Mnemonist: Locating Model Parameters that Memorize Training Examples

Ali Shahin Shamsabadi[1] **Jamie Hayes**[2] **Borja Balle**[2] **Adrian Weller**[1,3]

[1]The Alan Turing Institute
[2]Google DeepMind
[3]University of Cambridge

Abstract

Recent work has shown that an adversary can reconstruct training examples given access to the parameters of a deep learning image classification model. We show that the quality of reconstruction depends heavily on the type of activation functions used. In particular, we show that ReLU activations lead to much lower quality reconstructions compared to smooth activation functions. We explore if this phenomenon is a fundamental property of models with ReLU activations, or if it is a weakness of current attack strategies. We first study the training dynamics of small MLPs with ReLU activations and identify redundant model parameters that do not memorise training examples. Building on this, we propose our Mnemonist method, which is able to detect redundant model parameters, and then guide current attacks to focus on informative parameters to improve the quality of reconstructions of training examples from ReLU models.

1 INTRODUCTION

Machine Learning (ML) models have the capacity to memorize examples from training data [Zhang et al., 2017]. Consequently, releasing ML models can be a risk to privacy if the training data is sensitive. In the most serious example of a privacy breach, verbatim examples of training data points can be reconstructed [Haim et al., 2022, Balle et al., 2022, Guo et al., 2022, Fowl et al., 2022]. For example, it has been shown that an informed adversary with knowledge of all the data points in a training set except one (the target point) can reconstruct the target point if they have access to the model parameters [Balle et al., 2022]. To do this, the informed adversary trains a Reconstructor Neural Network (RecoNN) that receives *all* parameters from the model as the input and reconstructs the target point as the output.

We show that this adversary will not successfully reconstruct training examples with high probability if the model is trained with ReLU [Glorot et al., 2011] activation functions. This is a significant weakness of the attack since ReLU activations are very common. Their popularity is in part because they often yield superior performance over models trained with other activation functions, and also due to their faster convergence rates [Krizhevsky et al., 2012, Glorot et al., 2011]. ReLU activations also negatively affect the success of attacks in other settings such as federated learning [Wei et al., 2020] where the attack relies on access to intermediate model updates. Haim et al. [2022] replaced ReLU activations with Sigmoid activations to run their training data reconstruction attack because ReLU "contains flat regions which are hard to optimize". In order to understand the extent to which models using ReLU activations are vulnerable to reconstruction attacks, we ask the following questions:

Q1: Why do ReLU activations lead to much lower quality reconstructions compared to smooth activations?
Q2: How can we improve the quality of the reconstruction of target points from models with ReLU activations?

Our approach to answering these questions is to learn how important each parameter of the model is to the success of the reconstruction of the target point. We then design new attack methods that shift the focus of the attack towards the parameters that are identified as important.

First, we analytically demonstrate that *not all* parameters in ReLU activated models store information about the target point. Intuitively, this is because ReLU deactivates neurons with negative outputs in the forward pass – later in the backward pass these non-activated neurons prevent their incoming parameters from being updated by making the gradient of the loss with respect to the input zero, thus no information about the input is stored in incoming parameters to non-activated neurons. This is not the case for models with smooth activations (including Sigmoid) as their derivatives are always nonzero. We empirically demonstrate this behaviour by studying the training dynamic of the models,

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

namely the per-example gradient and number of training examples stored in each parameter across all shadow models.

Second, we design an approach we call Mnemonist, which distinguishes between parameters that contain no information about the target point and parameters that contain a lot of information about the target point. Mnemonist is black-box in the sense that it does not need access to intermediate model updates throughout training, and is instantiated by extending the approach of Local Interpretable Model-Agnostic Explanations (LIME) [Ribeiro et al., 2016] to operate in the parameter space. Mnemonist starts by semantically grouping parameters into *superparameters*. These superparameters each represent all incoming parameters to each individual neuron. Then, Mnemonist learns the importance of each superparameter for data reconstruction as coefficients of an interpretable model which is trained on the response of RecoNN to the absence or presence of each superparameter.

Finally, based on Mnemonist, we introduce an attack, called Mnemonist-RecoNN, that can improve the quality of reconstructions of target points from models trained with ReLU activations by training RecoNN on *only* informative parameters. We highlight the following contributions:

- We characterise the fundamental property of the existence of redundant parameters in models with ReLU activations through both theoretically and empirically analysing their training dynamics.
- We propose a black-box explanation technique, Mnemonist, which identifies parameters that are likely to store information about the target point we aim to reconstruct. We provide theoretical and empirical justifications for the performance of Mnemonist.
- We show that naively applying the attack proposed by Balle et al. [2022] on models with ReLU activations results in much lower quality reconstruction than smooth activations. We improve the quality of reconstruction by applying the attack to only a subset of parameters that are identified by Mnemonist as informative.

2 THREAT MODEL AND SETUP

Recent work on training data reconstruction attacks has focused on attacking federated learning set-ups where an adversary has access to all intermediate model updates [Boenisch et al., 2023, Wen et al., 2022, Fowl et al., 2022]. Similar to the threat model considered by Balle et al. [2022], we assume the adversary does not have access to intermediate model updates; the adversary can only observe the initial and final model parameters. This restriction on the adversary means that our work is applicable to approaches beyond federated learning.

More formally, we assume a model developer trains an ML model on a supervised learning task, which we refer to as the *released* model. They use an off-the-shelf optimization algorithm such as SGD with momentum to transform a set of initial model parameters, $\mathbf{W}_{\text{init}}$, to a set of final model parameters $\mathbf{W}$, by training for T steps on a dataset $D_{-} \cup \{z\}$, where D_{-} is referred to as the *fixed dataset*, z is the target point, $|D_{-}| = N - 1$ and both z and all points in D_{-} are sampled from an input space $\mathcal{Z}$. Table 1 describes all necessary notation used throughout this paper.

