Split Ways: Privacy-Preserving Training of Encrypted Data Using Split Learning

Tanveer Khan[1], Khoa Nguyen[1] and Antonis Michalas[1,2]

[1]Tampere University, Tampere, Finland
[2]RISE Research Institutes of Sweden

Abstract

Split Learning (SL) is a new collaborative learning technique that allows participants, e.g. a client and a server, to train machine learning models without the client sharing raw data. In this setting, the client initially applies its part of the machine learning model on the raw data to generate activation maps and then sends them to the server to continue the training process. Previous works in the field demonstrated that reconstructing activation maps could result in privacy leakage of client data. In addition to that, existing mitigation techniques that overcome the privacy leakage of SL prove to be significantly worse in terms of accuracy. In this paper, we improve upon previous works by constructing a protocol based on U-shaped SL that can operate on homomorphically encrypted data. More precisely, in our approach, the client applies Homomorphic Encryption (HE) on the activation maps before sending them to the server, thus protecting user privacy. This is an important improvement that reduces privacy leakage in comparison to other SL-based works. Finally, our results show that, with the optimum set of parameters, training with HE data in the U-shaped SL setting only reduces accuracy by 2.65% compared to training on plaintext. In addition, raw training data privacy is preserved.

Keywords

Homomorphic Encryption, Privacy-preserving Machine Learning, Split Learning

1. Introduction

Machine Learning (ML) models have attracted global adulation and are used in a plethora of applications such as medical diagnosis, pattern recognition, and credit risk assessment. However, applications and services using ML are often breaching user privacy. As a result, the need to preserve the confidentiality and privacy of individuals and maintain user trust has gained extra attention. This is not only because of the technological advancements that privacy-preserving machine learning (PPML) can offer, but also due to its potential societal impact (i.e. building fairer, democratic and unbiased societies) [1].

Split Learning (SL) and Federated Learning (FL) are the two methods of collaboratively training – a model derived from distributed data sources without sharing raw data [2]. In FL, every client runs a copy of the entire model on its data. The server receives updated weights from each client and aggregates them. The SL model divides the neural network into two parts: the client-side and the server-side [3]. SL is used for training Deep Neural Networks (DNN) among multiple data sources, while mitigating the need to directly share raw labeled data with collaboration parties. The advantages of SL are multifold: *(i)* it allows users to train ML models without sharing their raw data with a server running part of a DNN model. *(ii)* it protects both the client and the server from revealing their parts of the model, and *(iii)* it reduces the client's computational overhead by utilizing a smaller number of layers [4]. Though SL offers an extra layer of privacy protection by definition, there are no works exploring how it is combined with popular privacy-preserving techniques like Homomorphic Encryption (HE) [5]. In [6], the authors studied whether SL can handle sensitive time-series data and demonstrated that SL alone is *insufficient* when performing privacy-preserving training for 1-dimensional (1D) CNN models. More precisely, the authors showed raw data can be reconstructed from the activation maps of the intermediate split layer. The authors also employed two mitigation techniques, adding hidden layers and applying differential privacy to reduce privacy leakage. However, based on the results, none of these techniques can effectively reduce privacy leakage from all channels of the SL activation. Furthermore, both these techniques result in reducing the joint model's accuracy.

In this work, we construct a model that uses HE to mitigate privacy leakage in SL. In our model, the client first encrypts the activation maps and then sends the encrypted activation maps (EAMs) to the server. The EAMs do *not* reveal anything about the raw data (i.e. it is *not* possible to reconstruct raw data from the EAMs).

Vision AI systems have proven surpass people in recognizing abnormalities such as tumours on X-rays and

Published in the Workshop Proceedings of the EDBT/ICDT 2023 Joint Conference (March 28-March 31, 2023), Ioannina, Greece
tanveer.khan@tuni.fi (T. Khan); khoa.nguyen@tuni.fi (K. Nguyen); antonios.michalas@tuni.fi (A. Michalas)
https://www.amichalas.com/ (A. Michalas)
0000-0001-7296-2178 (T. Khan); 0000-0002-0189-3520 (A. Michalas)

ultrasound scans [7]. In addition to that, machines can reliably make diagnoses equal to those of human experts. All the evidence indicates that we can now build systems that achieve human expert performance in analyzing medical data – systems allowing humans to send their medical data to a remote AI service and receive an accurate automated diagnosis. An intelligent and efficient AI healthcare system of this type offers a great potential since it can improve the health of humans but also have an important social impact. However, these opportunities come with certain pitfalls, mainly concerning privacy. With this in mind, we have designed a system that analyzes images in a privacy-preserving way. More precisely, we show how encrypted images can be analyzed with high accuracy without leaking information about their actual content. While this is still far from our big dream (namely automated AI diagnosis) we still believe it is an important step that will eventually pave the way towards our timate goal.

