Learning DNN Model with Error – Exposing the Hidden Model of BAYHENN

Harry W. H. Wong, **Jack P. K. Ma**, **Donald P. H. Wong**,
Lucien K. L. Ng and **Sherman S. M. Chow***
Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong
{wwh016, mpk016, wph019, nkl018, sherman}@ie.cuhk.edu.hk

Abstract

Privacy-preserving deep neural network (DNN) inference remains an intriguing problem even after the rapid developments of different communities. One challenge is that cryptographic techniques such as homomorphic encryption (HE) do not natively support non-linear computations (e.g., sigmoid). A recent work, BAYHENN (Xie et al., IJCAI'19), considers HE over the Bayesian neural network (BNN). The novelty lies in "meta-prediction" over a few noisy DNNs. The claim was that the clients can get intermediate outputs (to apply non-linear function) but are still prevented from learning the exact model parameters, which was justified via the widely-used learning-with-error (LWE) assumption (with Gaussian noises as the error). This paper refutes the security claim of BAYHENN via both theoretical and empirical analyses. We formally define a security game with different oracle queries capturing two realistic threat models. Our attack assuming a semi-honest adversary reveals all the parameters of single-layer BAYHENN, which generalizes to recovering the whole model that is "as good as" the BNN approximation of the original DNN, either under the malicious adversary model or with an increased number of oracle queries. This shows the need for rigorous security analysis ("the noise introduced by BNN can obfuscate the model" fails – it is beyond what LWE guarantees) and calls for the collaboration between cryptographers and machine-learning experts to devise practical yet provably-secure solutions.

1 Introduction

Some machine-learning as-a-service (MLaaS) platforms allow clients to obtain inference results given by a deep neural network (DNN) on their queries. The query often involves sensitive data that the MLaaS provider should not know, especially in healthcare/financial applications. Likewise, leaking the valuable well-trained model, which could further allow the recovery of the sensitive training data, can be catastrophic.

*Supported by General Research Fund (CUHK 14210319)

An ideal (and seemingly counterintuitive) privacy goal is that the MLaaS provider can carry out inference without knowing the query and without leaking the model to the querying client. This is also known as secure two-party computation. Using garbled circuits (GC), everything computable (by a Boolean circuit) can be done "securely." Nevertheless, applying GC over the whole inference computation is naturally less efficient than tailor-made designs optimized for some specific computations. Moreover, cryptographic computations mostly work on a finite field of integers, while floating-point computations are extensively used in machine-learning computations. Constructing a secure (*i.e.*, privacy-preserving) solution for computation of machine-learning tasks is thus highly non-trivial, which requires the expertise of multiple communities, including *scientific computing*, *machine learning*, and *cryptography* in particular. It becomes even more complicated if the design relies on the trusted computation enabled by *trusted processors* (*e.g.*, limited memory space), the speedy computation enabled by *graphic processing units* (*e.g.*, incompatibility with cryptographic operations), or the involvement of *external servers* (which are not fully-trusted) to perform the computation collaboratively. One may consult [Chow, 2019] for a brief overview.

Homomorphic encryption (HE) is a fundamental building block that is helpful for secure computation of arithmetic circuits, which allows performing linear operations over the encrypted data, *e.g.*, a feature vector. Fully homomorphic encryption (FHE) allows an arbitrary number of additions and multiplication, while somewhat homomorphic encryption allows a limited number of them. The simplest HE is additive HE (AHE) that only allows addition but cannot support the multiplication of two ciphertexts. Naturally, the efficiency of the less versatile HE schemes is higher than the more powerful ones in general. FHE has been applied in private inference for simpler models such as support vector machines, naïve Bayes classifiers, and decision trees [Bost *et al.*, 2015]. Subsequently, a tailored design for decision trees by using just AHE is proposed [Tai *et al.*, 2017]. For neural networks (NN), CryptoNet [Gilad-Bachrach *et al.*, 2016] utilized FHE.

As HE cannot natively support non-linear operations, one of the major obstacles is to also support non-linear operations efficiently, *e.g.*, argmax, comparison, or many popular activation functions used in NN. Apart from linear approximation [Gilad-Bachrach *et al.*, 2016; Wagh *et al.*, 2019], many

different flavors of techniques have been proposed, such as reducing the model size [Bourse *et al.*, 2018] or operating over transformed ciphertext [Juvekar *et al.*, 2018]. However, the involved cryptographic techniques still slow down the inference in the plaintext domain by several orders of magnitude.

Realizing that data scientists might not be that familiar with HE, nGraph-HE2 [Boemer *et al.*, 2019] is proposed as a compiler that produces HE-based programs for popular DNN frameworks, *e.g.* TensorFlow and PyTorch, for oblivious inference. With various (encoding and arithmetic) optimization for HE, their experiment results show an impressive improvement over CryptoNet. Specifically, for inference over MobileNetV2, a lightweight network with a relatively small number of parameter, they achieve $60.4\%/82.7\%$ top-1/top-5 accuracy and an amortized runtime of 381 ms/image on the ImageNet dataset. Yet, these figures are obtained under the "client-aided model," in which the client decrypts the intermediate result of each linear layer for non-linear activations.

