Privacy in Neural Network Learning: Threats and Countermeasures

Shan Chang and Chao Li

Abstract

Algorithmic breakthroughs, the feasibility of collecting huge amount of data, and increasing computational power, contribute to the remarkable achievements of NNs. In particular, since Deep Neural Network (DNN) learning presents astonishing results in speech and image recognition, the amount of sophisticated applications based on it has exploded. However, the increasing number of instances of privacy leakage has been reported, and the corresponding severe consequences have caused great worry in this area. In this article, we focus on privacy issues in NN learning. First, we identify the privacy threats during NN training, and present privacy-preserving training schemes in terms of using centralized and distributed approaches. Second, we consider the privacy of prediction requests, and discuss the privacy-preserving protocols for NN prediction. We also analyze the privacy vulnerabilities of trained models. Three types of attacks on private information embedded in trained NN models are discussed, and a differential privacy-based solution is introduced.

Introduction

An Artificial Neural Network (ANN), also called a Neural Network (NN), is a computational model inspired by the structures and functions of biological neural networks constituting human brains, acting as the underlying model of artificial intelligence, which is, not surprisingly, the hottest research area nowadays. An ANN is composed of a collection of connected artificial neurons, attempting to recreate the computational mirror of a biological neural network, in which a biological neuron receives inputs from other neurons, combines them in some way, performs a non-linear operation on the result, and then outputs the final result. The huge numbers of neurons and the multiple connections between them endow human beings with the ability to remember, think, and apply previous experiences to actions. Benefitting from the ability that extracts advanced features from original data to effectively represent input space, those NN methods have been widely applied on a variety of complex tasks, including driver assistance, face recognition, medical diagnosis, AI gaming, and so on, and the accuracy of NNs outperforms human beings in many domains.

Big companies like Google, Apple and Amazon have exploited a tremendous amount of information collected from their customers and super-powerful computational abilities of GPU farms to implement Deep Neural Network (DNN) on a large scale. Alternatively, many NN-based machine learning services have been provided for generating tailored models for their customers (e.g., Google Cloud's AI), such that one customer holding a set of data and data classification requirements can upload the dataset to the service, to construct models available to the customer. Customers can use the model through a new service paradigm of Machine-Learning-as-a-Service (MLasS), which uses cloud infrastructures offering online prediction services to customers, or can download the model and run the prediction phase on the client-side.

Despite the tremendous business and technical advantages of NNs, privacy concerns have received more and more attention for four reasons. First, the training data of individuals or companies contain sensitive or proprietary information, such as financial records, health information, and social network profiles. Once those data are contributed or outsourced to prediction service providers for modeling, the owners can no longer control how they are used. It is not front-page news that companies leak private data of their customers for getting benefits or suffering hacking. Second, a trained model incorporates essential features about its training dataset, which implies that it is possible for a malicious user to extract sensitive information from the model. For example, Fredrikson *et al.* [1] introduced a *model inversion* attack, and demonstrated its feasibility by recovering images from NN-based facial recognition applications. Third, cloud-based prediction services put the privacy of users at risk since the prediction requests submitted to remote services may contain sensitive information. Finally, the knowledge, that is, the expertise, of service providers embedded in NN models may also be learned, for example, by adaptively feeding model crafted input samples.

Recently, many research results have demonstrated that attacks launched against NNs can successfully recover private information embedded in them. Although several privacy-preserving solutions have been proposed in the field of NN learning, the research on NN privacy is still at an early stage. Hence, in this article we investigate privacy issues of NNs and review advances in privacy protecting. The organization of this article is as follows. First, we provide preliminaries of NN learning in order to facilitate understanding of the threat model and attacks. Then we review the capabilities of inside attackers who take part

Shan Chang (corresponding author) and Chao Li are with Donghua University.

Digital Object Identifier:
10.1109/MNET.2018.1700447

in model training. In particular, we introduce an active attacking methodology reported recently. We also present several privacy-preserving NN training schemes in centralized and distributed approaches. In the next section we describe privacy concerns during the prediction phase, as well as attacks launched against trained models by both black-box and white box attackers, and we discuss the existing techniques to eliminate or mitigate potential privacy leakage.