Contributions The main contributions are:

- We designed a simplified version of the 1D CNN model presented in [6] and we are using it to classify the ECG signals [8] in both local and SL settings. More specifically, we construct a U-shaped split 1D CNN model and experiment using plaintext activation maps (PAMs) sent from the client to the server. Through the U-shaped 1D CNN model, clients do *not* need to share the input training samples and the ground truth labels with the server – this is an important improvement that reduces privacy leakage compared to [6].
- We constructed the HE version of the U-shaped SL. In the encrypted U-shaped SL, the client encrypts the activation map using HE and sends it to the server. The advantage of the HE encrypted U-shaped SL over the plaintext U-shaped SL is that the server performs computation over the EAMs.
- To assess the applicability of our framework, we performed experiments on a heartbeat datasets (MIT-DB [8]). We experimented with activation maps of 256 for both plaintext and homomorphically EAMs and we measured the model's performance by considering training duration, test accuracy, and communication cost.

2. Related Work

The SL approach proposed by Gupta and Raskar [9] offers a number of significant advantages over FL. Similar to FL [10], SL does *not* share raw data. In addition, it has the benefit of *not* disclosing the model's architecture and weights. For example, [9] predicted that reconstructing raw data on the client-side , while using SL would be difficult. In addition, the authors of [4]employed the SL model to the healthcare applications to protect the users' personal data. Vepakomma *et al.* found that SL outperforms FL in terms of accuracy [4].

Initially, it was believed that SL is a promising approach in terms of client raw data protection, however, SL provides data privacy on the grounds that only intermediate activation maps are shared between the parties. Different studies showed the possibility of privacy leakage in SL. In [2], the authors analyzed the privacy leakage of SL and found a considerable leakage from the split layer in the 2D CNN model. Furthermore, the authors mentioned that it is possible to reduce the distance correlation between the split layer and raw data by slightly scaling the weights of all layers before the split. This type of scaling works well in models with a large number of hidden layers before the split.

The work of Abuadbba *et al.* [6] is the first study exploring whether SL can deal with time-series data. It is dedicated to investigating *(i)* whether an SL can achieve the same model accuracy for a 1D CNN model compared to the non-split version and *(ii)* whether it can be used to protect privacy in sequential data. According to the results, SL can be applied to a model without the model classification accuracy degradation. As for the second question, the authors proved it is possible to reconstruct the raw data (personal ECG signal) in the 1D CNN model using SL by proposing a privacy assessment framework. They suggested three metrics: visual invertibility, distance correlation, and dynamic time warping. The results showed that when SL is directly adopted into 1D CNN models for time series data could result in significant privacy leakage. Two mitigation techniques were employed to limit the potential privacy leakage in SL: *(i)* increasing the number of layers before the split on the client-side and *(ii)* applying differential privacy to the split layer activation before sending the activation map to the server. However, both techniques suffer from a loss of model accuracy, particularly when differential privacy is used. The strongest differential privacy can increase the dissimilarity between the activation map and the corresponding raw data. However, *it degrades the classification accuracy significantly from 98.9% to 50%.*

In [6], during the forward propagation, the client sends the PAMs to the server, where the server can easily reconstruct the original raw data from the activated vector of the split layer leading to clear privacy leakage. In our work, we constructed a training protocol, where, instead of sending PAMs, the client first conducts an encryption using HE and then sends said maps to the server. In this way, the server is unable to reconstruct the original raw data, but can still perform a computation on the EAMs and realize the training process.

3. Architecture

In this section,we first describe the non-split version or local model of the 1D CNN used to classify the ECG

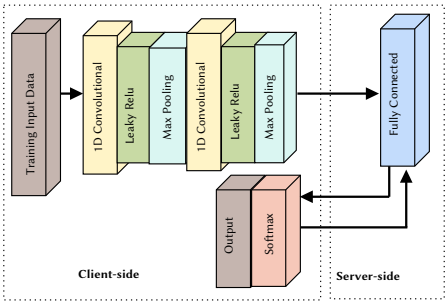

Figure 1: U-shaped Split-Learning

signal. Then, we discuss the process of splitting this local model into a U-shaped split model. Furthermore, we also describe the involved parties (a client and a server) in the training process of the split model, focusing on their roles and the parameters assigned to them throughout the training process.

3.1. 1D CNN Local Model Architecture

We first implement and successfully reproduce the local model results [6]. This model contains two Conv1D layers and two FC layers. The optimal test accuracy that this model achieves is 98.9%. We implement a simplified version where the model has one less FC layer compared to the model from [6]. Our local model consists of all the layer of Figure 1 without any split between the client and the server. As can be seen in Figure 1, we limit our model to two Conv1D layers and one linear layer as we aim to reduce computational costs when HE is applied on activation maps in the model's split version. Reducing the number of FC layers leads to a drop in the accuracy of the model. The best test accuracy we obtained after training our local model for 10 epochs with a batch size of 4 is 92.84%. *Although reducing the number of layers affects the model's accuracy, it is not within our goals to demonstrate how successful our ML model is for this task; instead, our focus is to construct a split model where training and evaluation on encrypted data are comparable to training and evaluation on plaintext data.*

In section 5, we detail the results for the non-split version and compare them with the split version.

