Deep Learning on Private Data

M. Sadegh Riazi, Bita Darvish Rouhani, and Farinaz Koushanfar
University of California San Diego

Emerging complex deep neural networks require a large amount of data to achieve a high precision. However, the high-volume of data is often collected from user's logs and personal data which contains sensitive information about individuals. We provide a summary of recent cryptographic methodologies for provably privacy-preserving deep learning and inference.

With the ever-increasing volume of data, powerful machine learning models are needed to efficiently process the vast amount of data in different applications and industries. Hierarchical deep neural networks, also known as Deep Learning (DL), provide efficient modeling methodologies for automatic extraction of the underlying features in the content. In several important and frequent tasks, including but not limited to visual classification, game playing, and speech recognition, DL achieves a superior accuracy in comparison with the human cognition.

DL in a supervised setting involves two distinct phases: training and inference (prediction). Figure 1 illustrates the high-level block diagram of the learning process for devising a typical machine learning system. The performance of a DL model is directly dependent on the volume of the available training data. However, the training samples are usually collected from users' content stored on the cloud servers which contain sensitive information such as image, voice, location log, and medical record. The users' privacy is of critical concern not only during training but also for inference. Internet companies are now providing machine learning as a service where users can send their queries to the cloud servers and receive the prediction result by paying certain fees. These queries can include sensitive data such as users' health records sent to the remote DL servers for medical diagnosis. One naive solution to mitigate the risk of data leakage is to have the consumers download the model and run it on their own trusted platform. This solution is not attainable in real-world settings because the DL model is learned by processing massive amounts of training data. As such, the trained AI models are usually considered a sensitive Intellectual Property (IP);[1] companies require confidentiality to preserve their competitive advantage.

In a nutshell, three main requirements should be considered in the privacy-preserving deep learning: (i) during the training phase, users' sensitive data should not be revealed to the central server who is training the DL model, (ii) during inference, users' query should not be revealed to the server, and (iii) server's proprietary DL model should not be revealed to the user.

It is highly desirable to devise privacy-preserving frameworks in which neither of the participating parties reveal their information during the joint computation. In theory, any function can be evaluated on private inputs from two or more distinct parties using the Secure Function Evaluation (SFE) protocols. However, a naive transformation of a particular function using the generic SFE protocols incurs between one to three orders of magnitude overhead both in terms of computation and communication. To empower privacy-preserving computation in the context of deep learning, it is crucially important to devise novel domain-specific customized techniques that can reduce the complexity of SFE protocols.

This article presents a systemization of knowledge of the most recent privacy-preserving frameworks for DL and provides a comprehensive comparison in terms of the unique properties and performances on standard benchmarks. We also include a summary of the recent attacks and illustrate how they can be prevented by utilizing the privacy preserving frameworks. The yellow box on the next page provides a glossary of the low-level building blocks used to compose higher-level protocols.

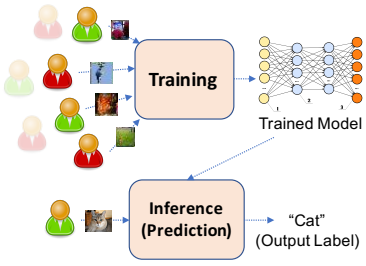

Figure 1. Overview of training and inference in deep learning.

Glossary of Low-Level Building Blocks Leveraged for Privacy-Preserving DL

Garbled Circuits (GC)[1] is one of the generic Secure Function Evaluation (SFE) protocols. GC enables two parties, Alice and Bob, to jointly compute a function on their private inputs without revealing anything but the outcome of the function. The first step in utilizing this protocol is to describe the function as a *Boolean circuit* with two-input logic gates. Alice then starts assigning random keys to each wire in the circuit. For each gate, she encrypts the output keys of the gate using the corresponding input key pairs and forms garbled tables. She sends all of the garbled tables as well as the input keys associated with her input. Bob also acquires the correct keys associated with his input and decrypts each gate one by one until he finds the output keys. Finally, Alice provides the mapping between output keys to the true semantic values to generate the plaintext output of the function. Note that the number of interactions between Alice and Bob is *constant* and independent of the function.