Neural Network Learning

NNs are remarkably effective tools to learn complicated relationships (i.e., parameterized functions) from high dimensional input to output data. Figure 1 shows a diagram of a typical neural network with three layers organized as a bottom-up pipeline. Circles are *neurons*, representing computational units. Neurons are connected through weighted directed edges, which start from bottom (input) layer neurons, and end up with top (output) layer neurons. Each neuron is associated with an activation function. Typically, we focus on supervised learning. Popular activation functions include *sigmoid*, *ReLU*, and *softmax* (applied at the output layer for multi-class classification). The input layer receives one signal (input) which will be transmitted to interneurons located in hidden layers. After applying linear transformations (weighting) and activation functions with one or multiple hidden layers, the signal is propagated to the output layer which will provide final output. By adjusting the weights, we can 'train' such a parameterized function to fit a given finite set of input-output training samples, and wish that the function can generalize to samples beyond the training set. Thus, applications of NN learning methods commonly have a two-phase paradigm: the training phase in which a model is trained with a certain training dataset, and a prediction phase in which the trained model is used to predict categories (classification) or continuous values (regression) given some input data (prediction requests). There exist the following two types of training methodologies (Fig. 2):

- Centralized Training: all training data are submitted to a third party, especially a company providing learning services.
- Collaborative Training: also known as distributed, federated or decentralized training. Parties contributing training data will also take part in the training procedure, i.e., participants coordinating with each other or with a central server to achieve the final model.

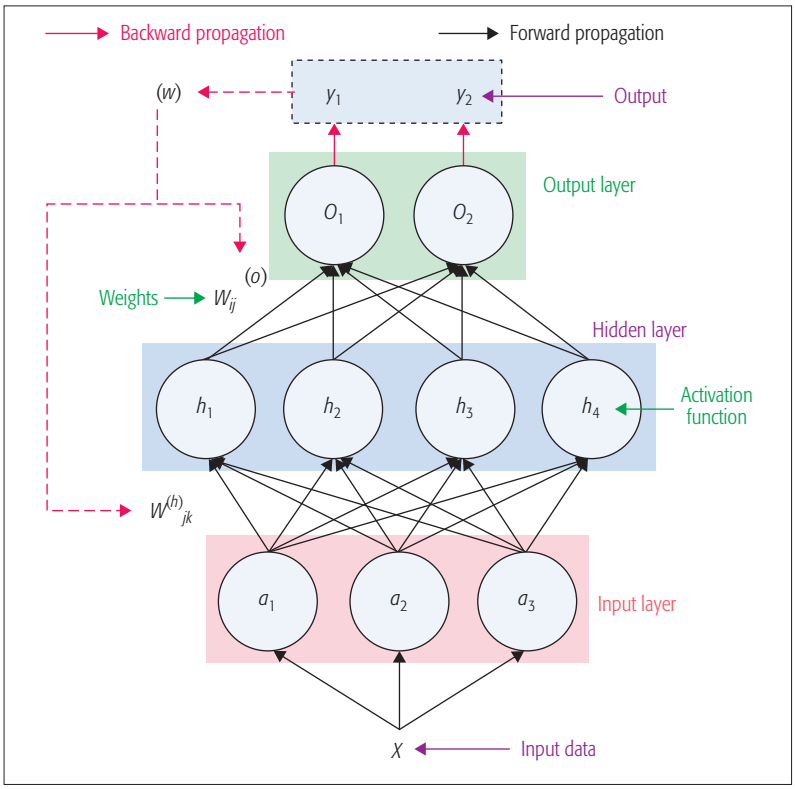

FIGURE 1. A three-layer neural network.

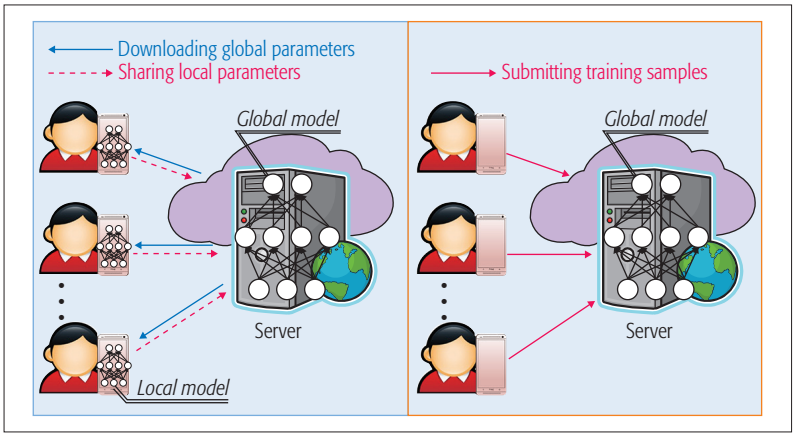

FIGURE 2. Approaches for centralized (right) and distributed (left) training.