3.2. U-shaped Split 1D CNN Model

The SL protocol consists of two parties: the client and server. We split the local 1D CNN into multiple parts, where each party trains its part(s) and communicates with others to complete the overall training procedure. More specifically, we construct the U-shaped split 1D CNN in such a way that the first few as well as the last layer are on the client-side, while the remaining layers are on the server-side.

Actors in the Split Learning Model As mentioned earlier, in our SL setting, we have two involved parties: the client and the server. Each party plays a specific role and has access to certain parameters. More specifically, their roles and accesses are described as:

- Client: In the plaintext version, the client holds two Conv1D layers and can access their weights and biases in plaintext. Other layers (Max Pooling layers, Leaky ReLU layers, Softmax layer) do not have weights and biases. Apart from these, in the HE encrypted version, the client is also responsible for generating the context for HE and has access to all context parameters (Polynomial modulus ($\mathcal{P}$), Coefficient modulus ($\mathcal{C}$), Scaling factor (Δ), Public key (pk) and Secret key (sk)). Note that for both training on plaintext and EAMs, the raw data examples $\mathbf{x}$'s and their corresponding labels $\mathbf{y}$'s reside on the client side and are never sent to the server during the training process.
- Server: In our model, the computation performed on the server-side is limited to only one linear layer. Hence, the server can exclusively access the weights and biases of this linear layer. Regarding the HE context parameters, the server has access to $\mathcal{P}$, $\mathcal{C}$, Δ, and pk shared by the client, with the exception of the sk. Not holding the sk, the server cannot decrypt the HE EAMs sent from the client. The hyperparameters shared between the client and the server are the learning rate (η), batch size (n), number of batches to be trained (N), and number of training epochs (E).

4. Split Model Training Protocols

In this section, we first present the protocol for training the U-shaped split 1D CNN on PAMs, followed by the protocol for training the U-shaped split 1D CNN on EAMs.

4.1. Training U-shaped Split Learning with Plaintext Activation Maps

We have used algorithm 1 and algorithm 2 to train the U-shaped split 1D CNN reported in subsection 3.2. First, the client and server start the socket initialization process and synchronize the hyperparameters η, n, N, E. They also initialize the weights ($\boldsymbol{w}^i$) and biases ($\boldsymbol{b}^i$) of their layers according to Φ.

During the forward propagation phase, the client forward-propagates the input $\mathbf{x}$ until the l^{th} layer and sends the activation $\mathbf{a}^{(l)}$ to the server. The server continues to forward propagate and sends the output $\mathbf{a}^{(L)}$ to the client. Next, the client applies the Softmax function

on $\mathbf{a}^{(L)}$ to get $\hat{\mathbf{y}}$ and calculates the error $J = \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$. The client starts the backward propagation by calculating

Algorithm 1: Client Side

Initialization:
$s \leftarrow$ socket initialized with port and address;
s.connect
$\eta, n, N, E \leftarrow s.synchronize()$
$\{\boldsymbol{w}^{(i)}, \boldsymbol{b}^{(i)}\}_{\forall i \in \{0..l\}} \leftarrow initialize\ using\ \Phi$
$\{\mathbf{z}^{(i)}\}_{\forall i \in \{0..l\}}, \{\mathbf{a}^{(i)}\}_{\forall i \in \{0..l\}} \leftarrow \emptyset$
$\left\{\frac{\partial J}{\partial \mathbf{z}^{(i)}}\right\}_{\forall i \in \{0..l\}}, \left\{\frac{\partial J}{\partial \mathbf{a}^{(i)}}\right\}_{\forall i \in \{0..l\}} \leftarrow \emptyset$
for $e \in E$ **do**
  for *each batch* $(\mathbf{x}, \mathbf{y})$ *generated from* D **do**
    Forward propagation :
      $O.zero\_grad()$
      $\mathbf{a}^0 \leftarrow \mathbf{x}$
    for $i \leftarrow 1$ *to* l **do**
      for $i \leftarrow 1$ **to** l **do**
        $\mathbf{z}^{(i)} \leftarrow f^{(i)}\left(\mathbf{a}^{(i-1)}\right)$
        $\mathbf{a}^{(i)} \leftarrow g^{(i)}\left(\mathbf{z}^{(i)}\right)$
    end
      $s.send\ (\mathbf{a}^{(l)})$
      $s.receive\ (\mathbf{a}^{(L)})$
      $\hat{y} \leftarrow Softmax\left(\mathbf{a}^{(L)}\right)$
      $J \leftarrow \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
    Backward propagation :
      Compute $\left\{\frac{\partial J}{\partial \hat{\mathbf{y}}}\ \&\ \frac{\partial J}{\partial \mathbf{a}^{(L)}}\right\}$
      $s.send\ \left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right)$
      $s.receive\ \left(\frac{\partial J}{\partial \mathbf{a}^{(l)}}\right)$
    for $i \leftarrow 1$ *to* l **do**
      Compute $\left\{\frac{\partial J}{\partial \boldsymbol{w}^{(i)}}, \frac{\partial J}{\partial \boldsymbol{b}^{(i)}}\right\}$
        Update $\boldsymbol{w}^{(i)}, \boldsymbol{b}^{(i)}$
    end
  end
end