Table 1: Notation.

	Meaning		Meaning
$\mathbf{W}_{\text{init}}$	Initial model param.	$\mathbf{W}$	Released model param.
$\mathbf{W}_i$	Shadow model param.	$\bar{z}$	A public target example
$\mathbf{w}^{(s)}$	Model superparam.	$\mathbf{W}'$	Perturbed model
M	# perturbed models	$\mathbf{b}$	binary mask
D_{-}	Fixed dataset	$N-1$	Size of fixed set
K	# shadow models	RecoNN	Reconstructed network
T	# training steps	$\mathcal{S}$	Adversary side knowledge
$\mathcal{L}_{Rec}$	Reconstruction loss	$\boldsymbol{\phi}$	RecoNN parameters

Following the terminology used by Balle et al. [2022], we assume the adversary is *informed*. That is, the adversary has knowledge of the tuple (D_{-}, $\mathbf{W}_{\text{init}}$, $\mathbf{W}$, $\mathcal{S}$), where $\mathcal{S}$ represents the side-knowledge available to the adversary. We assume $\mathcal{S}$ includes all training hyperparameters such as T, initial model parameters $\mathbf{W}_{\text{init}}$, the size of mini-batch, the optimizer and the learning rate of the optimizer. However, we do not assume the adversary knows the randomness used to sample mini-batches from $D_{-} \cup \{z\}$ at each step, nor do they have access to intermediate model parameters. The goal of the adversary is to reconstruct the target point z given (D_{-}, $\mathbf{W}_{\text{init}}$, $\mathbf{W}$, $\mathcal{S}$). To do this, we run the attack proposed by Balle et al. [2022]. This attack is designed based on the intuition that the impact of the private target point z on the released model $\mathbf{W}$ trained on $D_{-} \cup \{z\}$ is similar to the impact of a public target point $\bar{z}$ on a shadow model $\bar{\mathbf{W}}$ trained on $D_{-} \cup \{\bar{z}\}$. In particular, the attack consists of the following three stages [Balle et al., 2022]:

1. **Training shadow models to collect information about the impact of training examples on model parameters.** We assume the adversary has access to a public dataset $\bar{Z} = \{\bar{z}_i\}_{i=1}^{K}$ containing K data points disjoint from $D_{-} \cup \{z\}$. We train K shadow models, $\{\bar{\mathbf{W}}_i\}_{i=1}^{K}$, where each shadow model $\bar{\mathbf{W}}_i$ is trained on the fixed dataset plus the i-th public data point, $D_{-} \cup \{\bar{z}_i\}$ using side-knowledge $\mathcal{S}$ (including the same initial parameters $\mathbf{W}_{\text{init}}$ and optimizer as the ones used for the released model).
2. **Training a Reconstructor Neural Network to output training examples from model parameters.** We train a Reconstructor Neural Network,

RecoNN$(\cdot)$, whose inputs lie in the parameter space of shadow models and outputs lie in the input space of the shadow models. In particular, the RecoNN receives $\mathbf{W}_i$ as an input and tries to reconstruct its corresponding target point $\bar{z}_i$ as an output by minimizing the Mean Squared Error (MSE) and Mean Absolute Error (MAE) between the target $\bar{z}_i$ and its reconstruction $\text{RecoNN}(\mathbf{W}_i)$, as $\mathcal{L}_{Rec} = \text{MSE}\big(\text{RecoNN}(\mathbf{W}_i), \bar{z}_i\big) + \text{MAE}\big(\text{RecoNN}(\mathbf{W}_i), \bar{z}_i\big)$.

3. **Producing a candidate reconstruction for the target point.** We obtain a reconstruction candidate for the target point z by inputting the released model parameters $\mathbf{W}$ to the trained RecoNN.

While the assumption that such an informed adversary exists is perhaps unrealistic for practical attacks, the attacks we study in this work are designed to reveal the maximum amount of privacy leakage that could be revealed to such an adversary. As such, our work is similar in spirit to the long list of research on auditing the privacy of machine learning models [Lu et al., 2022, Nasr et al., 2021, Jagielski et al., 2020, Zanella-Béguelin et al., 2023], and should be viewed as complementary to research on reconstructing training data from federated learning systems, which leans more on the practical side. Finally, we note that recent work by Haim et al. [2022] also investigates reconstructing training data without adversarial access to intermediate model updates. However, the focus of their work is untargeted in that they try and reconstruct *any* training data points, while our work aims to reconstruct a specific training example. Furthermore, their reconstruction attacks are confined to simple linear models, while our attacks operate on fully-connected neural networks; Balle et al. [2022] have already shown that closed form solutions for reconstruction attacks with informed adversaries exist on convex models.

Setup. We focus on fully-connected neural networks (FCNN) and CIFAR10 following the baseline approach of Balle et al. [2022][1]. Our experimental setup is summarised in Table 2 and unless stated otherwise, all experimental results are reported by averaging across 1,000 reconstructions where the targets are selected from the CIFAR10 test set. Regarding performance measures, we quantify the quality of reconstruction by computing the MAE+MSE between the target point and its reconstructed point.

3 RECONSTRUCTING TRAINING DATA FROM MODELS WITH RELU ACTIVATIONS IS HARD

We examine issues associated with training data reconstructions mounted against models with ReLU activations. In particular, we compare the effect of changing the model activation functions from Sigmoid[2] to ReLU on the quality of reconstructing target points from model parameters.

Table 2: Experimental setup.