During training phase, a loss function $\mathcal{L}$ is defined to represent the punishment for mismatching the training samples. The loss on a matrix of weights w is calculated by the average of the loss over a set of training samples $\{x_1, x_2, ..., x_n\}$, that is, $\mathcal{L}(w) = 1/n\Sigma_{i=1}^{n} \mathcal{L}(w, x_i)$, $i = 1, ..., n$. Training refers to finding w which yields hopefully the smallest $\mathcal{L}(w)$ under practice resource constraints, which is a nonlinear optimization problem. In practice, the mini-batch Stochastic Gradient Descent (SGD) algorithm is applied to solve such a problem. Gradient descent starts at a random point, that is, w is randomly initialized. At each step, first one randomly selects a small batch $\mathcal{B}$ of samples and calculates the loss $\mathcal{L}(w, \mathcal{B})$. The above procedure is called *forward propagation*. Then, an estimation of the global gradient $\nabla_w \mathcal{L}(w)$ is computed, by averaging gradients of samples in the mini-batch, that is, $\mathcal{G}(\mathcal{B}) = 1/\mathcal{B} \Sigma_{x \in \mathcal{B}} \nabla_w \mathcal{L}(w, x)$. w is updated following the gradient direction $-\mathcal{G}(\mathcal{B})$. The second phase is backward propagation. The two phases repeat until the algorithm converges to a local optimum.

During the prediction phase, a trained model is used to make predictions on inputs. Since the weight parameters w hold the model knowledge acquired by training, ideally the model should generalize and make accurate predictions for inputs outside of the domain explored during training. The model can be encapsulated into an online prediction service, that is, MLaaS, and users access the service through cloud infrastructures. An alternate is to allow users to download the model and to run the prediction phase on the cli-

ent sides. We consider a model for classification. For input $\tilde{x}$, the model produces the prediction vector of probabilities (confidence values) encoding its belief of $\tilde{x}$ being in each of the classes.

Privacy Issues During Training

The main privacy leakage during training comes from inside adversaries who take part in the training procedure. More specifically, companies providing learning services and participants in collaborative training are included. First, once training data are revealed to companies providing learning services, sensitive information might be recovered by the companies themselves. It is also possible that other entities can access the data through legal or extra-legal means, for example, by compromising the companies' storage. Second, in collaborative training scenarios, it should be considered that private information might also be leaked to other participants. In this section, we identify the attack models, attacking methodologies and, when possible, discuss solutions under both training methodologies.

Attack Models

Under both centralized and collaborative training scenarios, companies providing learning services are considered as *passive adversaries* who follow the semi-honest (also known as honest-but-curious) model. It means that the corresponding servers execute the pre-designed training algorithm honestly but attempt to learn or infer sensitive information from training datasets as much as possible. It is reasonable since the primary objective of those companies is to provide high quality services, that is, achieving accurate NN models.

On the other side, participants in collaborative training are not necessarily passive. An *active insider*, attempting to extract information about the training data it does not own, can surreptitiously influence or deviate from the training process. B. Hitaj *et al.* [2] proposed a Generative Adversarial Network (GAN) based attack on DNN, in which an inside attacker simply runs the collaborative learning algorithm and forces other honest participants into releasing more information about their private datasets, and reconstructs sensitive information.

Attacking Methodologies of Active Adversaries: B. Hitaj *et al.* [2] devised an attack on a collaborative deep learning procedure launched by its participants. An adversary can train a GAN to generate prototypical samples of a target training set that are meant be private. To the best of our knowledge, this is the first work to disclose the privacy vulnerability of collaborative deep learning. Since the adversary is one of the participants, it is reasonable that it has knowledge about the model structure and data labels of other participants. Furthermore, the adversary can observe the development of the model, and consequently, can work adaptively to maximize the benefit.