and sending the gradient of the error w.r.t $\mathbf{a}^{(L)}$, i.e. $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$, to the server. The server continues the backward propagation, calculates $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$ and sends $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$ to the client. After receiving the gradients $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$ from the server, the backward propagation continues to the first hidden layer on the client-side. Note that the exchange of information between client and server in these algorithms takes place in plaintext. The client sends the activation maps $\mathbf{a}^{(l)}$ to the server in plaintext and receives the output of the linear layer $\mathbf{a}^{(L)}$ from the server in plaintext (see algorithm 1). The same applies on the server side: receiving $\mathbf{a}^{(l)}$ and sending $\mathbf{a}^{(L)}$ in the plaintext as can be seen in algorithm 2. Sharif *et al.* [6] showed that the exchange of PAMs between client and server using SL reveals important information regarding the client's raw sequential data. Later, in subsection 5.1 we show in detail how passing the forward activation maps from the client to the server in the plaintext will result in information leakage. To mitigate this privacy leakage, we propose the protocol, where the client encrypts the activation maps before sending them to the server, as described in subsection 4.2.

Algorithm 2: Server Side

Initialization:
$s \leftarrow$ socket initialized with port and address;
s.connect
$\eta, n, N, E \leftarrow s.synchronize()$
$\{\boldsymbol{w}^{(i)}, \boldsymbol{b}^{(i)}\}_{\forall i \in \{0..l\}} \leftarrow initialize\ using\ \Phi$
$\{\mathbf{z}^{(i)}\}_{\forall i \in \{l+1..L\}} \leftarrow \emptyset$
$\left\{\frac{\partial J}{\partial \mathbf{z}^{(i)}}\right\}_{\forall i \in \{l+1..L\}} \leftarrow \emptyset$
for $e \in E$ **do**
  for $i \leftarrow 1$ **to** N **do**
    Forward propagation :
      $O.zero\_grad()$
      $s.receive\ (\mathbf{a}^{(l)})$
      $\mathbf{a}^{(L)} \leftarrow f^{(i)}\left(\mathbf{a}^{(l)}\right)$
      $s.send\ \left(\mathbf{a}^{(L)}\right)$
    Backward propagation :
      $s.receive\ \left(\frac{\partial J}{\partial \mathbf{a}^{(L)}}\right)$
      Compute $\left\{\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}, \frac{\partial J}{\partial \boldsymbol{b}^{(L)}}\right\}$
      Update $\boldsymbol{w}^{(L)}, \boldsymbol{b}^{(L)}$
      Compute $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$
      $s.send\ \left(\frac{\partial J}{\partial \mathbf{a}^{(l)}}\right)$
  end
end

4.2. Training U-shaped Split 1D CNN with Encrypted Activation Maps

The protocol for training the U-shaped 1D CNN with a homomorphically EAP consists of four phases: initialization, forward propagation, classification, and backward propagation. The initialization phase only takes place once at the beginning of the procedure, whereas the other phases continue until the model iterates through all epochs. Each of these phases are described in detail in the following subsections.

Initialization The initialization phase consists of socket initialization, context generation, and random weight loading. The client first establishes a socket connection to the server and synchronizes the four hyperparameters $\eta,\ n,\ N, E$ with the server, shown in algorithm 3 and algorithm 4. These parameters must be synchronized on both sides to be trained in the same way. Also, the weights on the client and server are initialized with the same set of corresponding weights in the local model to accurately assess and compare the influence of SL on performance. On both the client and the server sides, $\boldsymbol{w}^{(i)}$ are initialized using corresponding parts of Φ. The activation map at layer i ($\mathbf{a}^{(i)}$), output tensor of a Conv1D layer ($\mathbf{z}^{(i)}$), and the gradients are initially set to zero. In this phase, the context generated is a specific object that holds encryption keys pk and sk of the HE scheme as well as additional parameters like $\mathcal{P}, \mathcal{C}$ and Δ.

Further information on the HE parameters and how to choose the best-suited parameters can be found in the TenSEAL's benchmarks tutorial[1]. As shown in algorithm 3 and algorithm 4, the context is either public ($\mathsf{ctx}_{\mathsf{pub}}$) or private ($\mathsf{ctx}_{\mathsf{pri}}$) depending on whether it holds the secret key sk. Both the $\mathsf{ctx}_{\mathsf{pub}}$ and $\mathsf{ctx}_{\mathsf{pri}}$ have the same parameters, though $\mathsf{ctx}_{\mathsf{pri}}$ holds a sk and $\mathsf{ctx}_{\mathsf{pub}}$ does not. The server does not have access to the sk as the client only shares the $\mathsf{ctx}_{\mathsf{pub}}$ with the server. After the initialization phase, both the client and server proceed to the forward and backward propagation phases.