Goldreich-Micali-Wigderson (GMW)[2] is another generic SFE protocol. Similar to the GC protocol, GMW requires that the function is described as a Boolean circuit. However, unlike GC, two parties need to interact for every AND gate. Therefore, by processing all independent AND gates in parallel, the communication round complexity is *linear* with respect to the depth of the circuit. The GMW protocol only requires small communication for each gate compared to the GC protocol.

Secret Sharing (SS) is a method to distribute a secret to two or multiple parties where each share does not provide *any* information about the secret but the secret can be reconstructed from the shares. One of the most popular SS variants is additive SS. In this case, a secret is shared by sampling random numbers from $\mathbb{Z}_{2^l}$ and creating the last share such that summation of all shares yield the secret value. The secret can be reconstructed by adding all of the shares.

Homomorphic Encryption (HE) is a cryptographic primitive that enables one party to encrypt data and send it to another party who can then perform certain operations on the encrypted version of the data. Upon completion of the computation, the encrypted version of the result is sent back to the first party who can decrypt and get the result in plaintext. HE methods can be categorized into Partially HE and Fully HE. For example, the Paillier cryptosystem[3] only supports addition on the encrypted version of two numbers and is a partially HE. In contrast, a fully homomorphic encryption scheme[4] supports an arbitrary functional logic.

Differential Privacy (DP)[5] is a metric that defines how much information about a single entry in a database is revealed upon a query to the database. To preserve the privacy of database entries, a carefully-chosen noise is added to the database such that the statistical properties of the database are preserved while each data point is changed due to the added noise. Equivalently, DP can be seen as a way to reduce the dependency between the outcome of a query and individual data points in the database, thus, reducing the information leakage.

References

1. Yao, Andrew Chi-Chih. "How to generate and exchange secrets." Foundations of Computer Science, 1986., 27th Annual Symposium on. IEEE, 1986.
2. 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.
3. Paillier, Pascal. "Public-key cryptosystems based on composite degree residuosity classes." Eurocrypt. Vol. 99. 1999.
4. Gentry, Craig. "Fully homomorphic encryption using ideal lattices." STOC. Vol. 9. No. 2009. 2009.
5. Dwork, Cynthia. "Differential privacy." Encyclopedia of Cryptography and Security. Springer US, 2011. 338-340.

Deep Neural Networks

Deep Learning (DL) is a sub-class of machine learning that resembles the functionality of a human brain. There are different variants of deep learning architectures. However, all of them have a layered structure, starting from an input layer and ending with an output layer. There can be one or more layers in between which are called hidden layers. The output of a given layer is the input to the next layer. Once a new data is given as input to the neural network, the output of each layer is computed until reaching the final output layer. In most cases, the output values represent the probabilities of different classes that the input can belong to. Popular main categories of the neuron networks that are widely used in deep learning are Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs).

DNN is a more generic architecture used in various applications. The most computationally expensive parts of a DNN are the matrix mul-

tiplication, a.k.a., Fully-Connected (FC) layers. In addition, DNNs have activation layers which are the only non-linear part of their structure. Popular activation functions include Rectified Linear Unit (ReLu), logistic sigmoid, and Tangent-hyperbolic. The output layer is usually a softmax layer that translates the output of the last hidden layer to a probability vector. In addition to the aforementioned layers, CNNs have (i) convolution layer which are a weighted sum of different segments of the previous layer. These weights as well as the weights in the fully-connected layer are learned in the training phase of DL. (ii) Pooling layers select either the average (mean-pooling) or maximum (max-pooling) of different segments in the previous layer. CNNs are specifically used where the input data has spacial correlation such as face recognition systems. In contrast to DNNs/CNNs, RNNs have internal state. This characteristic makes RNNs suitable for applications where a sequence of input data should be processed. RNNs are widely used for speech recognition.