The power of the attack is that the adversary can learn a GAN model locally, to generate instances that look like the samples from victims. For example, the device of *Alice* (victim) stores pictures of herself. Although the GAN has never seen the pictures, it can reconstruct her face which is recognizable by human or facial recognition systems. To this end, the adversary takes part in the collaborative deep learning, and iteratively improves the GAN. The adversary first downloads parameters of the DNN from a central parameter server in order to update its local model. Then it trains a GAN model trying to mimic the training samples of its victims. The generated instances will be mislabelled deliberately, and be used to update the global model. In this way, in order to distinguish their training samples from generated ones, victims are forced to reveal much more information about their training samples than initially intended, which helps the adversary improve its GAN model continuously until convergence. Figure 3 demonstrates the GAN attack on collaborative deep learning.

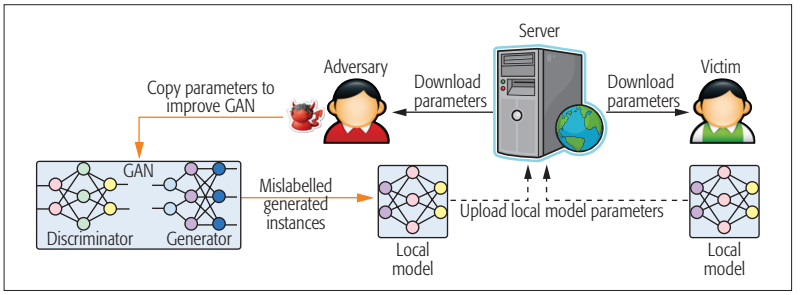

FIGURE 3. GAN attack on collaborative deep learning. Both the adversary and the victim upload the parameters of their local models to the sever, however, the adversary uses mislabelled instances generated from the generator, rather than real training samples with target label, to train its local model, such that the victim has to embed more information about its private training samples (with the target label) into the uploaded parameters. Consequently, the adversary downloads the parameters to improve its secret GAN.

Remarks: First, the adversary neither needs to compromise the central parameter server, nor collude with it. Second, from both the perspectives of honest participants and the central parameter server, the adversary acts properly, including uploading and downloading necessary information as agreed in advance. Third, record-level Differential Privacy (DP) may be ineffective, since a low privacy budget makes the local model unable to learn, and consequently, collaborative learning fails completely, while a high privacy budget might not prevent the adversary from achieving a good GAN model. Setting differential privacy granularity properly is difficult.

Countermeasures

We now discuss privacy-preserving NN training schemes in both centralized and distributed approaches (Table 1).

Privacy-Preserving Centralized Training:

Secure Two-Party Computation-Based: P. Mohassel and Y. Zhang [5] extended the single server training method to a two-server model, where training samples are distributed among two untrusted servers. The two servers train a target NN model on the union of training samples from different contributors using Secure Two-Party Computation (STPC), such that no information is exposed except the trained model. In their solution, all training samples are securely separated into two parts, and each server holds one of them. In addition, intermediate results, such as coefficients (weights) of neurons, are also secretly shared between servers. During forward and

Reference	Design goal	Key technique	Training methodology	Attack model
[3]	Protecting training data	Sharing local model parameters rather than training samples.	Collaborative	Semi-honest
[4]	Protecting privacy information in trained models	Conducting SGD within the framework of differential privacy.	Collaborative	Black/white-box
[5]	Protecting training data	Distributing private data among two servers to train models using secure two-party computation.	Centralized (two servers)	Semi-honest
[6]	Protecting training data	Integrating one-time pads based double-masking and threshold secret sharing to achieve secure aggregation.	Centralized	Semi-honest
[7]	Protecting prediction requests and trained models	Applying two-party secure scalar product to protect prediction requests and weights of a neural network.	–	Semi-honest
[8]	Protecting training data	Encrypting training data of participants using multi-key fully HE, and submitting the ciphertext to a server, decrypting the resulted model using SMC.	Collaborative	Semi-honest
[9]	Protecting training data	Deploying both learning code and training samples from different owners into a trusted processor-protected memory region.	Centralized	Semi-honest
[10]	Protecting arbitrarily partitioned training data	A public-key doubly homomorphic encryption scheme, supporting one multiplication and unlimited number of addition operations.	Collaborative	Semi-honest
[11]	Protecting training data	Parameters of local models are encrypted using an additively HE scheme before being shared (an improvement of [5]).	Collaborative	Semi-honest
[12]	Protecting training data	Local gradients (masked with local DP) are exchanged using HE and TSS. Global gradient is automatically decryptable once enough local gradients are aggregated.	Collaborative	Semi-honest
[13]	Protecting prediction requests and trained models	Having two parties additively share each of the input and output values for every layer of an NN, common NNs are transformed into oblivious NNs (by using STPC and HE).	–	Semi-honest

TABLE 1. Privacy-preserving solutions for NN training and prediction.

backward propagations for model development, secure two-party addition and multiplication can be achieved taking advantage of additive sharing and Beaver's triplet technique, while activation functions and partial derivatives are calculated based on Yao sharing. We emphasize that the confidentiality of training samples relies on the assumption that two servers will never collude with each other.