Forward propagation The forward propagation starts on the client side. The client first zeroes out the gradients for the batch of data $(\mathbf{x}, \mathbf{y})$. He then begins calculating the $\mathbf{a}^{(l)}$ activation maps from $\mathbf{x}$, as can be seen in algorithm 3 where each $f^{(i)}$ is a Conv1D layer. The Conv1D layer can be described as following: given a 1D input signal that contains C channels, where each channel $\mathbf{x}_{(i)}$ is a 1D array ($i \in \{1, \ldots, C\}$), a Conv1D layer produces an output that contains C' channels. The j^{th} output channel $\mathbf{y}_{(j)}$, where $j \in \{1, \ldots, C'\}$ is:[2]

$$\mathbf{y}_{(j)} = \boldsymbol{b}_{(j)} + \sum_{i=1}^{C} \boldsymbol{w}_{(i)} \star \mathbf{x}_{(i)}, \tag{1}$$

where $\boldsymbol{w}_{(i)}, i \in \{1, \ldots, C\}$ are the weights, $\boldsymbol{b}_{(j)}$ are biases of the Conv1D layer, and $\star$ is the 1D cross-correlation operation. The $\star$ operation can be described as

$$\mathbf{z}(i) = (\boldsymbol{w} \star \mathbf{x})(i) = \sum_{j=0}^{m-1} \boldsymbol{w}(j) \cdot \mathbf{x}(i+j), \tag{2}$$

where $\mathbf{z}(i)$ denotes the i^{th} element of the output vector $\mathbf{z}$, and i starts at 0 and size of 1D weighted kernel is m.

[1] https://bit.ly/3KY8ByN

[2] https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html

In algorithm 3, $g^{(i)}$ can be seen as the combination of Max Pooling and Leaky ReLU functions. The final output activation maps of the l^{th} layer from the client is $\mathbf{a}^{(l)}$. The client then homomorphically encrypts $\mathbf{a}^{(l)}$ and sends the EAMs $\overline{\mathbf{a}^{(l)}}$ to the server. In algorithm 4, the server receives $\overline{\mathbf{a}^{(l)}}$ and then performs forward propagation, which is a linear layer evaluated on HE encrypted data $\overline{\mathbf{a}^{(l)}}$ as

$$\overline{\mathbf{a}^{(L)}} = \overline{\mathbf{a}^{(l)}} \boldsymbol{w}^{(L)} + \boldsymbol{b}^{(L)}. \tag{3}$$

After that, the server sends $\overline{\mathbf{a}^{(L)}}$ to the client (algorithm 4). Upon reception, the client decrypts $\overline{\mathbf{a}^{(L)}}$ to get $\mathbf{a}^{(L)}$, performs Softmax on $\mathbf{a}^{(L)}$ to produce the predicted output $\hat{\mathbf{y}}$ and calculate the loss J (algorithm 3). Having finished the forward propagation we may move on to the backward propagation part of the protocol.

Backward propagation After calculating the loss J, the client starts the backward propagation by computing $\frac{\partial J}{\partial \hat{\mathbf{y}}}$ and then $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ and $\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$ using the chain rule (algorithm 3). Specifically, the client calculates

$$\frac{\partial J}{\partial \mathbf{a}^{(L)}} = \frac{\partial J}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{a}^{(L)}}, \text{and} \tag{4}$$

$$\frac{\partial J}{\partial \boldsymbol{w}^{(L)}} = \frac{\partial J}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \boldsymbol{w}^{(L)}}. \tag{5}$$

Following, the client sends $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ and $\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$ to the server. Upon reception, the server computes $\frac{\partial J}{\partial \boldsymbol{b}}$ by simply doing $\frac{\partial J}{\partial \boldsymbol{b}} = \frac{\partial J}{\partial \mathbf{a}^{(L)}}$, based on equation (3). The server then updates the weights and biases of his linear layer according to equation (6).

$$\boldsymbol{w}^{(L)} = \boldsymbol{w}^{(L)} - \eta \frac{\partial J}{\partial \boldsymbol{w}^{(L)}}, \quad b^{(L)} = \boldsymbol{b}^{(L)} - \eta \frac{\partial J}{\partial \boldsymbol{b}^{(L)}}. \tag{6}$$

Next, the server calculates

$$\frac{\partial J}{\partial \mathbf{a}^{(l)}} = \frac{\partial J}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{a}^{(l)}}, \tag{7}$$

and sends $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$ to the client. After receiving $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$, the client calculates the gradients of J with respect to the weights and biases of the Conv1D layer using the chain-rule, which can generally be described as

$$\frac{\partial J}{\partial \boldsymbol{w}^{(i-1)}} = \frac{\partial J}{\partial \boldsymbol{w}^{(i)}} \frac{\partial \boldsymbol{w}^{(i)}}{\partial \boldsymbol{w}^{(i-1)}} \tag{8}$$

$$\frac{\partial J}{\partial \boldsymbol{b}^{(i-1)}} = \frac{\partial J}{\partial \boldsymbol{b}^{(i)}} \frac{\partial \boldsymbol{b}^{(i)}}{\partial \boldsymbol{b}^{(i-1)}} \tag{9}$$

Finally, after calculating the gradients $\frac{\partial J}{\partial \boldsymbol{w}^{(i)}}$, $\frac{\partial J}{\partial \boldsymbol{b}^{(i)}}$, the client updates $\boldsymbol{w}^{(i)}$ and $\boldsymbol{b}^{(i)}$ using the Adam optimization algorithm [11].