Released/Shadow models		
Architecture	Optimizer	#steps
FCNN (layer:4, width:10)	SGD+Momentum (Full-Batch)	T=40

CIFAR10 dataset		RecoNN		
Fixed size	Shadow size	Architecture	Loss	#steps
$N-1$=10k	K=40k	Transposed CNN	MAE+MSE	200

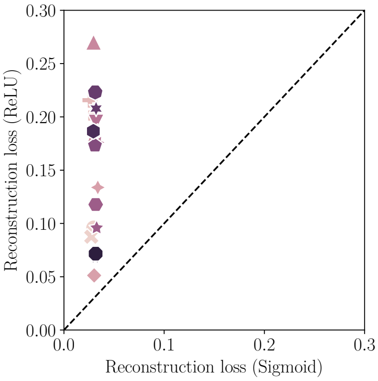

Figure 1: Reconstruction loss of target points in ReLU versus Sigmoid activated models across 20 different initializations. ReLU activations lead to higher reconstruction loss and higher variance compared to Sigmoid activations.

Figure 1 shows the impact of the activation function in the released model on the loss of reconstructing target points from the final released model parameters across initializations ($\mathbf{W}_{\text{init}}$). The results demonstrate that the reconstruction of target points from ReLU activated released models is, in general, harder than from Sigmoid activated released models. Examples of reconstructions are visualised in Figure 2 to help the interested reader calibrate how numeric reconstruction losses map to the visual quality of reconstructions; in general, one can confidently pair the reconstruction with the target if it has a reconstruction loss smaller than 0.15. The quality of target points reconstructed from Sigmoid activated models are better than the quality of target points reconstructed from ReLU activated models. Figure 1 also shows that the variation of the gap between ReLU and Sigmoid due to the randomness of the parameter initialization is large. The reconstruction loss of models with ReLU activations ranges from 0.05 to 0.29, while Sigmoid activations lead to a low magnitude and narrow range of around 0.03.

We consider the quality of reconstruction in the Sigmoid

[1] See Section 6 for a discussion on the choice of the dataset and model architectures.

[2] Other smooth activation functions give similar results to Sigmoid in terms of data reconstruction (Table V in Balle et al. [2022]).

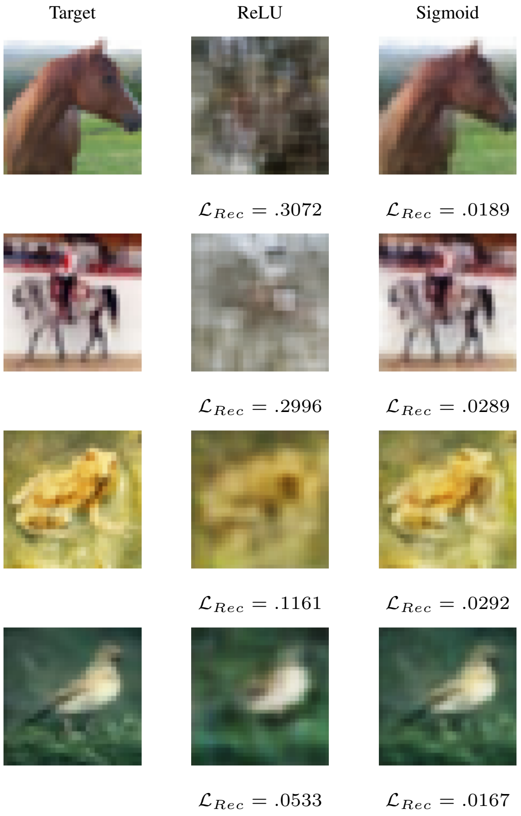
Target
ReLU
Sigmoid
$\mathcal{L}_{Rec}$ = .3072
$\mathcal{L}_{Rec}$ = .0189
$\mathcal{L}_{Rec}$ = .2996
$\mathcal{L}_{Rec}$ = .0289
$\mathcal{L}_{Rec}$ = .1161
$\mathcal{L}_{Rec}$ = .0292
$\mathcal{L}_{Rec}$ = .0533
$\mathcal{L}_{Rec}$ = .0167

Figure 2: Original target example and their reconstruction candidates obtained by training RecoNN on ReLU and Sigmoid activated fully connected models. The quality of reconstructing target examples from Sigmoid activated models is closer to the quality of the original target example.

case to be the benchmark, and identify intializations for released models that lead to a small or large gap between ReLU and Sigmoid reconstruction losses. We refer these two groups as *good* and *bad* initializations. Using these two initialization groups, we study the reconstruction loss of individual examples (i.e., per-sample reconstructions) as well as per-class reconstructions in which the reconstruction losses of data points belonging to the same class are aggregated.

Per-class reconstruction losses in Figure 3 show that, in general, samples belonging to all classes are harder to reconstruct in the ReLU case than the Sigmoid case. For "good" initializations, ReLU activations still lead to larger reconstruction losses than Sigmoid activations, however, this gap is much larger for "bad" initializations. Reconstruction losses across classes vary slightly more in the ReLU case than the Sigmoid case, and this variation increases with "bad" initializations. Across both initializations and activations, we observe that some data points and classes are inherently more difficult to reconstruct than others – possibly because some classes are less complex e.g. the airplane class has many images with blue skies while the truck class images have more intricate backgrounds on average.

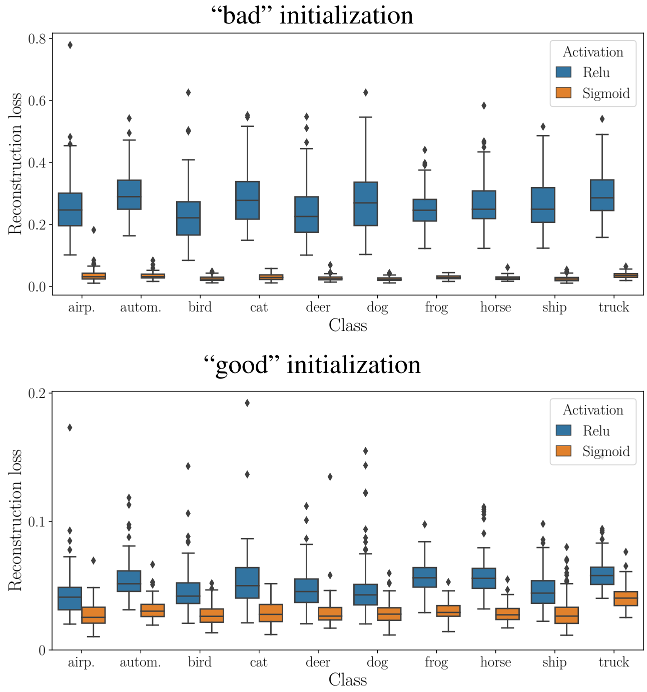

Figure 3: Impact of the choice of the activation function on the per-class reconstruction loss across two different initializations. Reconstructing target points from models with ReLU activations, independent of the class that they belong to, is less successful than those with Sigmoid activations.