Delivering an efficient solution for privacy-preserving deep learning has been an active research area in recent years. Training and inference are inherently of two different natures, thus, the proposed frameworks can be categorized based on whether they support training or inference. In training, the data is usually distributed among many parties and a central entity, e.g., an Internet company, wishes to learn the model on all the data samples. Therefore, training the model without revealing each party's input is essentially a *multi-party* problem. In contrast, the inference is a *two-party* problem in which a party holds an already trained model and another party has a private input.

Privacy-preserving deep learning frameworks can also be categorized based on the family of the utilized secure computation protocols, i.e., Homomorphic Encryption (HE), Garbled Circuits (GC), Secret Sharing (SS), and Goldreich-Micali-Wigderson (GMW). While all of these methods have provable privacy properties, each one of them has specific characteristics and capabilities. In addition to secure computation approaches, Differential Privacy (DP) has been suggested as a solution for *mitigating* the information leakage when training a neural network. However, unlike SFE protocols, DP does not provide a perfect leakage-proof solution.

Survey of Private Training Frameworks

In what follows, we present a brief description of the most efficient frameworks for learning a deep neural network on private data. All discussed frameworks are secure in the Honest-but-Curious (HbC) adversary model in which all parties are assumed to follow the protocol but they might attempt to infer more information based on the protocol transcript (all the data that they send and receive during the protocol execution). HbC adversary model is a stepping stone towards more powerful models such as security against malicious adversaries in which any party can deviate from the protocol at any time.

Shokri and Shmatikov[2] propose a privacy-preserving method based on Differential Privacy (DP) for collaborative deep learning when the data is distributed among different parties. In this scenario, each party locally trains her own version of the neural network and selectively shares some of the updated parameters with other parties. The intuition is that optimization algorithms can be run in parallel by different individuals and the results can be aggregated later on. In order to further mitigate the information leakage, they apply DP when sharing the parameters instead of sharing the raw values. As a result, they introduce a trade-off between the accuracy of the trained neural network and the privacy of the data.

SecureML[3] is a system for privacy-preserving machine learning in general, and neural networks in particular. The system is based on HE, GC, and SS protocols. Data owners secret share their data with two non-colluding servers which privately train the neural network. SecureML uses a customized activation function that is more efficient for training a neural network using secure computation protocols. At the end of the computation, the trained model is secret-shared between the two servers. In addition to training, SecureML also provides privacy-preserving inference.

Google[4] has introduced a protocol for secure aggregation of high-dimensional vectors held by different users. This protocols can be used in *federated learning* in which users hold their private database and models. A central server learns a machine intelligence model by securely aggregating the learning updates from users. The approach is based on Shamir's secret sharing and is robust against users exiting the protocol at any time.

We do not quantitatively compare the solutions for privacy-preserving training since the methodologies have different security guarantees and computational models. Therefore, comparing the training times is not meaningful.

Framework	Preserving Accuracy	Independence of Secondary Server	Data &Network Embedding	Support for Batch Processing	Fixed-Point Operations	Scalability for Large DL Models	Non-linear Act. and Pooling functions	Cryptographic Protocol(s)
CryptoNets	✗	✓	✗	✓	✗	✗	✗	Leveled HE
SecureML	✗	✗	✗	✗	✓	✓	✗	Linearly HE, GC, SS
MiniONN (w/ Sqr Act.)	✗	✓	✗	✗	✓	✓	✗	Additively HE, GC, SS
MiniONN (w/ Relu & Pooling)	✓	✓	✗	✗	✓	✓	✓	
DeepSecure	✓	✓	✗	✗	✓	✓	✓	GC
DeepSecure (+ Preprocessing)	✗	✓	✓	✗	✓	✓	✓	
Chameleon	✓	✗	✗	✗	✓	✓	✓	GMW, GC, SS

Table 1. Characteristics of different frameworks that enable privacy-preserving inference and their underlying cryptographic primitives.