GELU-net [Zhang *et al.*, 2018] proposed a secure DNN *training* protocol that still relies on HE for outsourcing the weighted sum operation to a centralized server (for each neuron layer during the forward propagation). For non-linear activation over the resulting encrypted dot-product, instead of still relying on the server to perform some complicated cryptographic operation, it is returned to a data provider for decryption and performing activation on the *plaintext*. A take-home message confirmed by GELU-net via experiment is that such plaintext computation, even with the frequent network communication, is better than the FHE approach of using fifth-order Taylor expansion for approximating the activation function via FHE. Note that backpropagation also relies on HE. For the server to update the model, the data provider applies random masks to the gradients before returning them.

Seeing the potential of using extremely-efficient plaintext computation for "supplementing" HE-based linear operations, BAYHENN [Xie *et al.*, 2019] proposed a secure Bayesian neural network (BNN) inference protocol by allowing the client to learn intermediate results in clear, but "without" using *the original* private DNN model. Presumably, their weights in "BNN" are sampled from a trained DNN model with Gaussian noises added, and taking the average over the final results. Their security argument resorts to the learning-with-error assumption (LWE), which is a cornerstone of lattice-based cryptography and widely used. Observing the similarity between LWE and DNN with Gaussian noise, they claim the protocol is secure against any semi-honest adversary for an unlimited number of queries.

1.1 Our Contribution

Cryptography is famous for its provably-secure guarantee against generic attack strategies without imposing too many ad-hoc assumptions (*e.g.*, the plaintext data is full of entropy). To avoid "too much" plaintext leakage, GELU-net suggested a heuristic defense of imposing a bound on the number of times a client's data can be used continuously for training, which is randomly picked from $[1, m + 2]$, where m is the number of neurons in a layer. Learning the rationale behind the rate-limiting of GELU-net, lifting the restriction in BAYHENN without a formal analysis is walking a tightrope.

This paper refutes the security claim of BAYHENN via both theoretical and empirical analyses. We formally define a security game with oracle queries modeling a realistic threat in practice. Concretely, given merely meta information (the input and output sizes), our attack can completely reveal the parameters in a single layer (from the leakage of the intermediate outputs), which in turn recover the whole model ultimately (under a slightly stronger adversary model or with more oracle queries). The model learned from our attack is "as good as" how good the BNN approximated the original DNN in the first place. This showcases the risk of adding noise "casually" instead of using well-studied cryptographic tools – the noise introduced by BNN (originally for another purpose) cannot provide the intended protection.

To measure the effectiveness of our attack, we analyze the extraction accuracy, which is the closeness of the results between the extracted and the original models for the same test dataset with respect to the number of queries and parameters. We conduct experiments to extract the BNN model on different real-world datasets to validate our claim. We argue that the LWE assumption can never protect the model for scenarios implicitly envisioned by BAYHENN.

1.2 Related Work

In model inversion attack, some specific background information and their connection with a specific training data item are often needed. For example, for face recognition tasks, a unique identifier or a blurred image is needed [Fredrikson *et al.*, 2015]. In contrast, our attack aims to "denoise" the obfuscated model without the need for such auxiliary information.

Cryptography is for reducing the trust of the well-behavior of the computing party. Some researchers thus shoot for using lesser cryptographic techniques by relying on extra trust assumptions or the involvement of more servers instead of more server-client interactions. Examples include relying on a trusted processor [Tramèr and Boneh, 2019] or the assumption that two or more individually-untrustworthy servers collaboratively perform the inference yet without colluding with each other. To overcome HE inefficiency on non-linear functions, SecureNN [Wagh *et al.*, 2019] utilizes additive-secret sharing[1] that also allows linear operation, but multiplication requires online interaction for the share-holding parties (that is why they cannot collude). Computations are faster than prior work but at the cost of increased communication rounds linear in the maximum multiplicative depth of the function.

With the current state-of-the-affairs, it remains a challenging open problem to come up with a secure design that can speed up privacy-preserving inference.

Outline. Section 2 presents some background of cryptographic primitives and BAYENN. We present and analyze our attack theoretically in Section 3 and empirically in Section 4.

2 Preliminary

Throughout the paper, boldface capital letters (*e.g.*, $\mathbf{A}$) and small letters (*e.g.*, $\mathbf{a}$) respectively denote matrix and vector of scalars, which are denoted by regular small letters (*e.g.*, a).

[1]Secret sharing of x is $\{x_k\}$. x can be recovered given a sufficient number of x_k. Insufficient shares leak no information about x.

Algorithm 1: BNN or DNN ($S = 1$, $\mathbf{\Sigma} = \emptyset$) Inference

Input: $\mathbf{a}$, $\{\mathbf{W}^i, \mathbf{b}^i, \mathbf{\Sigma}^i, \varphi^i\}_{i\in[1,\ell]}, S$
Output: The mean of the output $\mathbf{z}$
1: $\{\mathbf{a}_k^0 \leftarrow \mathbf{a}\}_{k\in[1,S]}$ {initialize $\mathbf{a}_k^0$ with the same $\mathbf{a}$}
2: **for** $i \in [1, \ell]$ **do**
3: $\{\mathbf{W}_k^i, \mathbf{b}_k^i\}_{k\in[1,S]} \leftarrow \mathcal{N}((\mathbf{W}^i, \mathbf{b}^i), \mathbf{\Sigma}^i)$
4: $\{\mathbf{z}_k^i \leftarrow \mathbf{a}_k^{i-1}\mathbf{W}_k^i + \mathbf{b}_k^i\}_{k\in[1,S]}$
5: $\{\mathbf{a}_k^i \leftarrow \varphi^i(\mathbf{z}_k^i)\}_{k\in[1,S]}$
6: **end for**
7: **return** $\mathbf{z} \leftarrow \frac{1}{S}\sum_{k=1}^{S} \mathbf{a}_k^\ell$