Figure 4 shows the effect of changing the released model activations from Sigmoid to ReLU on per-sample reconstruction losses using both "good" and "bad" initializations. In general, the histogram of per-sample reconstruction loss in Sigmoid is more condensed than the ReLU ones. We plot the histogram across initializations using the same x-axis scale in the first column of Figure 4 to highlight that the spread of reconstruction losses on ReLU models drops significantly when transitioning from "bad" to "good" initializations. For "good" initializations there is a large overlap between ReLU and Sigmoid reconstruction losses, which is not the case for "bad" initializations. The second column of Figure 4 shows that a few samples are reconstructed better in the ReLU case than the Sigmoid case when we use the "good" initialization.

4 WHY IS TRAINING DATA RECONSTRUCTION FROM RELU ACTIVATED MODELS HARD?

We theoretically and empirically analyze why ReLU activations lead to higher reconstruction loss and variations across parameter initialization compared to the Sigmoid activation.

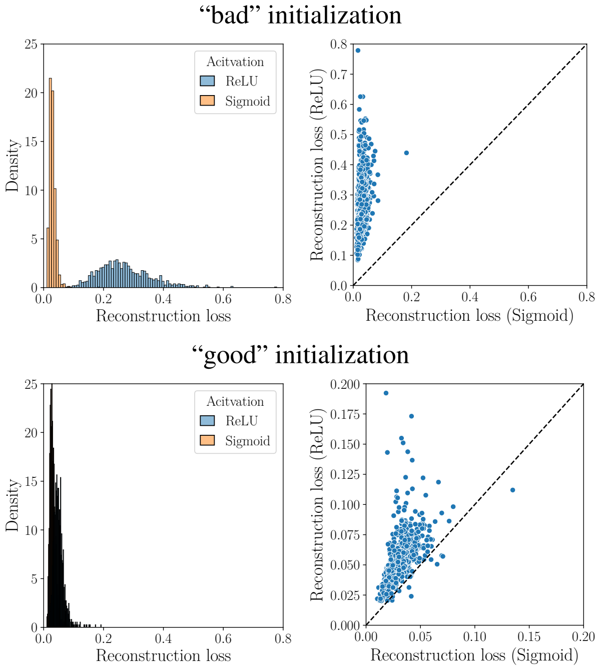

Figure 4: Comparing the effect of changing the activation function from Sigmoid to ReLU on the per-sample reconstruction loss using “bad” initialization and “good” initializations. ReLU activations lead to higher per-sample reconstruction loss compared to Sigmoid activations especially in the “bad” initialization case.

4.1 EXISTENCE OF REDUNDANT MODEL PARAMETERS IN THEORY

Consider a fully connected layer that receives an input $\mathbf{x} \in \mathbb{R}^{n_x}$ and outputs the activation $\mathbf{a} \in \mathbb{R}^{n_h} = R(\mathbf{h})$ which is computed by applying a non-linear activation function $R(\cdot)$ on the pre-activation $\mathbf{h} \in \mathbb{R}^{n_h} = \mathbf{W}\mathbf{x} + \mathbf{b}$. Parameters $\mathbf{W} \in \mathbb{R}^{n_h \times n_x}$ and bias $\mathbf{b} \in \mathbb{R}^{n_h}$ are initialized randomly. The parameter matrix $\mathbf{W}$ contains as many rows as the number of neurons at the output of the fully connected layer such that each row $\mathbf{w}^l$ denotes all the edges connecting $\mathbf{x}$ to an output neuron h^l. At each training step t, each row $\mathbf{w}^l$ is updated based on the gradient of the loss $\mathcal{L}$ w.r.t. this row as $\mathbf{w}_{t+1}^l = \mathbf{w}_t^l - \text{lr}\frac{\partial \mathcal{L}}{\partial \mathbf{w}_t^l}$, where lr is the learning rate. The gradient is obtained as

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{w}_t^l} &= \frac{\partial \mathcal{L}}{\partial a_t^l} \cdot \frac{\partial a_i^l}{\partial h_t^l} \cdot \frac{\partial h_t^l}{\partial \mathbf{w}_t^l} \\ &= \frac{\partial \mathcal{L}}{\partial a_t^l} \cdot R'(h_t^l) \cdot \mathbf{x} \ \ , \end{aligned} \tag{1}$$

where R' is the derivative of the activation function R.

For the Sigmoid activation function, R' is always non-zero, thus each row stores a copy of $\mathbf{x}$. However, recall that for the ReLU activation we have $R'(h) = 0$ whenever $h \leq 0$, and $R'(h) = 1$ otherwise. This means that

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_t^l} = \begin{cases} 0 & \text{if} \quad h_t^l \leq 0 \\ \mathit{scale} \cdot \mathbf{x} & \text{otherwise} \ \ , \end{cases} \tag{2}$$

where $\mathit{scale} = \frac{\partial \mathcal{L}}{\partial a_t^l}$. In particular, when h_t^l is positive, the update step will store a copy of $\mathbf{x}$ (proportional to $\text{lr}\frac{\partial \mathcal{L}}{\partial a_t^l}$) in $\mathbf{w}_{t+1}^l$, but otherwise the parameter update will be independent of the input $\mathbf{x}$. In the latter case, the update of row $\mathbf{w}^l$ does not store any information useful to perform a reconstruction attack against target $\mathbf{x}$ – we call such a row *redundant*. The existence of these redundant rows decreases the quality of reconstruction. In addition to this, the number of redundant parameters varies across different initializations, resulting in high variance. Next we empirically investigate how these redundant parameters manifest during training of ReLU activated FCNN.