Survey of Private Inference Frameworks

In the past few years, privacy-preserving inference for an *already trained* neural network has been the main research focus. In this article, we discuss the suggested methods in a chronological order. The high-level comparison of the frameworks is provided in Table 1 while the quantitative comparison of their performance is shown in Table 2.

CryptoNets,[5] developed by Microsoft Research, leverages Leveled Homomorphic Encryption (LHE)[6] (a variant of partially HE). Since non-linear activation functions cannot be realized using LHE, the authors propose to approximate the activation functions using low-degree polynomials. Therefore, the neural network should be retrained in plaintext using the same activation function in order to maintain a good prediction accuracy. Another shortcoming of this approach is that there is a certain limit on the number of sequential multiplications imposed by LHE which makes the solution prohibitive for deep (more than a few layers) neural networks. In addition, CryptoNets has a privacy/utility tradeoff: in order to achieve a higher level of privacy, accuracy must be reduced within the same computing capabilities. They report experimental results on the MNIST dataset which is a collection of hand-written digits, each represented as a 28by28 pixel image where each pixel takes a gray-scale value between 0 and 255. To perform a single private inference, CryptoNets takes 298s and requires 372MB of communication. As we mentioned, SecureML also supports the private inference. SecureML reduces the prediction latency to 4.9s in a similar setting and comparable neural network architecture.

MiniONN[7] improves upon SecureML and reduces the latency of private inference from 4.9s to 1s for a similar network using polynomial activation functions. MiniONN leverages additive HE, GC, and SS and supports non-linear activation functions as well as max-pooling required by the convolutional neural networks. MiniONN has two main phases: (i) an offline phase based on additive HE to precompute random shares that are independent of the actual inputs and (ii) an online phase based on garbled circuits and secret sharing. Linear layers are processed using SS and GC is utilized for non-linear layers. Switching between different protocols requires secure secret type conversion. All the intermediate values are kept as secret shares between two parties until the final prediction is computed.

DeepSecure[8] is one of the recent frameworks that is based on the GC protocol. Since GC is a generic SFE protocol, the framework supports all of the non-linear activation functions as well as pooling operations. DeepSecure introduces the idea of reducing the size of the data and the network by a *preprocessing* stage before executing the GC protocol, thus, compacting the computation and communication by up to two orders of magnitude. The preprocessing stage is independent of the underlying cryptographic protocol and can be adopted by any other back-end engines for private inference. DeepSecure also supports secure outsourcing of the computation to a secondary server when the client is resource-constrained. In the next section, we further elaborate on the preprocessing stage and secure outsourcing mode.

Chameleon[9] is a mixed-protocol framework for privacy-preserving machine learning. It utilizes the GMW protocol for low-depth non-linear activation functions and GC for more complicated non-linear activation functions (e.g., Sigmoid) as well as for pooling layers. All arithmetic operations such as addition and multiplication are performed using secret sharing. Similar to its concurrent work, MiniONN, Chameleon

Framework	Prediction Timing (s)			Communication (MB)			Accuracy (%)	Deep Neural Network Architecture
	Offline	Online	Total	Offline	Online	Total		
CryptoNets	-	297.5	297.5	-	372.2	372.2	98.95	Conv - Sqr Act - MeanP - Sqr Act - FC
SecureML	4.7	0.18	4.88	-	-	-	93.1	FC - Sqr Act - FC - Sqr Act - FC
MiniONN (w/ Sqr Act.)	0.90	0.14	1.04	3.8	12	15.8	97.6	
MiniONN (w/ Relu & Pooling)	3.58	5.74	9.32	20.9	636.6	657.5	99	Conv - ReLu - MaxP - ReLu - MaxP - FC - ReLu - FC
DeepSecure	-	9.67	9.67	-	791	791	99	Conv - ReLu - FC - ReLu - FC
DeepSecure (+ Preprocessing)	-	1.08	1.08	-	88.2	88.2	99	
Chameleon	1.34	1.36	2.70	7.8	5.1	12.9	99	