2.1 Cryptographic Primitives

Learning with Errors (LWE). Let $\mathbb{Z}_q$ denote the ring of integers modulo a positive integer q and let $\mathbb{Z}_q^n$ denote the set of n-vectors over $\mathbb{Z}_q$. Given a probability distribution χ over $\mathbb{Z}$, a positive integer n, and a positive integer q of size dependent on n, the LWE distribution $\mathcal{L}_{\chi,q}$ is obtained by: for any $\mathbf{s} \in \mathbb{Z}_q^n$, sample $\mathbf{a}$ *uniformly* from $\mathbb{Z}_q^n$ and the error $e \in \mathbb{Z}_q$ according to χ, and output $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. The (search) LWE problem is to recover $\mathbf{s}$ from any LWE instance randomly sampled from $\mathcal{L}_{\chi,q}$. This is conjectured to be computationally hard. Such an assumption is widely used in cryptography, in particular, for building SHE/FHE.

Homomorphic Encryption (HE). In a public-key encryption (PKE) scheme, a client generates a key pair $(\mathsf{pk}, \mathsf{sk})$ by invoking $\mathsf{KGen}(1^\lambda)$, where λ is the security parameter. The secret key sk is kept private while pk will be published. Enc encrypts a message m under a public key pk, which outputs $\tilde{m} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$. We use the $\tilde{m}$ notation to denote an encryption of m. Dec takes sk and $\tilde{m}$, and outputs $m \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\tilde{m})$. HE is PKE that features homomorphic operations ($\oplus$, $\otimes$) where $\tilde{x} \oplus \tilde{y} = \mathsf{Enc}_{\mathsf{pk}}(x + y)$ and $(c \text{ or } \tilde{c}) \otimes \tilde{y} = \mathsf{Enc}_{\mathsf{pk}}(c \times y)$. Specifically, (linear) HE supports only addition and plaintext-ciphertext multiplication ($c \otimes \tilde{y}$, but not $\tilde{c} \otimes \tilde{y}$). An HE scheme that supports some/unlimited ciphertext-ciphertext multiplications is called somewhat/fully homomorphic encryption (S/FHE). The standard security requirement of HE is indistinguishability against chosen-plaintext attack, or semantic security. Namely, it guarantees that any probabilistic polynomial-time adversary cannot distinguish between the ciphertexts encrypting two adversarially-chosen messages.

2.2 Neural Network

Deep Neural Network (DNN). A DNN consists of weight matrix $\mathbf{W}^i$, bias $\mathbf{b}^i$, and activation function φ^i for each layer $i \in [1, \ell]$. We call $\{\mathbf{W}^i, \mathbf{b}^i\}_{i\in[1,\ell]}$ the model parameter. For a DNN inference task over an input feature vector $\mathbf{a}$ ($:= \mathbf{a}^0$), the i-th layer ($i > 0$) takes the output from the $(i-1)$-th layer $\mathbf{a}^{i-1}$ and computes $\mathbf{z}^i = \mathbf{a}^{i-1}\mathbf{W}^i + \mathbf{b}^i$, then it outputs the activated value of $\mathbf{z}^i$ as $\mathbf{a}^i = \varphi^i(\mathbf{z}^i)$. The final output is a class label t or a score for each class.

Secure Computation of Non-linear Function. To protect $\{\mathbf{W}^i, \mathbf{b}^i\}_{i\in[1,\ell]}$, we should also protect the i-th (linear) intermediate vector $\mathbf{z}^i = \mathbf{a}^{i-1}\mathbf{W}^i + \mathbf{b}^i$ before it is activated by the non-linear $\varphi^i(\cdot)$. Most works compute the non-linear φ^i via (polynomial or piecewise linear) approximation, which requires only addition and multiplication for HE-based approach, or relies on OR and AND gates for the garbled-circuit approach with addition processing. Alternatively, secret-sharing approaches [Wagh *et al.*, 2019] supports efficient online computation by doing a lot of offline pre-computation. Also, they either rely on two non-colluding servers or letting the client play as one such "server."