4.2 EXISTENCE OF REDUNDANT MODEL PARAMETERS IN PRACTICE

We study the training dynamics of ReLU activated models versus those of Sigmoid activated models. We focus on the first layer where each row of parameters can store a scaled version of target points depending on the value of *scale* in Equation 2. In our experiments, we efficiently compute the *scale* of each row l at each training step t using the gradient of the loss with respect to the bias of l-th neuron as:

$$\frac{\partial \mathcal{L}}{\partial b_t^l} = \frac{\partial \mathcal{L}}{\partial a_t^l} \cdot \frac{\partial a_t^l}{\partial b_t^l} = \frac{\partial \mathcal{L}}{\partial a_t^l} = \mathit{scale} \ \ . \tag{3}$$

We compute the *scale* of each row for all 1,000 target points, and binarize to demarcate which row stores a copy of the target input:

$$\mathit{B\text{-}scale} = \begin{cases} 0 & \text{if} \quad \mathit{scale} = 0 \\ 1 & \text{otherwise} \ \ , \end{cases} \tag{4}$$

where 0 implies that no information about the target point is stored, while 1 indicates that an exact copy of the target point is stored in that specific row. Figure 5 shows the histogram of the summation of *B-scale* of all rows per target point (i.e., the number of rows that store each target point) over time. Almost[3] all the parameters of models with Sigmoid activations store all the target points, while for ReLU activations, some rows store no information about the target training point. This effect becomes more severe at later steps, where a larger number of rows store no information about the target point for ReLU models. The histogram of

[3] Figure 5 shows that none of superparameters in Sigmoid activated model store two (out of 1000) target points. We hypothesize that this is due to the saturation of Sigmoid for these two target points whose pixel values are mostly zero (visually black).

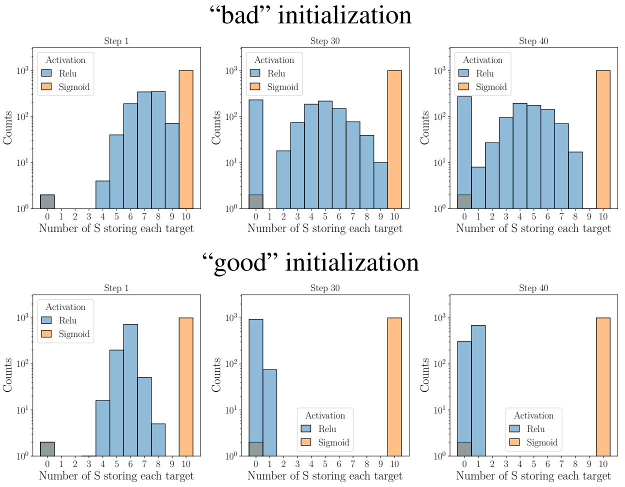

Figure 5: Training dynamics of ReLU activated models versus Sigmoid activated models using different initializations. Each plot shows the histogram of number of rows (S) that store each target point. Not all parameters of ReLU activated models store target points, while all parameters of Sigmoid activated models store information about target points.

the number of rows that store each target point in "good" initializations is more condensed than for "bad" initializations. This offers an intuitive explanation for why reconstruction becomes more difficult on ReLU models with "bad" initializations: the number of rows storing target points varies across points and time in the "bad" initialization. That is, for "bad" initializations the pattern for redundant rows over training can differ wildly for two different target points, making it more difficult for the RecoNN to learn.

5 MNEMONIST: FINDING PARAMETERS THAT STORE TARGET EXAMPLES

5.1 METHODOLOGY

With a view to identifying redundant parameters that are not useful for a reconstruction attack without accessing intermediate updates from the released model, we introduce Mnemonist. Our proposed Mnemonist method is a black-box approach that explains which parameters of the released model store target points. Mnemonist quantifies the contribution of parameters on reconstructing each target point by extending techniques from explainable ML. In particular, we extend the approach of Local Interpretable Model-Agnostic Explanations (LIME [Ribeiro et al., 2016]) to operate in parameter space. Figure 6 shows an overview of Mnemonist, which consists of three sequential phases: 1) Grouping model parameters into sets, which we term *superparameters*; 2) Building different variants of released model based on the presence or absence of each superparameter; 3) Training an interpretable model specifying the importance of each superparameter. Next, we describe each phase of Mnemonist in detail.

Phase 1: Grouping parameters into superparameters. Randomly changing an individual parameter of the model cannot change the output of RecoNN as models often contains a large number of parameters. We propose to group the model parameters into superparameters such that changing individual superparameters significantly changes the quality of the reconstructed target point output by RecoNN if the superparameter stores the target point. For each layer i, we group the parameters into S superparameters $\{\mathbf{w}^{(i,s)}\}_{s=1}^{S}$ based on the destination nodes of this layer, where S is the total number of destination nodes. Each superparameter $\mathbf{w}^{(i,s)}$ represents the s-th row of model parameters of the i-th layer which might store a scaled copy of the input depending on the sign of the signal passed to ReLU activations in the forward pass while training the model (see Section 4).

