifier, each worker will *update its weights locally* as if they are training in isolation. After a fixed interval of steps (based on user's configuration), all the workers will *send their weights* (unlike the distributed learning classifier, they *do not send their gradients*) to the parameter server. After this, the chief worker will read the weights from the parameter server, average them and write it back to the parameter server so that all the worker can update their weights. The detailed working of the federated machine learning classifier is shown in Figure. 2(a).

The initial implementation of federated learning is similar to the distributed machine learning classifier. However, there are two major differences, which are as follows:

- First, we use the *federated-average-optimizer*, an API using which we optimize the federated averaging process.
- Second, we use a variable called *interval-steps*. This variable keeps a count of the pre-defined number of steps after which the workers have to update their weights in the parameter server so that the chief worker can average it. These averaged weights are then updated by all the workers.

In distributed learning, we used the *replica-device-setter* to automatically select which device we had to place each specified operation. Here, we create it only to pass it to the custom optimizer, created to contain the federated average logic as an argument. The *replica-device-setter* will be used by this custom optimizer to place a copy of each trainable variable in the *ps*. These variables will store the average values of all local models. Once this optimizer was defined, we configured the training procedure and, as we did with *SyncReplicasOptimizer*, a hook running within the *MonitoredTrainingSession* handled the synchronization.

The execution of the training session is similar to the distributed machine learning classifier. A notable difference here is that we do not need to define the chief worker: each worker needs to initialize their session and train as if they are the only ones being trained. After the training process was completed, we moved to testing our neural network model to check its accuracy and how well it had generalized on the training data.

4. Experimental Results

In this section, we compare the results of the execution between the *three classifiers*- the basic machine learning classifier, the distributed machine learning classifier and the federated machine learning classifier on the Fashion MNIST data-set (having 60,000 images and 10 classes). As discussed earlier, the core of the model uses a Neural Network algorithm to classify images. Each model is trained for 5 epochs with a batch size of 32 images. In addition to that, we have used categorical cross-entropy to calculate the loss after each epoch. The loss is seen to be successively minimized after each epoch. The detailed results of the classifiers are described below, in Figure.3 and Table 1.

The training of the *basic classifier* model takes place on the same node and there is no distribution of data before training the model. Thus, it has no communication overhead. However, due to limited compute capacity at a single node, it is slow to complete the training. It took *26.4 seconds* to train the model completely and resulted in a test accuracy of *87%*.

In the *distributed classifier*, we had 3 nodes: node 1 as parameter server, and nodes 2 and 3 as worker nodes. Once the parameter server script blocks are ready, the two worker nodes start training the model with 30,000 images each. The training time for each worker node is noted to be only *14.11 seconds* (as compared to 25.4 seconds of the basic classifier). The accuracy drops a bit, to *86.23%*, which is not a very large difference in accuracy, compared to the basic classifier. We note that there is some communication overhead in distributed machine learning classifier: since each epoch is followed by gradient being sent by each worker to the parameter server.

In the *federated learning classifier* however, we do not have each worker generating its gradient and writing to the parameter server. Here, 2 local models are trained on each worker node. The nodes behave as if they are the only nodes on which models are being trained. After a certain number of steps, the weights are sent to the parameter server for aggregation. The training time for each worker node here was noted to be *16.75 seconds*, which still significantly faster when compared to the basic classifier, with the accuracy being *85.15%*.

The extra 2 seconds that the federated classifier takes when compared to the distributed classifier (as shown in Table 1), is because of the additional step federated learning classifier which requires averages to be applied to the two local models that were trained previously. Thus, as a consequence, the distributed classifier produces a single model, while the federated learning classifier produces 2 local models that are aggregated at the end.

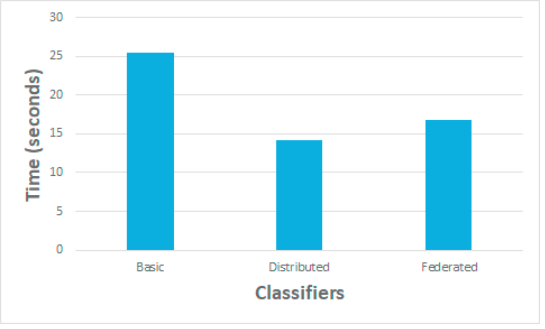


Figure 3: Time Comparison between the classifiers

Table 1: Performance comparison of the classifiers.