Algorithm 3: Client Side

Context Initialization:
$\mathsf{ctx}_{\mathsf{pri}} \leftarrow \mathcal{P}, \mathcal{C}, \Delta, \mathsf{pk}, \mathsf{sk}$
$\mathsf{ctx}_{\mathsf{pub}} \leftarrow \mathcal{P}, \mathcal{C}, \Delta, \mathsf{pk}$
$s.send(\mathsf{ctx}_{\mathsf{pub}})$
for e *in* E **do**
  for *each batch* $(\mathbf{x}, \mathbf{y})$ *generated from* $\mathbf{D}$ **do**
    Forward propagation :
      $O.zero\_grad()$
      $\mathbf{a}^0 \leftarrow \mathbf{x}$
    for $i \leftarrow 1$ **to** l **do**
      $\mathbf{z}^{(i)} \leftarrow f^{(i)}\left(\mathbf{a}^{(i-1)}\right)$
      $\mathbf{a}^{i} \leftarrow g^{(i)}\left(\mathbf{z}^{(i)}\right)$
    end
      $\overline{\mathbf{a}^{(l)}} \leftarrow \mathsf{HE.Enc}\left(\mathsf{pk}, \mathbf{a}^{(l)}\right)$
      $s.send\ \left(\overline{\mathbf{a}^{(l)}}\right)$
      $s.receive\ \left(\overline{\mathbf{a}^{(L)}}\right)$
      $\mathbf{a}^{(L)} \leftarrow \mathsf{HE.Dec}\left(\mathsf{sk}, \overline{\mathbf{a}^{(L)}}\right)$
      $\hat{\mathbf{y}} \leftarrow Softmax\left(\mathbf{a}^{(L)}\right)$
      $\mathbf{J} \leftarrow \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
    Backward propagation :
      Compute $\left\{\frac{\partial J}{\partial \hat{\mathbf{y}}} \& \frac{\partial J}{\partial \mathbf{a}^{(L)}} \& \frac{\partial J}{\partial \boldsymbol{w}^{(L)}}\right\}$
      $s.send\left(\frac{\partial J}{\partial \mathbf{a}^{(L)}} \& \frac{\partial J}{\partial \boldsymbol{w}^{(L)}}\right)$
      $s.receive\left(\frac{\partial J}{\partial \mathbf{a}^{(l)}}\right)$
    for $i \leftarrow l$ *down to* 1 **do**
      Compute $\left\{\frac{\partial J}{\partial \boldsymbol{w}^{(i)}}, \frac{\partial J}{\partial \boldsymbol{b}^{(i)}}\right\}$
      Update $\boldsymbol{w}^{(i)}, \boldsymbol{b}^{(i)}$
    end
  end
end

Algorithm 4: Server Side

Context Initialization:
$s.receive(\mathsf{ctx}_{\mathsf{pub}})$
for $\mathbf{e}$ *in* $\mathbf{E}$ **do**
  for $i \leftarrow 1$ **to** N **do**
    Forward propagation :
      $O.zero\_grad()$
      $s.receive\ \left(\overline{\mathbf{a}^{(l)}}\right)$
      $\overline{\mathbf{a}^{(L)}} \leftarrow \mathsf{HE.Eval}\left(f^{(i)}\left(\overline{\mathbf{a}^{(l)}}\right)\right)$
      $s.send\left(\overline{\mathbf{a}^{(L)}}\right)$
    Backward propagation :
      $s.receive\left\{\frac{\partial J}{\partial \mathbf{a}^{(L)}} \& \frac{\partial J}{\partial \boldsymbol{w}^{(L)}}\right\}$
      Compute $\frac{\partial J}{\partial \boldsymbol{b}^{(L)}}$
      Update $\boldsymbol{w}^{(L)}, \boldsymbol{b}^{(L)}$
      Compute $\frac{\partial J}{\partial \mathbf{a}^{(l)}}$
      $s.send\left(\frac{\partial J}{\partial \mathbf{a}^{(l)}}\right)$
  end
end

Note that in the backward pass, by sending both $\frac{\partial J}{\partial \mathbf{a}^{(L)}}$ and $\frac{\partial J}{\partial \boldsymbol{w}^{(L)}}$ to the server, we help the server keep his parameters in plaintext and prevent the multiplicative depth of the HE from growing out of bound, however, this leads to a privacy leakage of the activation maps.

5. Performance Analysis

We evaluate our method on the MIT-BIH dataset [8].

MIT-BIH We use the pre-processed dataset from [6], which is based on the MIT-BIH arrhythmia (abnormal heart rhythm) database [8]. The processed dataset contains 26,490 samples of heartbeat that belong to 5 different types: N (normal beat), L (left bundle branch block), R (right bundle branch block), A (atrial premature contraction), V (ventricular premature contraction). An example heartbeat of each class is visualized in Figure 2.