Trusted Processor-Based: O. Ohrimenko *et al.* [9] introduced a solution, which allows an NN to be trained on a combination of outsourced data from multiple parities, by utilizing a trusted processor on the cloud side. In this design, a collection of publicly-known learning codes is deployed into a processor-protected memory region, i.e., an *enclave*. Training samples from different parties are encrypted using symmetric keys and uploaded into the enclave together with the keys. The trustworthiness of the scheme relies on the following properties:

- Only those codes running in the enclave can access the training samples, which is monitored by the trustworthy processor.
- Training samples are always maintained in ciphertext except when they are loaded into caches of the processor.
- The training algorithm is data-oblivious, which implies that an attacker interacting with the algorithm and observing its interaction with the memory and disk learns nothing secret.

This benefits from the dense processing on training samples, which makes the memory access patterns independent of any particular data instance, preventing the attacker from inducing secrets by using side channel information.

Privacy-Preserving Collaborative Training:

The State-of-the-Art: For the first time, R. Shokri and V. Shmatikov [3] presented the concept of collaborative deep learning as a way to protect the privacy of training datasets. The authors considered a passive adversary model where both central parameter server and participants are honest-but-curious. In this model, participants learn their local NN models by utilizing the data of their own, and selectively share subsets of parameters (the gradients computed) of their local models, to optimize a global NN model. In this way, participants can concurrently train their models, and can benefit from others without sharing datasets. The effectiveness of collaborative training is attributed to the fact that the SGD algorithm underlying NN training can be parallelized and executed asynchronously. Based on this, after each round of local training, participants share the gradients they got to a central server separately. The sum of all gradients for a certain parameter is used to calculate the magnitude of the global descent toward the optima of the parameter. Participants then download the up-to-date parameters from the sever and use them to refine their local models.

However, it is demonstrated that even a small portion of the gradients can be exploited to recover local data. The following *enhanced schemes* are proposed to ensure that shared parameters do not leak much information about any single record in training datasets.

Conducting Differentially Private Randomization on Local Gradients: This is known as the local DP solution. Differentially private mechanisms add just the right amount of randomness (noise)

into datasets, for example, Laplace or Gaussian noises, such that real data can be concealed. DP provides a strong and rigorous definition of data privacy, requiring that two adjacent datasets, which differ in only one record, are statistically indistinguishable. This guarantee is particularly effective for making participants feel comfortable in contributing local gradients. However, there is a trade-off between the privacy-preserving training efficiency and privacy cost. Excessive noise injected might significantly hamper a learning procedure, making the model estimated low utility.

Using Additively Homomorphic Encryption (He) to Protect Gradients: L. T. Phong *et al*. [11] applied cryptography on local gradients to ensure confidentiality without reducing model accuracy. In the solution, participants are considered honest, and share a pair of public and secret keys generated according to an additively HE scheme. The public key is used to encrypt local gradients obtained after feeding a set of samples to the model. Due to the additively homomorphic property of the encryption algorithm, addition can be conducted over the ciphertexts by an untrusted central server, achieving correct model updating. Participants download corresponding parameters (in ciphertext), and decrypt them using the secret key.

Combining DP with HE: X. Zhang *et al.* [12] pointed out that in each mini-batch, global gradient descent can be computed by aggregating local gradients from participants. The authors observed that if DP mechanisms apply random noise sampled from symmetric distributions, as more participants aggregate their local gradients, a majority of the injected noise can be cancelled out, meaning that the aggregated gradients can approximate global gradients well. Thus, a gradient exchange protocol is designed based on HE and Threshold Secret Sharing (TSS). As long as enough participants have contributed their local gradients, the global gradient is automatically decryptable (which is defined as ρ-*visibility*), thereby available for all participants to access.