Algorithm 2: BAYHENN

Client Input: Feature vector $\mathbf{a}$
Server Input: Model with Gaussian noise covariance $\{\mathbf{W}^i, \mathbf{b}^i, \mathbf{\Sigma}^i, \varphi^i\}_{i\in[1,\ell]}, S$
Output: The label with the highest mean score t
1: Client: Execute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^n)$ and send pk
2: Server: Send $S, \{\varphi^i\}_{i\in[1,\ell]}$
3: Client: $\{\mathbf{a}_k^0 \leftarrow \mathbf{a}\}_{k\in[1,S]}$
4: **for** $i \in [1, \ell]$ **do**
5: Client: Send $\{\tilde{\mathbf{a}}_k^{i-1} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{a}_k^{i-1})\}_{k\in[1,S]}$
6: Server: $\{\mathbf{W}_k^i, \mathbf{b}_k^i\}_{k\in[1,S]} \leftarrow \mathcal{N}((\mathbf{W}^i, \mathbf{b}^i), \mathbf{\Sigma}^i)$
7: Server: $\{\tilde{\mathbf{b}}_k^i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{b}_k^i)\}_{k\in[1,S]}$
8: Server: Send $\{\tilde{\mathbf{z}}_k^i \leftarrow \tilde{\mathbf{a}}_k^{i-1} \otimes \mathbf{W}_k^i \oplus \tilde{\mathbf{b}}_k^i\}_{k\in[1,S]}$
9: Client: $\{\mathbf{z}_k^i \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\tilde{\mathbf{z}}_k^i)\}_{k\in[1,S]}$
10: Client: $\{\mathbf{a}_k^i \leftarrow \varphi^i(\mathbf{z}_k^i)\}_{k\in[1,S]}$
11: **end for**
12: $\mathbf{p} \leftarrow \frac{1}{S}\sum_{k=1}^{S} \mathbf{a}_k^\ell$
13: **return** $t \leftarrow \arg\max_j(p_j)$

Bayesian Neural Network (BNN). Unlike the DNN, which uses fixed weights, BNN represents the weights by a probability distribution over possible values. It trains an ensemble of networks with weights drawn from a private probability distribution and can be used to measure the uncertainty of weights. In BAYHENN, the posterior distribution of the weights given the dataset is estimated by Bayes by Backprop [Blundell *et al.*, 2015]. The BNN inference in BAYHENN is obtained by averaging evaluations of multiple DNN models, with the weights and biases sampled from the posterior distribution. More formally, an ℓ-layer BNN model is defined by $\{\mathcal{N}((\mathbf{W}^i, \mathbf{b}^i), \mathbf{\Sigma}^i)\}_{i\in[1,\ell]}$, *i.e.*, ℓ Gaussian distributions with means $(\mathbf{W}, \mathbf{b})$ and covariance $\mathbf{\Sigma}$, respectively. The server samples S instances of noisy DNN model $\{\{\mathbf{W}_k^i, \mathbf{b}_k^i\}_{i\in[1,\ell]}\}_{k\in[1,S]}$ (*cf.* Algorithm 1, DNN can be considered as a special instance with $S = 1$ and $\mathbf{\Sigma}^i = \mathbf{0}$ $\forall i \in [1, \ell]$). Inference invokes S DNN inference instances in parallel, and the user receives S output vectors (or label distribution) $\mathbf{a}_k^\ell$. The user averages the S output vectors (as the predicative distribution) and returns the index t that has the greatest value. We will focus on this kind of BNN in the rest of the paper since it is at the core of BAYHENN.

2.3 Bayesian (Homomorphically-Encrypted) NN

BAYHENN (Algorithm 2) separates the model evaluation into linear and non-linear (activation) parts, which are evaluated by the server and the client, respectively. For the k-th model in layer i, the client encrypts the input $\mathbf{a}_k^{i-1}$ under

Algorithm 3: Experiment $\mathsf{Exp}^{\mathcal{A}}_{\mathsf{BAYHENN\text{-}mode\text{-}type}}$

Input: $\{\mathbf{W}^i, \mathbf{b}^i, \mathbf{\Sigma}^i, \varphi^i\}_{i\in[1,\ell]}, S, \mathsf{mode}, \mathsf{type}$
Output: $0/1$
1: Embed $(\{\mathbf{W}^i, \mathbf{b}^i, \mathbf{\Sigma}^i, \varphi^i\}_{i\in[1,\ell]}, S)$ to $\mathcal{O}^{\mathsf{BAYHENN\text{-}mode}}$
2: $\{\mathbf{W}'^i, \mathbf{b}'^i\}_{i\in[1,\ell]} \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{BAYHENN\text{-}mode}}(\cdot)}(\{\varphi^i\}_{i\in[1,\ell]}, S)$
3: **return** $\mathsf{isClose}_{\mathsf{type}}((\mathbf{W}, \mathbf{b}), (\mathbf{W}', \mathbf{b}'))$

the client public key; and sends the resulting $\tilde{\mathbf{a}}_k^{i-1}$ to the server. The server then homomorphically evaluate[2] the sampled noisy model $(\mathbf{W}_k^i, \mathbf{b}_k^i)$ perturbed by Gaussian noises with covariance $\mathbf{\Sigma}^i$ over $\tilde{\mathbf{a}}_k^{i-1}$. Such linear (homomorphic) evaluation requires the bias vector $\mathbf{b}_k^i$ to be encrypted under the client public key as well. For the non-linear part, a client merely evaluates φ^i over the plaintext $\mathbf{z}_k^i$ obtained from decrypting the evaluated ciphertext returned by the server, and uses the evaluated value as the input for the $(i+1)$-th layer. This process is repeated until it reaches the final layer. Over S parallel-run evaluations, the client computes the mean of the outputs and returns the label with the highest mean score.

In BAYHENN, the privacy of the client data is inherited from the security of the HE scheme. The privacy of the BNN model is claimed to be protected via LWE, presumably from its structure analogous to the model evaluation, *e.g.*, $\mathbf{z} = \mathbf{aW} + \mathbf{b}$. The connection may look specious at first glance. It turns out there exists some explicit connection after some reformulation. (See Table 1). However, there is still a missing piece. We will explain why the LWE assumption is not applicable for the usage expected by BAYHENN, and propose a concrete attack to refute their security claim.