Phase 2: Building different variants of the released model based on the presence or absence of each superparameter. In order to determine whether a particular superparameter stores the target point, we capture the effect of masking out the superparameter on the RecoNN responses. To do that, we create M perturbed models $\{\mathbf{W}'_m\}_{m=1}^{M}$ by randomly selecting several superparameters identified by 1s in each binary mask $\{\mathbf{b}_m \in \{0,1\}^S\}_{m=1}^{M}$ and replacing the value of the rest of superparameters with values used to initialise parameters. Reverting back values of a superparameter into its initial values removes the effect of all updates done during training, thus removing any information about target points that might have been stored in that superparameter. In particular, the value of each superparameter within $\mathbf{W}'_m$ is set as follows:

$$\mathbf{w}'^{(s)}_{m} = \begin{cases} \mathbf{w}^{(s)} & \text{if} \quad b_m^{(s)} \neq 0 \\ \mathbf{w}^{(s)}_{\text{init}} & \text{otherwise} \quad . \end{cases} \tag{5}$$

Phase 3: Training an interpretable model specifying the importance of each superparameter. We aim to create an interpretable model that can capture and explain the RecoNN responses to the present or absence of each individual superparameter. We consider a 1-dimensional output linear regression model whose coefficient explains the importance of each individual input feature on its 1 dimensional output. We train the linear regression model Regressor : $B \to L$ where the input is a binary vector indicating the presence or absence of each superparameter, and the output is MSE loss between the reconstructed target example and the target example. In particular, we create the input and output of the regression model as follows:

- Output: we query RecoNN to obtain reconstructed images $\{\hat{\mathbf{z}}_m\}_{m=1}^{M}$ on these M perturbed released models.

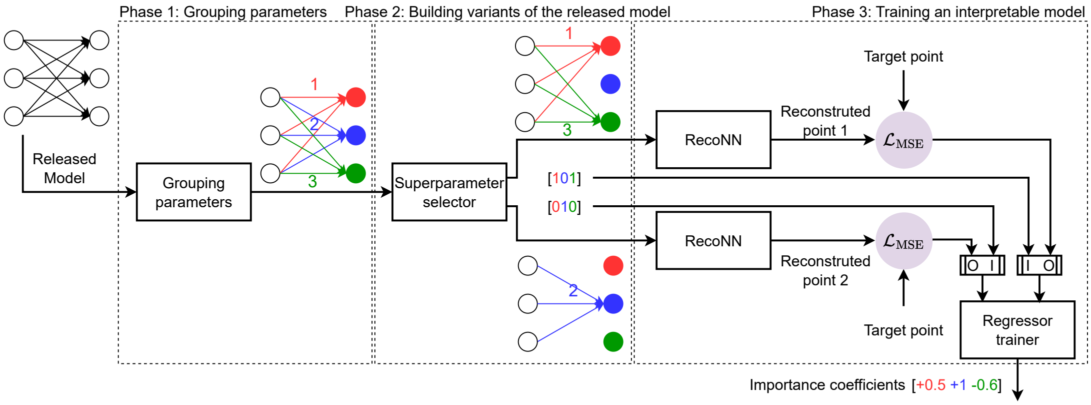
Phase 1: Grouping parameters
Phase 2: Building variants of the released model
Phase 3: Training an interpretable model
Released Model
Grouping parameters
1
2
3
Superparameter selector
1
3
2
[101]
[010]
RecoNN
RecoNN
Target point
Reconstruted point 1
$\mathcal{L}_{\text{MSE}}$
Reconstruted point 2
$\mathcal{L}_{\text{MSE}}$
Target point
O I
I O
Regressor trainer
Importance coefficients [+0.5 +1 -0.6]

Figure 6: An overview of Mnemonist that quantifies the contribution of model parameters on reconstructing a target point.

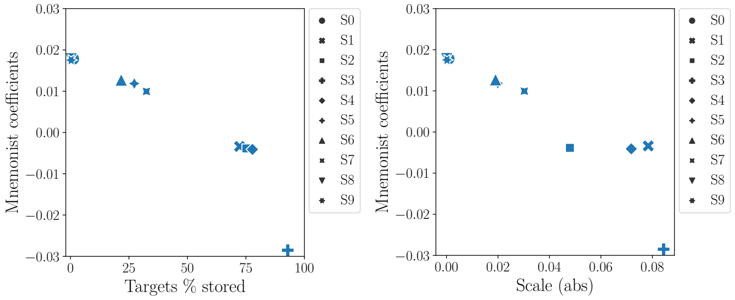

Figure 7: Coefficients of the interpretbale model in Mnemonist as a function of number of target points points stored in each superparameter across 1,000 released models (left) and *scale* of storing target points in each superparameters after 1 step. Mnemonist coefficients are aligned with training dynamics such that it can detect important and redundant superparameters.

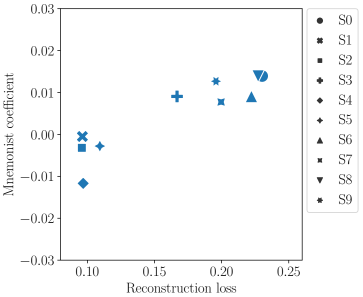

Figure 8: Coefficients of the interpretable model in Mnemonist as a function of reconstructing targets based on each superparameter after 40 steps. Mnemonist can detect important and redundant superparameters.

To create 1 dimensional output for the interpretable linear regression, we compute the MSE between each $\hat{\mathbf{z}}_m$ and the target example $\mathbf{z}$ as $\{l_m = \mathcal{L}_{\text{MSE}}(\hat{\mathbf{z}}_m, \mathbf{z})\}_{m=1}^{M}$.

- Input: binary masks, $\{\mathbf{b}_m \in \{0,1\}^S\}_{m=1}^{M}$, indicating the presence or absence of superparameter.

Once the linear regression model is trained, we have a coefficient per superparameter that identifies the effect of each individual superparameter on the quality of the reconstruction. We interpret the coefficients of the trained linear regression model as follows. The presence of superparameters with negative coefficients can decrease the MSE reconstruction loss, thus improving the quality of the reconstruction. However, the presence of superparameters with positive coefficients can increase the MSE reconstruction loss, thus damaging the quality of the reconstruction. Therefore, we identify superparameters with negative (or small) coefficients as important superparameters for reconstructing target points.