To train our network, the dataset is then split into a train and test split according to [6]. This results in both the train and test split as matrices of size $[13245, 1, 128]$, meaning that they contain 13,245 ECG samples, each sample has one channel and 128 timesteps.

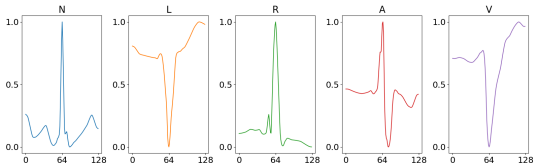

Figure 2: Heartbeats from the processed ECG dataset.

Experimental Setup All neural networks are trained on a machine with Ubuntu 20.04 LTS, processor Intel Core i7-8700 CPU at 3.20GHz, 32Gb RAM, GPU GeForce GTX 1070 Ti with 8Gb of memory. We write our program in the Python programming language version 3.9.7. The neural nets are constructed using the PyTorch library version 1.8.1+cu102. For HE algorithms, we employ the TenSeal library version 0.3.10. We perform our experiments in the localhost setting. The open source

implementation of our work is publically available[3].

In terms of hyperparameters, we train all networks with 10 epochs, $\eta = 0.001$ learning rate, and $n = 4$ training batch size. For the split neural network with HE activation maps, we use the Adam optimizer for the client model and mini-batch Gradient Descent for the server. We use GPU for networks trained on the plaintext. For the U-shaped SL model on HE activation maps, we train the client model on GPU, and the server model on CPU.

5.1. Evaluation

In this section, we report the experimental results in terms of accuracy, training duration and communication throughput. We measure the accuracy of the neural nets on the plaintext test set after the training processes are completed. The 1D CNN models used on MIT-BIH dataset have two Conv1D layers and one linear layer. The activation maps are the output of the last Conv1D layer.

We experiment with the activation maps of $[\text{batch size}, 256]$ for the MIT-BIH dataset. We denote the 1D CNN model with an activation map sized $[\text{batch size}, 256]$ as M_1.

Training Locally Results when training M_1 locally on the MIT-BIH plaintext dataset are shown in Figure 3. The neural network learns quickly and is able to decrease the loss drastically from epoch 1 to 5. From epoch 6-10, the loss begins to plateau. After training for 10 epochs, we test the trained neural network on the test dataset and get 88.06% accuracy. Training the model locally on plaintext takes 4.8sec for each epoch on average.

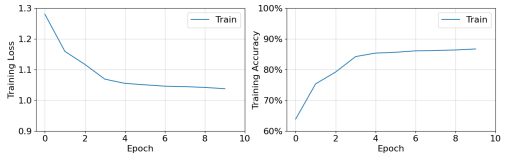

Figure 3: Results when training locally on the plaintext MIT-BIH dataset with activation maps of size $[\text{batch size}, 256]$.

U-shaped Split Learning using Plaintext Activation Maps Our experiments, show that training the U-shaped split model on plaintext (reported in section 3.2) produces the same results in terms of accuracy compared to local training for model M_1. This result is similar to the findings of [6]. Even though the authors of [6] only used the vanilla version of the split model, they too found that, compared to training locally, accuracy was not reduced.

We will now discuss the training time and communication overhead of the U-shaped split models and compare them to their local versions. For the split version of M_1, each training epoch takes 8.56 seconds on average, hence 43.9% longer than local training. The U-shaped split models take longer to train due to the communication between the client and the server. The communication cost for one epoch of training split M_1 is 33.06 Mb.

Visual Invertibility In the SL model, the activation maps are sent from client to server to continue the training process. A visual representation of the activation maps reveals a high similarity between certain activation maps and the input data from the client, as demonstrated in Figure 4 for the models trained on the MIT-BIH dataset. The figure indicates that, compared to the raw input data from the client (the first row of Figure 4), some activation maps (as plotted in the second row of Figure 4) have exceedingly similar patterns. This phenomenon clearly compromises the privacy of the client's raw data. The authors of [6] quantify the privacy leakage by measuring the correlations between the activation maps and the raw input signal by using two metrics: distance correlation and Dynamic Time Warping. This approach allows them to measure whether their solutions mitigate privacy leakage work. Since our work uses HE, said metrics are unnecessary as the activation maps are encrypted.

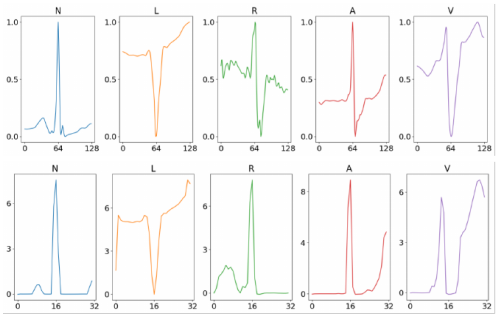

Figure 4: Top: client input data. Bottom: one of the output channels from the M_1 model's second convolution layer.