3 Proposed Attack

Our attack issues direct queries to the model (*e.g.*, via a machine-learning service API) on arbitrary input $\mathbf{a}$. The goal is to extract a model that is "close" to the victim model. We present our attack for single layer BAYHENN under semi-honest setting, and generalize it to whole-model extraction for multiple-layer BAYHENN under the malicious setting.

3.1 Formal Security Model for BAYHENN

Algorithm 3 defines our proposed security game (also known as an experiment), which formally captures what an attacker can do and what the goal of an attacker is, which in turn defines the security requirement. The challenger of the game holds a DNN model with a Gaussian distribution covariance. It embeds the DNN model to an oracle, for which the adversary can query by issuing a feature vector of its choice. The challenger first picks a Gaussian distribution for producing S instances of noisy DNN model. Upon the feature vector query, the challenge evaluates the noisy DNN models over it. The actual evaluation depends on the type of adversary to be explained below. Eventually, the adversary returns an estimated model that is evaluated by the isClose function to check whether the *error* is within a certain bound and output a bit $0/1$ to indicate loss or win. It either measures the uniform error or the test error. The uniform error is determined by ℓ_1-norm of difference between $(\mathbf{W}, \mathbf{W}')$ and $(\mathbf{b}, \mathbf{b}')$, while the test error is determined by the occurrence of outputting the same label for the same input derived from real experiments.

[2]More precisely, HE encrypts an SIMD encoding of $\mathbf{a}$ for efficiency, but it does not matter for the security analysis in this paper.

Algorithm 4: Oracle $\mathcal{O}^{\mathsf{BAYHENN\text{-}semi\text{-}honest}}$

Input: $\mathbf{a}$
Secret Internal Input: $\{\mathbf{W}^i, \mathbf{b}^i, \mathbf{\Sigma}^i, \varphi^i\}_{i\in[1,\ell]}, S$
Output: $\{\{\mathbf{a}_k^i, \mathbf{z}_k^i\}_{k\in[1,S]}\}_{i\in[1,\ell]}$
1: $\{\mathbf{a}_k^0 \leftarrow \mathbf{a}\}_{k\in[1,S]}$
2: **for** $i \in [1, \ell]$ **do**
3: $\quad\{\mathbf{W}_k^i, \mathbf{b}_k^i\}_{k\in[1,S]} \leftarrow \mathcal{N}((\mathbf{W}^i, \mathbf{b}^i), \mathbf{\Sigma}^i)$
4: $\quad\{\mathbf{z}_k^i \leftarrow \mathbf{a}_k^{i-1} \times \mathbf{W}_k^i + \mathbf{b}_k^i\}_{k\in[1,S]}$
5: $\quad\{\mathbf{a}_k^i \leftarrow \varphi^i(\mathbf{z}_k^i)\}_{k\in[1,S]}$
6: **end for**
7: **return** $\{\{\mathbf{a}_k^i, \mathbf{z}_k^i\}_{k\in[1,S]}\}_{i\in[1,\ell]}$

Algorithm 5: Oracle $\mathcal{O}^{\mathsf{BAYHENN\text{-}malicious}}$

Input: $\{\mathbf{a}_k\}_{k\in[1,S]}$
Secret Internal Input: $\{\mathbf{W}^i, \mathbf{b}^i, \mathbf{\Sigma}^i, \varphi^i\}_{i\in[1,\ell]}, S$
Initial State: $\mathsf{st} = 0$
Output: $\{\mathbf{z}_k\}_{k\in[1,S]}$
1: **if** $\mathsf{st} = 0$ **then**
2: $\quad\{\mathbf{a}_k \leftarrow \mathbf{a}_1\}_{k\in[1,S]}$
3: **end if**
4: $\{\mathbf{W}_k, \mathbf{b}_k\}_{k\in[1,S]} \leftarrow \mathcal{N}((\mathbf{W}^{\mathsf{st}}, \mathbf{b}^{\mathsf{st}}), \mathbf{\Sigma}^{\mathsf{st}})$
5: $\{\mathbf{z}_k \leftarrow \mathbf{a}_k \times \mathbf{W}_k + \mathbf{b}_k\}_{k\in[1,S]}$
6: $\mathsf{st} \leftarrow \mathsf{st} + 1 \bmod \ell$
7: **return** $\{\mathbf{z}_k\}_{k\in[1,S]}$

The adversarial power is determined by the attack mode, which can either be semi-honest or malicious. A semi-honest adversary does not deviate from the protocol specification, which is captured by the oracle in Algorithm 4. In particular, the oracle directly evaluates the activation function, as the adversary would faithfully do by itself regardless, and returns all layers' intermediate results in one shot upon a query $\mathbf{a}$. Since the adversary (as a querying client) holds the decryption key, the oracle returns all the intermediate values in plaintext.

A malicious adversary can arbitrarily choose its input to each layer and hence to a stateful oracle different from the semi-honest one. Algorithm 5 defines the oracle for malicious attack mode. It returns the intermediate results for the st-th layer, where st is the state storing the current layer number.