5.2 VALIDATION

We validate the performance of Mnemonist in estimating the importance of parameters. First, we analyze the performance of Mnemonist on released models trained with a single update step in order to evaluate how well Mnemonist detects superparameters with non-zero gradients. Figure 7 illustrates the coefficients of Mnemonist as a function of both *scale* of each superparameter (see Equation 2) and number of target points stored in each superparameter. The smaller the value of the Mnemonist coefficient, the more important the superparameter. As the *scale* (or number of stored target examples) increases, the Mnemonist coefficient decreases. For example, the Mnemonist coefficient of $S3$ that stores more than 90% of target points is (-0.03) while the coefficient of $S6$ that stores less than 25% of target points is $(+0.01)$.

Second, we evaluate the behaviour of Mnemonist on released models trained for more than one step. In Figure 8, we again observe that superparameters with negative coefficients are those superparameters with the best performance when we train RecoNN on each individual superparame-

Algorithm 1: Mnemonist-RecoNN

```
Input: Fixed set $D_-$, $K$ public target examples $\{\bar{z}_k\}_{k=1}^{K}$,
       Shadow model training Algorithm $A(\cdot)$, RecoNN
       training algorithm $B(\cdot)$.
Output: Mnemonist-guided RecoNN.
for all $k \in$ range$(K)$ do
    $\bar{\mathbf{W}}_k \leftarrow A(D_- \cup \{\bar{z}_k\})$    ▷ Train $K$ shadow models
$\phi \leftarrow B(\{\bar{\mathbf{W}}_k, \bar{z}_k\}_{k=1}^{K})$    ▷ Train RecoNN
$I \leftarrow$ Mnemonist$(\phi)$    ▷ Apply Mnemonist to identify the importance of superparameters
for all $k \in$ range$(K)$ do
    $\tilde{\mathbf{W}}_k \leftarrow$ Selector$(\bar{\mathbf{W}}_k, I)$    ▷ Select only important superparameters
$\tilde{\phi} \leftarrow B(\{\tilde{\mathbf{W}}_k, \bar{z}_k\}_{k=1}^{K})$    ▷ Train RecoNN on the selected important superparameters
return $\tilde{\phi}$    ▷ Mnemonist-guided RecoNN
```

Table 3: Mnemonist improves the loss of reconstructing target points from models with ReLU activation function.

Approach	run1	run2	run3	run4
RecoNN	.2040	.2705	.0908	.1315
RecoNN +Mnemonist	**.1738**	**.2385**	**.0730**	**.1158**

ter: the smaller the Mnemonist coefficient, the better the reconstruction loss.

As Mnemonist coefficients are aligned with the training dynamic of released models, they can be used to improve the success of RecoNN by shifting its focus towards important superparameters and ignoring redundant superparameters.

5.3 APPLICATION

We aim to improve the quality of reconstruction of training examples obtained by RecoNN, based on insights provided by Mnemonist regarding where and how target examples are stored in ReLU activated models. To do that, we design our Mnemonist-RecoNN attack in which RecoNN are trained only on those superparameters that have Mnemonist negative coefficients (see Algorithm 1). Table 3 and Figure 9 show the effect of Mnemonist on the reconstruction success of current attack. Results show that Mnemonist can guide and improve the performance of the current RecoNN.

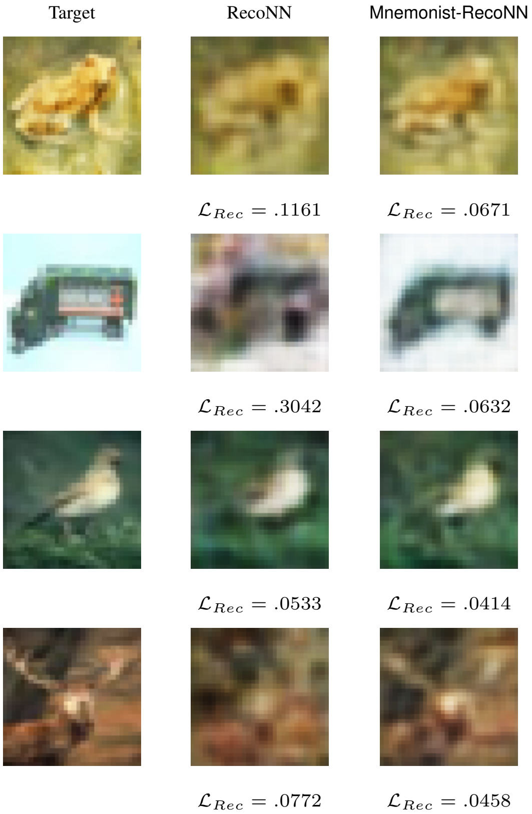
Target
RecoNN
Mnemonist-RecoNN
$\mathcal{L}_{Rec} = .1161$
$\mathcal{L}_{Rec} = .0671$
$\mathcal{L}_{Rec} = .3042$
$\mathcal{L}_{Rec} = .0632$
$\mathcal{L}_{Rec} = .0533$
$\mathcal{L}_{Rec} = .0414$
$\mathcal{L}_{Rec} = .0772$
$\mathcal{L}_{Rec} = .0458$

Figure 9: Target examples and their reconstruction candidates obtained from ReLU activated fully connected models using RecoNN and Mnemonist-RecoNN. Mnemonist helps RecoNN to improve the quality of reconstructions.

6 DISCUSSION AND FUTURE WORK

We provided theoretical and empirical analyses investigating the effect of the type of non-linearity used in the model specification on the quality of reconstructing examples from the model parameters. We proposed a theoretically motivated explanation technique, Mnemonist, to locate model parameters that memorize training examples, thus improving the quality of reconstructions of current attack on small MLPs with ReLU activations. Below, we discuss some limitations of our approach and promising directions for future work.