Parameters	Basic	Distributed	Federated
Time(seconds)	26.40	14.11	16.75
Accuracy	87%	86.23%	85.15%

5. Conclusion

In this paper, we analyzed the issue of training on the private data by evaluating 3 different approaches - basic, distributed and the federated learning classifiers. Although there wasn't a significant difference in the accuracy of the three models trained (since the underlying model to train was the same), there was a considerable difference in the time taken to build the models. While the basic machine learning classifier took 25.4 seconds to train, the distributed machine learning classifier was the fastest and required only 14.1 seconds to converge. The second fastest was the federated learning classifier which took 16.7 seconds. This difference in the distributed machine learning classifier and the federated learning classifier was identified and attributed to the extra step involved during aggregation of the model. The result is in agreement with the notion that with more and more data, we must distribute the data and compute into several smaller tasks for good performance. Federated learning is very promising as it is able to train models on a user's device, without sharing the raw data, thereby preserving privacy. Not just that, it enables edge devices to collaboratively learn a shared prediction model. It is effectively able to decouple the training of effective machine learning models from the need of sending and/or storing data on a central server. Thus, among the three models, federated learning is the best model to preserve privacy while training accurate models on distributed devices.

References

[1] Yang, Qiang, et al. "Federated machine learning: Concept and applications." ACM Transactions on Intelligent Systems and Technology (TIST) 10.2 (2019): 12.

[2] Al-Rubaie, Mohammad, and J. Morris Chang. "Privacy-Preserving Machine Learning: Threats and Solutions." IEEE Security and Privacy 17.2 (2019): 49-58.

[3] Bonawitz, Keith, et al. "Towards federated learning at scale: System design." arXiv preprint arXiv:1902.01046 (2019).

[4] Smith, Virginia, et al. "Federated multi-task learning." Advances in Neural Information Processing Systems. 2017.

[5] Papernot, Nicolas, et al. "Towards the science of security and privacy in machine learning." arXiv preprint arXiv:1611.03814 (2016).

[6] Bonawitz, Keith, et al. "Practical secure aggregation for privacy-preserving machine learning." Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017.

[7] Bost, Raphael, et al. "Machine learning classification over encrypted data." NDSS. Vol. 4324. 2015.

[8] Kamarinou, Dimitra, Christopher Millard, and Jatinder Singh. "Machine learning with personal data." Queen Mary School of Law Legal Studies Research Paper 247 (2016).

[9] Yu, Shui. "Big privacy: Challenges and opportunities of privacy study in the age of big data." IEEE access 4 (2016): 2751-2763.

[10] Duchi, John C., Michael I. Jordan, and Martin J. Wainwright. "Privacy aware learning." Journal of the ACM (JACM) 61.6 (2014): 38.

[11] Wang, K., et al. "Privacy-preserving data publishing: A survey on recent developments." ACM Computing Surveys (2010).

[12] McMahan, H. Brendan, et al. "Federated learning of deep networks using model averaging." (2016).

[13] Konen, Jakub, et al. "Federated learning: Strategies for improving communication e fficiency." arXiv preprint arXiv:1610.05492 (2016).

[14] Konen, Jakub, et al. "Federated optimization: Distributed machine learning for on-device intelligence." arXiv preprint arXiv:1610.02527 (2016).

[15] Du, Wenliang, and Zhijun Zhan. "Building decision tree classifier on private data." Proceedings of the IEEE international conference on Privacy, security and data mining-Volume 14. Australian Computer Society, Inc., 2002.

[16] Vaidya, Jaideep, and Chris Clifton. "Privacy-preserving decision trees over vertically partitioned data." IFIP Annual Conference on Data and Applications Security and Privacy. Springer, Berlin, Heidelberg, 2005.

[17] Vaidya, Jaideep, and Chris Clifton. "Privacy preserving naive bayes classifier for vertically partitioned data." Proceedings of the 2004 SIAM International Conference on Data Mining. Society for Industrial and Applied Mathematics, 2004.

[18] Vaidya, Jaideep, and Chris Clifton. "Privacy preserving association rule mining in vertically partitioned data." Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002.

[19] Vaidya, Jaideep, and Chris Clifton. "Privacy-preserving k-means clustering over vertically partitioned data." Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003.

[20] Yu, Hwanjo, Jaideep Vaidya, and Xiaoqian Jiang. "Privacy-preserving svm classification on vertically partitioned data." Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, Berlin, Heidelberg, 2006.

[21] Yu, Hwanjo, Xiaoqian Jiang, and Jaideep Vaidya. "Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data." Proceedings of the 2006 ACM symposium on Applied computing. ACM, 2006.

[22] Wan, Li, et al. "Privacy-preservation for gradient descent methods." Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2007.

[23] Wan, Li, et al. "Privacy-preservation for gradient descent methods." Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2007.

[24] Goldreich, Oded, Silvio Micali, and Avi Wigderson. "How to play any mental game." Proceedings of the nineteenth annual ACM symposium on Theory of computing. ACM, 1987.

[25] Yao, Andrew Chi-Chih. "Protocols for secure computations." FOCS. Vol. 82. 1982.