Table 2. Performance comparison of different frameworks for privacy-preserving inference (prediction) on the MNIST dataset given similar architectures and computational platforms.

has offline and online computation phases. Most of the computation is shifted to the offline phase to provide a fast online prediction phase. The offline phase comprises creating the correlated randomness used in the online phase for all three protocols, i.e., GC, GMW, and SS. These random (but correlated) shares are independent of the actual inputs and the functionality that is being evaluated and they have to be created by a third party. Similar to SecureML, Chameleon requires the presence of two non-colluding servers. However, in contrast to SecureML, the third party is not involved in the online phase. Moreover, the functionality of the third party can be replaced using the Intel Software Guard Extension (SGX)[10] and achieve a two-party computation model in which case the underlying assumption is that Intel is trusted. The GMW protocol in Chameleon supports Single Instruction Multiple Data (SIMD) which creates a way to process the same operation on multiple data considerably faster. In addition, Chameleon introduces a new and efficient vector dot product protocol which can accelerate the private execution of almost all machine learning applications. To the best of our knowledge, Chameleon achieves the best performance results thus far. However, without secure hardware such as SGX, it would require three parties for private inference.

Another differentiator between the aforementioned frameworks is their ability to support popular non-linear activation functions. Frameworks such as CryptoNets and SecureML propose customized activation functions while other frameworks such as MiniONN, DeepSecure, and Chameleon support regular non-linear activation functions, e.g., ReLu and Sigmoid. As a result, the second group does not require a modification to the training phase. That is, they can provide the privacy-preserving prediction on client's input based on an already trained neural network.

Batch Processing

The CryptoNets framework supports processing multiple inputs (up to 8192) at the cost of a single input: providing the notion of SIMD computation at the data-level. This, in fact, amortizes the cost when the server can batch and process multiple queries at the same time. The optimization is enabled by embedding multiple inputs in the same homomorphic ciphertext and processing the single ciphertext at the server side. However, performing the SIMD operation requires batching a very high number of inputs and processing them at once. While this results in a high throughput, the delay experienced by individual users can be significant. Note that the computation/communication costs of almost all other frameworks scale linearly with the number of inputs. As a result, there exists a *break-even* point at which CryptoNets starts having a higher throughput. For example, in case of a single image, DeepSecure is 58-fold faster than CryptoNets. However, when processing batches of size $\sim$ 2600, CryptoNets reaches the same throughput as DeepSecure. For batch data sizes larger than 2600, CryptoNets can be more efficient (compared to DeepSecure) but this is uncommon in distributed application settings. SIMD operations are possible when the server is processing multiple inputs for the *same* network architectures. In contrast, frameworks that have lower processing delays are suitable for real-time query applications where the low-delay is crucial. Note that independent of the batch sizes, the major drawback of CryptoNets is its limited accuracy.