Using Secure Multi-Party (SMP) Aggregation to Calculate Mini-Batch Loss Gradients: K. Bonawitz *et al.* [6] considered local gradients from one participant in each update as a private vector, and designed a protocol to compute the sum of gradient vectors from different participants in a secure fashion. In the solution, each pair of participants agrees on a common seed of a pseudorandom generator, used for generating a mask vector between them, such that one adds the mask to its private vector, while another subtracts it from its vector. Then the mask will be canceled in the sum of the two vectors. In order to handle dropped-out users, a TSS-based recovery scheme is introduced, so that as long as the number of survivors exceeds a certain threshold, the dropped seeds can be recovered and removed from the sum.

Homomorphic Encryption-Based: J. Yuan and S. Yu [10] provided a mechanism that allows multiple parties to conduct joint back-propagation NN learning on arbitrarily partitioned datasets, with the help of a cloud server. In their scheme, a Trusted Authority (TA) is introduced for the purpose of generating and issuing a pair of (public and secret) system keys. The scheme relies on an encryption algorithm with homomorphic property, carried out using the public key known to all parties, and a secret sharing supported decryption algorithm, implying that one party can decrypt its own share of a secret using its portion of the secret key, which is randomly split by the TA. During training, each party encrypts its input data using the same public key and uploads the encrypted data to the cloud. Taking advantage of the homomorphic property, the cloud can carry out feed forward and back-propagation stages on those ciphertexts. The resulting intermediate values, e.g., the weights and values of hidden layer nodes, can be decrypted in parallel by all parties. Each party only gets a random share of certain intermediate value. Moreover, the parties can conduct next-step operations collaboratively without knowing the actual intermediate values. On the other hand, the training dataset, weight matrix of the model, and intermediate values of neurons are distributed across different parties. In this way, both the private data of each party and all intermediate results will not be exposed to anybody but the TA (since it holds the master key of the system). In practice, government agents may serve as the TA.

The sum of all gradients for a certain parameter is used to calculate the magnitude of the global descent toward the optima of the parameter. Participants then download the up-to-date parameters from the sever and use them to refine their local models.

P. Li *et al.* [8] addressed a similar problem based on *multi-key fully homomorphic encryption*. In their solution, each party has its own pair of secret and public keys rather than sharing public key pairs. The parties exploit their own public key to encrypt their share of secrets, then update the ciphertexts to a cloud for secure deep learning. The parties jointly run an SMC protocol to decrypt the learning results, which requires the interaction among parties. An alternative solution is proposed, which includes a TA into the architecture, and combines a *double decryption mechanism* (besides the general secret keys, there exists a master secret key that can be used to decrypt any given ciphertext successfully) with HE. After receiving ciphertexts under different general secret keys, the cloud will blind them with random values, then send the blinded ciphertexts to the TA. The TA transforms them into ciphertexts under single-key using the master secret key, and sends them back to the cloud to conduct training. The training results will also be transformed by the TA such that each party can decrypt them independently.

Privacy Issues During Prediction

During the prediction phase, the main privacy concerns come from two aspects. First, prediction requests may contain sensitive information, thus submitting prediction requests to a cloud-based MLaaS puts the privacy of the users at risk. Thus, it is necessary to design privacy-preserving protocols for NN prediction. Second, the knowledge embedded in trained models should be well protected, for a two-fold reason. On the one side, the training data of a trained model are privacy-sensitive. However, the trained model learns essential relationships between the input data and corresponding output labels, so it is possible

Reference	Attack	Attack model	Methodology	Goal
[1]	Model inversion	White-box	Maximize confidence values	Extracting images of training subjects from facial recognition models.
[14]	Membership inference	Black-box	Shadow training	Determining if a record is in a model's training dataset.
[15]	Model extraction	Black-box	Generic equation-solving	Learning a close approximation of the targeted model using as few queries as possible.

TABLE 2. Attacks on trained NN models.

The training dataset, weight matrix of the model, and intermediate values of neurons are distributed across different parties. In this way, both the private data of each party and all intermediate results will not be exposed to anybody but the TA (since it holds the master key of the system).

for a sophisticated attacker to recover the inputs according to observed outputs. On the other side, internal information about the model architecture, such as weights, topology, and activation functions used, contains intellectual properties that should be kept private. An attacker may attempt to uncover that information by analyzing the relationship between carefully designed input dataset and corresponding output labels obtained from the prediction results of a victim model.

Privacy-Preserving Protocols for NN Prediction

Privacy-preserving protocols for NN prediction aim to solve the problem that *Alice* holds a set of prediction requests (personal data) wishing to be processed by an NN held by *Bob*, without leaking data to him. Meanwhile, knowledge about the NN should be kept in secret from *Alice*. Existing solutions are based on STPC or HE. In addition, honest-but-curious attackers are considered (Table 1).