While BAYHENN assumes semi-honest attackers, we believe security in the malicious setting is relevant and worthy to consider as attackers have no incentive to follow the protocol in practice, especially when BAYHENN inherently allows freely chosen input for all layers. Without extra cryptographic machinery, such kind of arbitrary adversarial behavior is hidden from the server due to the indistinguishability of HE.

Algorithm 6: Procedure of $\mathcal{A}^{\mathcal{O}^{\mathsf{BAYHENN\text{-}semi\text{-}honest}}(\cdot)}$

Input: S
Output: $(\mathbf{W}', \mathbf{b}')$
$\{\{\mathbf{z}_{0,k}^i\}_{k\in[1,S]}\}_{i\in[1,\ell]} \leftarrow \mathcal{O}^{\mathsf{BAYHENN\text{-}semi\text{-}honest}}(\mathbf{0})$
$\mathbf{z}_0^1 \leftarrow \frac{1}{S}\sum_{k=1}^{S} \mathbf{z}_{0,k}^1$
for $j \in [1, n]$ **do**
  $\{\{\mathbf{z}_{j,k}^i\}_{k\in[1,S]}\}_{i\in[1,\ell]} \leftarrow \mathcal{O}^{\mathsf{BAYHENN\text{-}semi\text{-}honest}}(\mathbf{e}_j)$
  $\mathbf{z}_j^1 \leftarrow \frac{1}{S}(\sum_{k=1}^{S} \mathbf{z}_{j,k}^1) - \mathbf{z}_0^1$
end for
return $(\mathbf{W}', \mathbf{b}') \leftarrow (\{\mathbf{z}_j^1\}_{j\in[1,n]}, \mathbf{z}_0^1)$

Algorithm 7: Procedure of $\mathcal{A}^{\mathcal{O}^{\mathsf{BAYHENN\text{-}malicious}}(\cdot)}$

Input: S
Output: $(\mathbf{W}', \mathbf{b}')$
for $i \in [1, \ell]$ **do**
  $\{\mathbf{z}_{0,k}^i\}_{k\in[1,S]} \leftarrow \mathcal{O}^{\mathsf{BAYHENN\text{-}malicious}}(\{\mathbf{0}\}_{k\in[1,S]})$
  $\mathbf{z}_0^i \leftarrow \frac{1}{S}\sum_{k=1}^{S} \mathbf{z}_{0,k}^i$
end for
for $j \in [1, n]$ **do**
  for $i \in [1, \ell]$ **do**
    $\{\mathbf{z}_{j,k}^i\}_{k\in[1,S]} \leftarrow \mathcal{O}^{\mathsf{BAYHENN\text{-}malicious}}(\{\mathbf{e}_j\}_{k\in[1,S]})$
    $\mathbf{z}_j^i \leftarrow \frac{1}{S}(\sum_{k=1}^{S} \mathbf{z}_{j,k}^i) - \mathbf{z}_0^i$
  end for
end for
return $(\mathbf{W}', \mathbf{b}') \leftarrow (\{\{\mathbf{z}_j^i\}_{j\in[1,n]}\}_{i\in[1,\ell]}, \{\mathbf{z}_0^i\}_{i\in[1,\ell]})$

3.2 Model Extraction Attack on BAYHENN

Since the client can learn the intermediate result $(\mathbf{aW} + \mathbf{b})$ in plain, our attack works by picking a suitable input $\mathbf{a}$. The server provides intermediate results over S parallel evaluations. If an attacker can execute the inference $(n+1)$ times on the same DNN model but with different inputs, by writing the $(n+1)$ arbitrarily chosen queries or inputs to the next layer in a matrix form, and further assume such matrix is invertible, then the underlying DNN model can be extracted easily.

However, the above straightforward attack does not work against BAYHENN since it samples noisy models for each query, *i.e.*, the weight $\mathbf{W}$ and bias $\mathbf{b}$ changes. In other words, writing the queries in a matrix in the form of $\mathbf{Z} = \mathbf{A}[\mathbf{W}^T||\mathbf{b}^T]^T$, with input $\mathbf{a}_j$ and output $\mathbf{z}_j$ of the j-th query being the j-th row of matrix $\mathbf{A}$ and $\mathbf{Z}$ respectively, the above attack requires $(\mathbf{W}, \mathbf{b})$ to be the same for all $\mathbf{a}_j$. So, the model cannot be approximated ideally by taking the inverse.

Attack Strategy. We then craft a designated "small" query. By keeping the same input for each query and then taking the average, we can effectively reduce the noise and obtain a better sample mean by increasing the number of queries. Furthermore, as the noise is amplified by the query input (covariance of the error increases linearly with the average of ℓ_1-norm of the involved queries' input), a fixed query with a small norm can minimize its effect on the extracted model.