U-shaped Split 1D CNN with Homomorphic Encrypted Activation Maps We train the split neural networks M_1 on the MIT-BIH dataset using EAMs according to subsection 4.2. To encrypt the activation maps on client side (i.e. before sending them to the server), we experiment with five different sets of HE parameters for model M_1. Additionally, we perform experiments using different combinations of HE parameters. Table 1 shows the results in terms of training time, testing accuracy, and communication overhead for the neural networks with different configurations. For the U-shaped SL version on the plaintext, we captured all communication between client and server. For training split models on EAPs, we approximate the communication overhead for one training epoch by getting the average communication of training on the first ten batches of data, then multiply that with the total number of training batches.

[3] https://github.com/khoaguin/HESplitNet

Table 1
Training and testing results on the MIT-BIH dataset. Training duration and communication are reported per epoch.

Network	Type of Network	HE Parameters				Training duration (s)	Test accuracy (%)	Communication (Tb)
		BE	$\mathcal{P}$	$\mathcal{C}$	Δ			
M_1	Local					4.80	88.06	0
	Split (plaintext)					8.56	88.06	33.06e-6
	Split (HE)	False	8192	[60,40,40,60]	2^{40}	50 318	85.31	37.84
			8192	[40,21,21,40]	2^{21}	48 946	80.63	22.42
			4096	[40,20,20]	2^{21}	14 946	85.41	4.49
			4096	[40,20,40]	2^{20}	18 129	80.78	4.57
			2048	[18,18,18]	2^{16}	5 018	22.65	0.58

For the M_1 model, the best test accuracy was 85.41%, when using the HE parameters with polynomial modulus $\mathcal{P} = 4096$, coefficient modulus $\mathcal{C} = [40, 20, 20]$, scale $\Delta = 2^{21}$. The accuracy drop was 2.65% compared to training the same network on plaintext. This set of parameters achieves higher accuracy compared to the bigger sets of parameters with $\mathcal{P} = 8192$, while requiring much lower training time and communication overhead. The result when using the first set of parameters with $\mathcal{P} = 8192$ is close (85.31%), but with a much longer training time (3.67 times longer) and communication overhead (8.43 times higher).

Uur experiments show that training on EAMs can produce optimistic results, with accuracy dropping by 2-3% for the best sets of HE parameters.

The set of parameters with $\mathcal{P} = 8192$ achieve the second highest test accuracy, though incurring the highest communication overhead and the longest training time. The set of parameters with $\mathcal{P} = 4096$ can offer a good trade-off as they can produce on-par accuracy with $\mathcal{P} = 8192$, while requiring significantly less communication and training time. Experimental results show that with the smallest set of HE parameters $\mathcal{P} = 2048$, $\mathcal{C} = [18, 18, 18]$, $\Delta = 2^{16}$, the least amount of communication and training time is required.

6. Conclusion

This paper focused on how to train ML models in a privacy-preserving way using a combination of split learning and homomorphic encryption. We constructed protocols by which a client and a server could collaboratively train a model without revealing significant information about the raw data. As far as we are aware, this is the first time split learning is used on encrypted data.

Acknowledgments

This work was funded by the HARPOCRATES EU research project (No. 101069535) and the Technology Innovation Institute (TII), UAE, for the project ARROWSMITH.

References

[1] T. Khan, A. Bakas, A. Michalas, Blind faith: Privacy-preserving machine learning using function approximation, in: 2021 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2021, pp. 1–7.

[2] P. Vepakomma, O. Gupta, A. Dubey, R. Raskar, Reducing leakage in distributed deep learning for sensitive health data, arXiv:1812.00564 (2019).

[3] A. Singh, P. Vepakomma, O. Gupta, R. Raskar, Detailed comparison of communication efficiency of split learning and federated learning, arXiv preprint arXiv:1909.09145 (2019).

[4] P. Vepakomma, O. Gupta, T. Swedish, R. Raskar, Split learning for health: Distributed deep learning without sharing raw patient data, arXiv preprint arXiv:1812.00564 (2018).

[5] J. H. Cheon, A. Kim, M. Kim, Y. Song, Homomorphic encryption for arithmetic of approximate numbers, in: International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2017, pp. 409–437.

[6] S. Abuadbba, K. Kim, M. Kim, C. Thapa, S. A. Camtepe, Y. Gao, H. Kim, S. Nepal, Can we use split learning on 1d cnn models for privacy preserving training?, in: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, 2020, pp. 305–318.

[7] M. Wooldridge, The Road to Conscious Machines: The Story of AI, Pelican Books, Penguin Books Limited, 2020.

[8] G. B. Moody, R. G. Mark, The impact of the mit-bih arrhythmia database, IEEE Engineering in Medicine and Biology Magazine 20 (2001) 45–50.

[9] O. Gupta, R. Raskar, Distributed learning of deep neural network over multiple agents, Journal of Network and Computer Applications 116 (2018).

[10] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, H. Yu, Federated learning, Synthesis Lectures on Artificial Intelligence and Machine Learning 13 (2019) 1–207.

[11] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).