Model architecture. Fully connected neural networks are the focus of many current training data reconstruction attacks [Fowl et al., 2022, Boenisch et al., 2023, Haim et al., 2022], and yet is it not fully understood what training conditions lead to successful reconstruction attacks in these networks. Our work investigates the necessary properties of model specification in fully connected neural networks that enable better reconstruction attacks. However, an interesting future direction is to extend our proposed explanation technique to other architectures such as Convolutional Neural Networks (CNNs) with ReLU activations. If the adversary has the ability to choose or design the model architecture [Fowl et al., 2022, Boenisch et al., 2023], then Section 4.1 shows that by using a fully connected layer in the first layer, reconstruction attacks become easier. This is because complete copies of targets can be stored in first layer updates (see Equation 2). Indeed, our Mnemonist approach can be used in this setting to help identify the parameters of linear layers that store these examples.

Extending our approach to CNNs in a benign setting. In more benign settings, where the adversary does not have the capability to manipulate or choose the model architecture, to apply our approach to standard CNN architectures, our theoretical analysis will need to be extended to establish how data points are memorised in each convolutional layer. CNNs typically have fully connected layers at the end: the last convolutional layer outputs embeddings which are the input of the first fully connected layer located after this last convolutional layer. Therefore, our approach can extend to CNNs using an embedding reconstruction attack followed by an embedding to data mapping as follows; i) dropping all convolutional layers and perform the attack on the first fully connected layer in which our fully connected based method and analyses can be used; ii) training RecoNN such that it receives the parameters of the first fully connected layer and tries to reconstruct its corresponding embeddings and iii) training a network that maps the embedding to data.

Efficiency. The reconstruction attacks described in this work inherit the computation bottleneck of the attack described in [Balle et al., 2022], as (1) thousands of shadow models need to be trained, and (2) the RecoNN uses the parameters of the shadow model as input, which can become extremely large for large models and high dimensional datasets. Reducing the number of shadow models that need to be trained, or reducing the number of parameters that need to be passed as inputs to the RecoNN can improve the efficiency of the attack. In turn, this will allow us to scale reconstruction attacks to larger datasets and models. For example, our experiments show that we don't need the full set of parameters to perform good reconstruction attacks, which opens the door for future work on identifying and reducing the number of parameters we need to use as inputs.

The quality of the data reconstruction versus privacy-accuracy trade-offs in DPSGD. Papernot et al. [2021] has demonstrated that replacing ReLU activation functions with smooth activation functions can improve the trade-offs between privacy and accuracy of Differentially Private Stochastic Gradient Descent (DPSGD). However, we demonstrate that the informed adversary proposed by Balle et al. [2022] cannot successfully reconstruct training examples from ReLU activated models. This conflicting evidence between the impact of the activation function on the quality of the data reconstruction versus privacy-accuracy trade-offs in DPSGD presents a promising direction for future work.

ACKNOWLEDGMENTS

The authors would like to thank David Stutz for feedback on an earlier version of this manuscript. Adrian Weller acknowledges support from EPSRC grant EP/V056883/1, a Turing AI Fellowship under EP/V025279/1, and the Leverhulme Trust via CFI. Code will be made available.

References

Borja Balle, Giovanni Cherubin, and Jamie Hayes. Reconstructing training data with informed adversaries. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA, May 2022.

Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. When the curious abandon honesty: Federated learning is not private. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroSP)*, Delft, Netherlands, July 2023.

Liam Fowl, Jonas Geiping, Wojtek Czaja, Micah Goldblum, and Tom Goldstein. Robbing the fed: Directly obtaining private data in federated learning with modified models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Online, April 2022.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, Fort Lauderdale, Florida, April 2011.

Chuan Guo, Brian Karrer, Kamalika Chaudhuri, and Laurens van der Maaten. Bounding training data reconstruction in private (deep) learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, Baltimore, Maryland, USA, July 2022.

Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, New Orleans, USA, November 2022.

Matthew Jagielski, Jonathan Ullman, and Alina Oprea. Auditing differentially private machine learning: How private is private SGD? In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, Online, December 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, Lake Tahoe, Nevada, USA, December 2012.

Fred Lu, Joseph Munoz, Maya Fuchs, Tyler LeBlond, Elliott Zaresky-Williams, Edward Raff, Francis Ferraro, and Brian Testa. A general framework for auditing differentially private machine learning. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, New Orleans, USA, November 2022.

Milad Nasr, Shuang Songi, Abhradeep Thakurta, Nicolas Papernot, and Nicholas Carlin. Adversary instantiation:

Lower bounds for differentially private machine learning. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, Online, May 2021.

Nicolas Papernot, Abhradeep Thakurta, Shuang Song, Steve Chien, and Úlfar Erlingsson. Tempered Sigmoid activations for deep learning with differential privacy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Online, February 2021.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD)*, San Francisco, CA, USA, August 2016.

Wenqi Wei, Ling Liu, Margaret Loper, Ka-Ho Chow, Mehmet Emre Gursoy, Stacey Truex, and Yanzhao Wu. A framework for evaluating client privacy leakages in federated learning. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, Online, September 2020.

Yuxin Wen, Jonas Geiping, Liam Fowl, Micah Goldblum, and Tom Goldstein. Fishing for user data in large-batch federated learning via gradient magnification. In *Proceedings of the International Conference on Machine Learning (ICML)*, Baltimore, Maryland, USA, July 2022.

Santiago Zanella-Béguelin, Lukas Wutschitz, Shruti Tople, Ahmed Salem, Victor Rühle, Andrew Paverd, Mohammad Naseri, and Boris Köpf. Bayesian estimation of differential privacy. In *Proceedings of the International Conference on Machine Learning (ICML)*, Hawaii, July 2023.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Toulon, France, April 2017.