BAYHENN	LWE	Requirements for LWE's Security
$[\mathbf{a}\ 1]$	$\mathbf{a}$	Uniformly distributed
$\begin{bmatrix}\mathbf{w}'\\ b\end{bmatrix}$	$\mathbf{s}$	Secret to be protected
$e_{\mathbf{b}}$	e	Bounded

Table 1: Connecting BAYHENN parameters to the LWE assumption

Attack Procedure and its Analysis. The attack procedure for single-layer BAYHENN ($\ell = 1$) is given in Algorithm 6. The semi-honest adversary issues the same query for the inference task of all noisy models. (The first output of the oracle, *i.e.*, the activated values, is not useful here, and hence omitted.) It firstly queries zero vector $\mathbf{0}$ and computes the average over the inference results as $\mathbf{z}_0^1$, which is an estimate of $\mathbf{b}$ from $\mathbf{0W}' + \mathbf{b}'$. The error of this estimation is a zero-mean Gaussian distribution with covariance $\mathbf{\Sigma}/S$. Second, for $j \in [1, n]$, where n is the dimension of the input feature vector, it queries a standard basis $\mathbf{e}_j$ and takes an average over the inference results, which gives an estimation of $(\mathbf{w}_j + \mathbf{b})$ from $(\mathbf{e}_j\mathbf{W}' + \mathbf{b}')$, where $\mathbf{w}_j$ is the j-th row of weight matrix $\mathbf{W}$. For $\mathbf{w}_j$ itself, the attacker can cancel out the bias $\mathbf{b}$ by subtracting the estimated bias $\mathbf{b}'$, *i.e.*, $\mathbf{w}'_j = \mathbf{w}_j + \mathbf{b} - \mathbf{b}'$. The estimated $\mathbf{w}_j$ has a 0-mean Gaussian noise with covariance $2\mathbf{\Sigma}/S$. Given the covariance of the error, the exact bound can be easily obtained via Gaussian tail inequality.

All-Level Recovery under Semi-Honest Attacks. For the remaining layers, the attacker needs to use the estimated weights and biases of all previous layers (and the activation functions $\{\varphi^i\}_{i\in[1,\ell]}$) to compute a special query, which leads to a designated input on the next layer. However, the error becomes larger since the noise added to the model perturbed the inverse calculation; it thus needs significantly more queries.

Actively Malicious Attack. A malicious adversary can arbitrarily choose the input for each layer as in Algorithm 7 to independently estimate the weight and bias for each layer.

3.3 Misuse of LWE

The LWE assumption was misused [Xie *et al.*, 2019]. An LWE problem instance requires a uniformly random basis, but the analogous basis in the BAYHENN is the input chosen by the attacker. As an illustration, for an LWE instance, if one keeps sampling for the same basis and secret, and keeps averaging the samples, the bound of the Gaussian noise tends to be zero, which leads to an instance with zero noise ultimately. Our attack on BAYHENN exploits this fact.

Table 1 formulates the structure in BAYHENN (a noisy weight vector $\mathbf{w}'$, a single element bias b, and the Gaussian noise added to the bias e_b) and draws the correspondence to the LWE instance. As our attacks show, the problem of BAYHENN stems from the input $\mathbf{a}$. It is supposed to be the basis for the LWE instance, but it is not sampled uniformly at random. An attacker can thus query $\mathbf{a} = (1, \ldots, 0)$ to obtain $(s_0 + e)$. Our attack technically reduces to estimating the maximum likelihood of a Gaussian distribution, instead of attacking any underlying hard lattice problems.

3.4 The Case for nGraph-HE2

nGraph-HE2 [Boemer *et al.*, 2019] did not specify the treatment for the non-linear layers and left it as a placeholder for any secure protocol. In its experiments, it reveals the intermediate results in clear to clients, without the "obfuscation" of BAYHENN. Our attack thus applies, showing that such treatment is insecure. We remark that the nGraph-HE2 team is aware of the potential leakage about the model. Correspondingly, they suggest using additive secret sharing and GC for secure computation, or using a trusted processor to perform the decryption, and call for more secure implementations.

3.5 The Case for GELU-net for Learning

GELU-net [Zhang *et al.*, 2018] also allows the client to learn the intermediate result in clear to facilitate efficient evaluations of activation functions. Such an interactive decryption plus random masks approach is subject to a major constraint in preserving security – for a layer having m neurons, a data provider cannot participate in more than $(m + 2)$ iterations and needs to switch to another non-colluding data provider (if available). Unfortunately, the security definition provided for GELU-net is informal and does not capture collusion among data providers. Their simulation-based security proof does not capture the information from the view of the data provider in the backpropagation phase either, in which it can observe the gradient of the weights (for gradient descent). More seriously, the protocol does leak intermediate values that contain information about the server's model-in-training. Specifically, when the model is converging, *i.e.*, the gradient of weights is small, the intermediate results, including gradients of weights, can be used to approximate the model accurately.

Consequently, the pitfall is that rate-limiting is required to prevent complete leakage of the parties' input. The user can recover the weights in a layer by Gaussian elimination during forward propagation if it can obtain enough linear equations $(\mathbf{aw} + \mathbf{b})$ (which is performed in DNN). Thus the user can only provide m rounds of forward propagation. Even worse, the client knows the gradient during backpropagation. With the magnitude of the gradient, one can infer whether the model is near satisfactory and possibly extract the model using this knowledge (small gradient) within the rate limit.

4 Experiments

We simulate an attacker to recover a single (fully-connected) layer BAYHENN (with 91.45% testing accuracy on the MNIST dataset) by querying input vector $\{\mathbf{a}_i\}_{i\in[m]}$ to the victim model and obtain query results $\{\mathbf{z}_i\}_{i\in[m]}$ for a single round of attack. Given this experiment, one can apply the single-layer extraction attack under the semi-honest setting to the whole model extraction under the malicious setting.