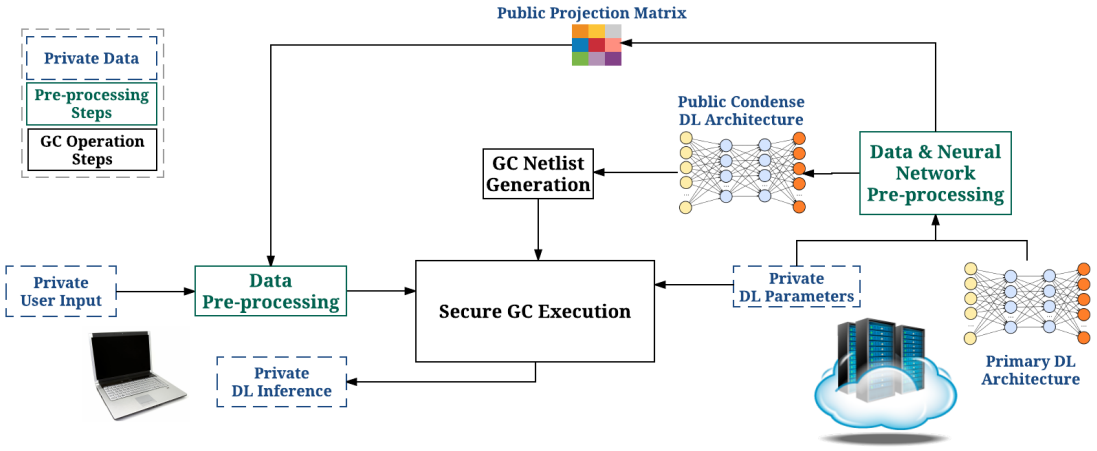

Figure 2. The internal architecture of the DeepSecure framework comprising core GC execution engine and the preprocessing steps. The private inputs and outputs are depicted as blue boxes.

Data and Network Preprocessing

DeepSecure introduces the idea of preprocessing the network and the input data before engaging in the secure computation protocol.[8] It is shown that such low-overhead preprocessing stage can reduce the subsequent computation and communication on encrypted data by up to two orders of magnitude. Figure 2 illustrates the architecture of the end-to-end system. The preprocessing comprises two different aspects: (i) data projection, and (ii) DL network distillation. Many modern data collections that are not even inherently low-rank can be modeled by a composition of multiple lower-rank subspaces. This composition of data as an ensemble of lower dimensional subspaces can significantly reduce the size of the data that is given as input to the secure computation protocol. The second preprocessing aspect is the DL network distillation. Recent empirical advances in DL has shown the importance of sparsity in training the DL model. As a result, DeepSecure suggests condensing the underlying DL network prior to the GC protocol execution. This is realized by modifying the Boolean circuit that corresponds to the DL network in the GC netlist generation step (see Figure 2). Note that the preprocessing stage is performed in plaintext and incurs negligible computation cost compared to the cryptographic protocols. Moreover, the DL network distillation is a one-time effort and the cost is amortized over many private predictions.

Securely Outsourcing the Computation

In scenarios where the client (input holder) is very resource-constrained such as mobile phones and embedded devices, it is favorable to outsource the computation to a secondary server. While there can be many solutions to this problem, a simple method is to use the XOR-sharing technique adopted by DeepSecure (outsourcing mode[8]). In this method, a client who owns the input data only has to XOR her original input with a random binary vector of the same size as input. She sends the XORed result to one of the servers and the random vector to the secondary server. Both servers then execute the GC protocol and send back the shares of the result. The client only has to XOR the final shares to reconstruct the true prediction result. In the outsourcing mode, the client only needs to perform the XOR operation which has a negligible computation cost. Outsourcing is well-suited for scenarios where the client is resource constrained, e.g., Internet of Things (IoT) in which the edge nodes usually have limited computational power. The security model is based on the non-collusion assumption of the servers. One of the disadvantageous of FHE is that it does not support XOR-sharing technique and incurs a high computational cost at the client side.