An early work was presented by M. Barni *et al*. [7]. In the solution, *Bob* conducts linear operations on *Alice*'s encrypted data and sends the results back to her, who decrypts, applies the non-linear activation functions on the plaintexts, and re-encrypts the results before sending them to *Bob* for next-layer processing. For a stronger privacy requirement that the activation functions should not be known by *Alice*, activation functions are approximated by certain polynomials, which can be divided into a set of linear operations, thus can be calculated securely. Topology protection is achieved by randomly embedding fake neurons into the network. A recent approach proposed by J. Liu *et al*. [13] introduced a technique to transform any common NN model into an *oblivious* NN without any modifications to the training phase. The core idea is to have Alice and Bob additively share each of the input and output values for every layer of an NN. That is, at the beginning of every layer, *Alice* and *Bob* will each hold a "share" such that modulo addition of the share is equal to the input to that layer in the non-oblivious version of that NN. The output values will be used as inputs for the next layer.

Attacks on Trained Models

Attack Models: During the prediction phase, black-box and white-box attackers are considered.

Black-Box Attacker: This is an attacker who only has the ability to make prediction queries against a model, but cannot actually obtain the model description. This means that internal information about the model, such as the architecture, optimization procedure, or training data, is not available to the attacker. Yet the queries can be adaptive, which implies that the attacker chooses a new query according to previous prediction outputs.

White-Box Attacker: This is an attacker who is able to obtain a description of a model, even partial training samples, besides querying a model to get predictions. Some MLaaS systems, such as BigML and Microsoft Azure Learning, allow their customers to specify whether APIs should provide white-box access to their models or not.

Actually, the line between white-box and black-box attackers is blurry. Black-box attackers may launch model extraction attacks, and ultimately turn a black-box model into white-box model.

Attacks on Trained NN Models: As far as we know, attempts to attack an NN model fall into three categories (Table 2):

Model Inversion Attacks: Model inversion attacks refer to the feasibility to back out sensitive data or features from a released model (by making prediction queries against the model). For example, M. Fredrikson *et al.* [1] launched white-box attacks against NN-based facial recognition APIs, leveraging the confidence values that are commonly revealed along with predictions. The authors demonstrated that it is possible for an attacker to reconstruct an image of a person only given his or her name (refered to as a class label of the output of the APIs). They formalized the model inversion attack into an optimization problem, that is, finding an image as input to maximize the resulting confidence value of the target class.

Membership Inference Attacks: Membership inference denotes that given a model and a record, an attacker determines whether the record is a part of the training dataset of the model or not. If such attacks can be launched successfully, it indicates that people who contribute training data may suffer privacy leakage. For example, a medical record of a patient used to train a disease model can reveal that the patient has such disease. R. Shokri *et al.* [14] observed that machine learning models tend to behave differently on their training data versus those they never "met" before, which might because of overfitting. Consequently, the authors developed a *Shadow Training* technique (through only black-box queries) to train an adversary model that is capable of distinguishing those behavior differences.

Model Extraction Attacks: A model extraction attack occurs when an attacker accesses certain target models in a black-box manner, and attempts to extract an equivalent or near-equivalent model (i.e., *avatar model*), achieving (nearly)

100 percent agreement with the target model on an input space of interest. Model extraction could, in turn, leak information about sensitive training data, thus it can be a stepping stone to those privacy-abusing attacks, e.g., model inversion. F. Trmèr *et al.* [15] demonstrated that, if confidence values are available, launching model extraction attacks against NNs is equal to solving non-linear equation systems in weights, which refers to gradient descent-based optimizations. If only labels are outputted, the target model can be extracted through *model retraining*, which finds samples along the decision boundary of the target model, and then trains an avatar model on them.

Countermeasures: Actually, there are few results in the field ficused on defending against attacks attempting to recover private information of trained models. One solution is to apply DP mechanisms in model training. M. Abadi *et al.* [4] combined a differentially private mechanism with the SGD algorithm, where noise is added onto the average gradients of mini-batches for the purpose of scrambling. Benefiting from the rigorous worst-case definition of privacy loss, the differentially private mechanism can defend against a strong white-box attacker, who has not only the full knowledge of the training mechanism and parameters of a model, but also controls all the training data except one record it is targeting. More importantly, by following the composability of DP, the authors provided a privacy-loss-tracking mechanism. A tight automated analysis of the overall privacy cost to the training data can be conducted, avoiding adding excessive noise to gradients, which would destroy the utility of the learned model.