Parameter Choices. The model parameter $\mathbf{W}$ and $\mathbf{b}$ of the victim model is of dimension 784×10 and 1×10, respectively. The entries in $\mathbf{a}_i$ are integers, while the entries in $\mathbf{W}$ and result matrix $\mathbf{Z}$ are floating-point numbers. As HE only works over $\mathbb{Z}_q$, we need to quantize all the values of $\mathbf{a}_i$, $\mathbf{W}$, and $\mathbf{b}$ (added by noises). We did that by first scaling up the values of each $\mathbf{a}_i$, $\mathbf{W}$, and $\mathbf{b}$ by 2^8 and then rounding them to $\mathbb{Z}_q$ for matching the HE computation. We chose $q = 2^{19} - 3$ as BAYHENN limits the bit-length to 20. To sample Gaussian noises used by in BAYHENN, we use 0 mean and different standard deviation σ (values provided are before scaled up) for different S. The test dataset consists of $10,000$ samples.

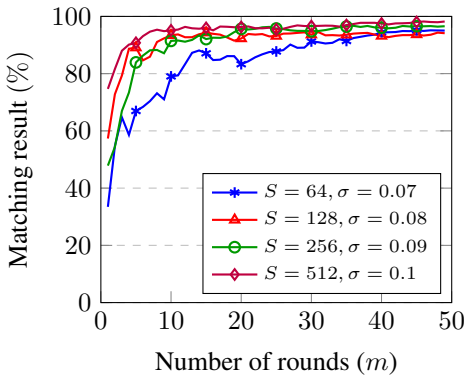

Figure 1: Closeness between the extracted and ground-truth models

Evaluation Goal. We repeat with different m to evaluate the number of queries needed to extract $(\mathbf{W}', \mathbf{b}')$ that performs as good as our ground-truth model $(\mathbf{W}, \mathbf{b})$. We evaluate how successful our attack is by computing $\mathsf{isClose}_{\mathsf{Test}}((\mathbf{W}, \mathbf{b}), (\mathbf{W}', \mathbf{b}'))$ that returns the occurrence of outputting the same label for the same test dataset. In BAYHENN, S is set by the model owner and known to the client. We compare the effect of different S in our attack.

Results. Figure 1 illustrates the effect of S and m. When S is large, we only need to conduct a small m number of rounds of attacks to obtain a model that is nearly equivalent to the ground truth. Our results show that the similarity between the testing results of our extracted model converges to the testing results of the victim model when m increases. If S is small, more rounds of attacks (m) are needed. When $S = 512$, $(\mathbf{W}', \mathbf{b}')$ attains 98.22% as the percentage of matching results relative to $(\mathbf{W}, \mathbf{b})$. We thus showed empirically that our attack can effectively extract a model as good as the one that was supposed to be protected by BAYHENN.

5 Conclusion and Open Problems

Cutting corners in designing cryptographic protocols, *e.g.*, using some "arbitrary" noises to replace the use of cryptography, is risky, especially without solid cryptographic proof. Unfortunately, in this paper, we refute the heuristic arguments given by a recent work for the privacy of model parameters, which comes from the resemblance of the noise introduced in the BNN model and that for the learning-with-error assumption. Several design defects break the security claim of BAYHENN – the introduced Gaussian noise can be easily eliminated, so the leakage of the recovered intermediate vector enables simple attacks from linear algebra, and the use of encryption prevents the server from detecting malicious attacks. We pose an open problem to investigate the possibilities of 1) complete model extraction under the semi-honest model or 2) extraction of more complex neural networks (*e.g.*, CNN with hidden hyper-parameters) under the BAYHENN framework.

References

[Blundell *et al.*, 2015] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *ICML*, pages 1613–1622, 2015.

[Boemer *et al.*, 2019] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE2: A high-throughput framework for neural network inference on encrypted data. In *Encrypted Computing & Applied Homomorphic Cryptography (WAHC), co-located with CCS*, pages 45–56, 2019.

[Bost *et al.*, 2015] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[Bourse *et al.*, 2018] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Crypto, Part III*, pages 483–512, 2018.

[Chow, 2019] Sherman S. M. Chow. Can we securely outsource big data analytics with lightweight cryptography? In *Security in Cloud Computing (SCC), co-located with AsiaCCS*, page 1, 2019.

[Fredrikson *et al.*, 2015] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *CCS*, pages 1322–1333, 2015.

[Gilad-Bachrach *et al.*, 2016] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*, pages 201–210, 2016.

[Juvekar *et al.*, 2018] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *Usenix Security*, pages 1651–1669, 2018.

[Tai *et al.*, 2017] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. In *ESORICS (Part II)*, pages 494–512, 2017.

[Tramèr and Boneh, 2019] Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *ICLR*, 2019.

[Wagh *et al.*, 2019] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *PoPETs*, (3):26–49, 2019.

[Xie *et al.*, 2019] Peichen Xie, Bingzhe Wu, and Guangyu Sun. BAYHENN: combining bayesian deep learning and homomorphic encryption for secure DNN inference. In *IJCAI*, pages 4831–4837, 2019.

[Zhang *et al.*, 2018] Qiao Zhang, Cong Wang, Hongyi Wu, Chunsheng Xin, and Tran V. Phuong. GELU-Net: A globally encrypted, locally unencrypted deep neural network for privacy-preserved learning. In *IJCAI-EJCAI*, pages 3933–3939, 2018.