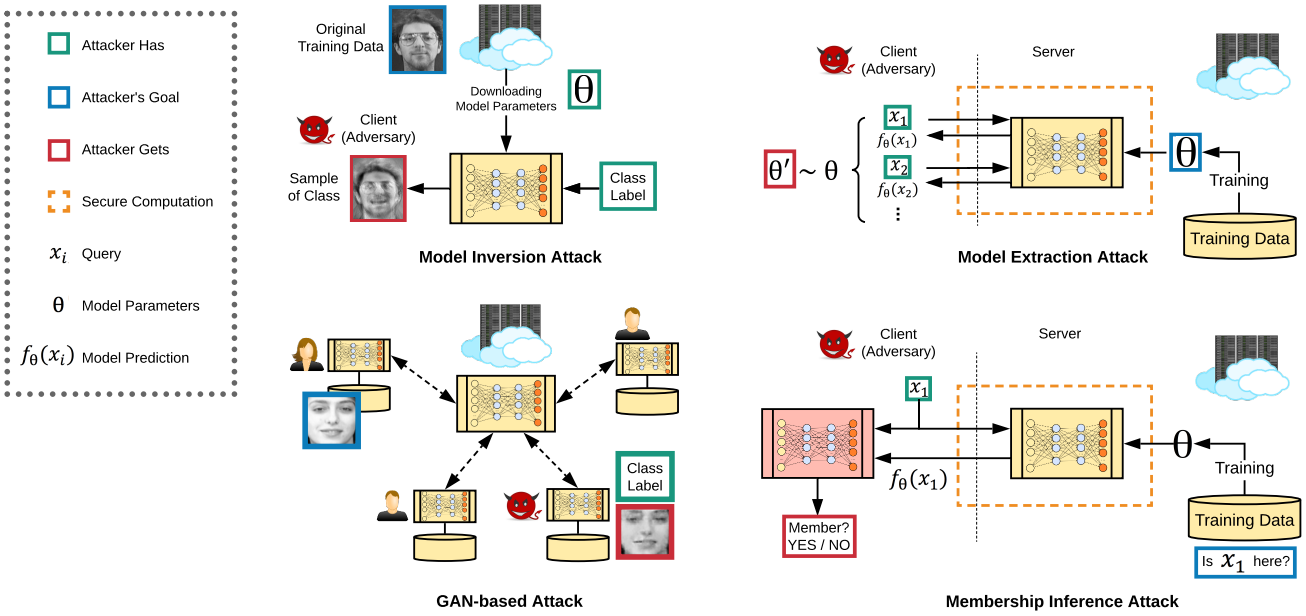

Figure 3. An overview of attacks on deep learning: scenarios, adversary's abilities, and goals.

Attacks on Deep Neural Networks

We survey four of the most important attacks on deep learning. The first three are designed for inference phase where there is a server holding the trained model and a client querying the model. Depending on the attack, the adversary may have either *black-box* or *white-box* access to the model. In the black-box scenario, an adversarial client can only query the server and receive the prediction result. The prediction can comprise the label with (or without) the confidence values for each class. In the white-box scenario, the attacker is able to download the model and acquire complete network parameters. The fourth attack is designed for the training phase where multiple parties want to jointly train a model on their private inputs. Figure 3 provides an overview of all four attacks.

Model Inversion[11] attack extracts information about the training data used for learning the model. For a given class, the attacker with either black-box or white-box access to the model attempts to create an input that maximizes the confidence score of that class using the gradient descent algorithm. Figure 3 illustrates the attack for the white-box access scenario. The generated input is a prototypical sample of that class. For example, in case of a facial recognition system, an attacker can produce an approximate image of one of the people whose image was used in the training phase by only having her name. Model inversion attack does not depend on how the network was trained and reveals information about the average features of a given class. In order to mitigate this attack in the black-box access scenario, it is suggested to round the confidence score before the prediction results are sent back to the client. It is not known how to eliminate the information leakage in the white-box scenario without diminishing the model performance.

Model Extraction[12] attack attempts to learn an approximation of a server model by querying the model many times. In the client-server computation model, a malicious client creates its own version of the model with the incentive of undermining the pay-per-prediction fees (stealing the model). Model extraction can also be used as a step towards model inversion: the attacker first learns an approximation of the model and then attempts to reconstruct the training data. The model can be extracted by n times querying and solving a linear/non-linear system of n equations where n is the total number of model parameters. However, similar to model inversion,

this attack can be prevented by not providing the confidence vector and only reporting the final class label that has the highest confidence value.

In **Membership Inference**[13] attack, the goal is to identify whether a specific record has been used in the training phase or not. There are certain privacy concerns that arise from this attack. For example, membership inference attack on a classifier that suggests medications for a given disease can reveal that a target person, whose information was used in the training process, actually has that disease. The intuition behind the attack is that DL models can perform better on the data records used in the training phase compared to the data that they have never seen. In this case, the model is *overfitting* the training data. Such difference in the behavior of the model enables membership inference attack. There are two main mitigation strategies: (i) during the training phase, *regularization* techniques should be used to avoid overfitting and, (ii) during the inference phase, the complete confidence vector should not be revealed to the client.