Conclusions

In this article, we concentrated on privacy issues in NN learning. Privacy threats have been well analyzed, and recent research advances in this field have been surveyed. During the NN training phase, we described attackers who are passive and active insiders, and the attacking methodologies of active attackers. Countermeasures for both centralized and collaborative training were introduced. During the prediction phase, we analyzed white-box and black-box attackers, and outlined attacking methodologies including model inversion, membership inference and model extraction, and discussed countermeasures.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant No. 61672151, 61370205, 61772340, 61472255, 61420106010); the Fundamental Research Funds for the Central Universities (Grant No. EG2018028); the Shanghai Rising-Star Program (Grant No.17QA1400100); and the DHU Distinguished Young Professor Program.

Countermeasures for both centralized and collaborative training were introduced. During the prediction phase, we analyzed white-box and black-box attackers, and outlined attacking methodologies including model inversion, membership inference and model extraction, and discussed countermeasures.

References

[1] M. Fredrikson *et al.*, "Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures," *Proc. ACM CCS*, 2015, pp. 1322–33.
[2] B. Hitaj *et al.*, "Deep Models under the GAN: Information Leakage from Collaborative Deep Learning," *Proc. ACM CCS*, 2017.
[3] R. Shokri *et al.*, "Privacy-Preserving Deep Learning," *Proc. ACM SIGSAC*, 2015, pp. 1310–21.
[4] M. Abadi *et al.*, "Deep Learning with Differential Privacy," *Proc. ACM CCS*, 2016, pp. 308–18.
[5] P. Mohassel *et al.*, "SecureML: A System for Scalable Privacy preserving Machine Learning," *IEEE Sym. SP*, 2017, pp. 19–38.
[6] K. Bonawitz *et al.*, "Practical Secure Aggregation for Privacy-Preserving Machine Learning," *Cryptology* ePrint Archive, 2017, pp. 1175–91.
[7] M. Barni *et al.*, "A Privacy-Preserving Protocol for Neural-Network- Based Computation," *Proc. ACM MM&Sec*, 2006, pp. 146–51.
[8] P. Li *et al.*, "Multi-Key Privacy-Preserving Deep Learning in Cloud Computing," *J. Future Gener. Comput. Syst.*, 2017, pp. 76–85.
[9] O. Ohrimenko *et al.*, "Oblivious Multi-Party Machine Learning on Trusted Processors," *USENIX Sec. Sym.*, 2016, pp. 619–36.
[10] J. Yuan *et al.*, "Privacy Preserving Back-Propagation Neural Network Learning Made Practical with Cloud Computing," *IEEE Trans. PDS*, 2014, pp. 212–21.
[11] L. T. Phong *et al.*, "Privacy-Preserving Deep Learning: Revisited and Enhanced," ATIS, 2017, pp. 100–10.
[12] X. Zhang et al., "Private, Yet Practical, Multiparty Deep Learning," *ICDCS*, 2017, pp. 1442–52.
[13] J. Liu *et al.*, "Oblivious Neural Network Predictions via MiniONN Transformations," *Cryptology* ePrint Archive, 2017, pp. 619–31.
[14] R. Shokri *et al.*, "Membership Inference Attacks against Machine Learning Models," *IEEE Sym. SP*, 2017, pp. 3–18.
[15] F. Tramèr *et al.*, "Stealing Machine Learning Models via Prediction APIs," *USENIX Sec. Sym.*, 2016.

Biographies

Shan Chang [M] (changshan@dhu.edu.cn) is an associate professor with the School of Computer Science and Technology, Donghua University, Shanghai. She received her B.S. degree from the Department of Computer Science and Technology at Xi'an Jiaotong University in 2004, and a Ph.D. degree from the Department of Computer Software and Theory at Xi'an Jiaotong University in 2012. Her research interests include computer network security, mobile and wireless communication security, and big data security and privacy. She is a member of the ACM.

Chao Li (chaoli@mail.dhu.edu.cn) is currently a postgraduate student in the School of Computer Science and Technology, Donghua University, Shanghai. He received a B.S. degree in computer science and technology from Anhui University. His research interests include security and privacy in distributed systems.