Hitaj et al.[14] crafted an attack based on Generative Adversarial Networks (GANs) to infer sensitive information in a collaborative deep learning. More specifically, they showed an adversarial client can learn the training samples held by other clients even in the privacy-preserving collaborative deep learning setting suggested by Shokri and Shmatikov.[2] GANs generate samples that look similar to the training data without having access to the training data itself. A GAN produces samples by only interacting with a discriminative deep neural network. The goal of GAN is to deceive the discriminative model that the produced sample is real and the goal of the discriminative model is to detect the synthetic samples. The process continues until the discriminative model cannot perform better than the random guess. In general, Hitaj et al. showed that applying record-level differential privacy in collaborative deep learning is not effective. They suggest that more effective solutions based on secure multi-party computation and homomorphic encryption are needed to guarantee the privacy of data owners.

Note that the goal of private inference frameworks is to preserve the privacy of *inputs* that are provided by the client and server. In the model inversion attack with a white-box access, the network parameters are downloaded by the client, hence, the attack is out of the scope of private inference frameworks. In the black-box setting, the attack can be mitigated by adding a layer inside the secure computation protocol to round the confidence scores before sending the result back to the client. Similarly, model extraction and membership inference attacks can be effective if the prediction confidence vector is sent to the client. Private inference frameworks that support non-linear functionalists[7–9] can also filter the output and only report the label of the class that has the highest confidence score. In this case, the confidence vector is not revealed to the client and the aforementioned attacks will become infeasible.[11]

Privacy is one of the biggest concerns when learning (and using) the neural network models on the users' sensitive information. This article provides a systematic view and comparison of the latest efficient privacy-preserving solutions for learning and inference of deep neural networks. The methodologies rely on various building blocks such as garbled circuits, homomorphic encryption, secret sharing, and differential privacy. We have also outlined various attacks on deep learning. Several of these attacks rely on the confidence vector provided by the cloud server. Hence, if only the output label is provided to the client, such attacks can be prevented. Techniques such as (i) devising mixed-protocol solutions,[7,9] (ii) pre-processing the network and data,[8] and (iii) using correlated randomness[9] can significantly reduce the overhead of secure computation protocols.

During past few years, more focus has been on the private inference than private learning. To bridge the wide gap between machine learning on plaintext and encrypted data, more research for finding efficient customized privacy-preserving learning techniques is needed.

References

1. Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *IJSN*, 2015.

2. Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *CCS*. ACM, 2015.

3. Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE S&P*, 2017.

4. Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving

machine learning. In *CCS*. ACM, 2017.

5. Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*, 2016.

6. Joppe W Bos, Kristin E Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *IMA*. Springer, 2013.

7. Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *CCS*. ACM, 2017.

8. Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. DeepSecure: Scalable provably-secure deep learning. In *DAC*. IEEE, 2018.

9. M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *ASIACCS*. ACM, 2018.

10. Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. Secure multiparty computation from SGX. In *FC*, 2017.

11. Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *CCS*. ACM, 2015.

12. Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *USENIX Security*, 2016.

13. Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *S&P*. IEEE, 2017.

14. Briland Hitaj, Giuseppe Ateniese, and Fernando Pérez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In *CCS*. ACM, 2017.

M. Sadegh Riazi is a Ph.D. student in the Department of Electrical and Computer Engineering at the University of California San Diego. Contact him at mriazi@ucsd.edu.

Bita Darvish Rouhani is a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of California San Diego. Contact her at bita@ucsd.edu.

Farinaz Koushanfar is a Professor and Henry Booker Faculty Scholar in the Department of Electrical and Computer Engineering at the University of California San Diego. Contact her at farinaz@ucsd.edu.