Reconstructing Training Data from Trained Neural Networks

Niv Haim*
Weizmann Institute of Science
niv.haim@weizmann.ac.il

Gal Vardi*†
TTI-Chicago and Hebrew University
galvardi@ttic.edu

Gilad Yehudai*
Weizmann Institute of Science
gilad.yehudai@weizmann.ac.il

Ohad Shamir
Weizmann Institute of Science
ohad.shamir@weizmann.ac.il

Michal Irani
Weizmann Institute of Science
michal.irani@weizmann.ac.il

Project page: https://giladude1.github.io/reconstruction

Abstract

Understanding to what extent neural networks memorize training data is an intriguing question with practical and theoretical implications. In this paper we show that in some cases a significant fraction of the training data can in fact be reconstructed from the parameters of a trained neural network classifier. We propose a novel reconstruction scheme that stems from recent theoretical results about the implicit bias in training neural networks with gradient-based methods. To the best of our knowledge, our results are the first to show that reconstructing a large portion of the actual training samples from a trained neural network classifier is generally possible. This has negative implications on privacy, as it can be used as an attack for revealing sensitive training data. We demonstrate our method for binary MLP classifiers on a few standard computer vision datasets.

1 Introduction

It is commonly believed that neural networks memorize the training data, even when they are able to generalize well to unseen test data (e.g., [Zhang et al., 2021, Feldman, 2020]). Exploring this memorization phenomenon is of great importance both practically and theoretically. Indeed, it has implications on our understanding of generalization in deep learning, on the hidden representations learnt by neural networks, and on the extent to which they are vulnerable to privacy attacks.

A fundamental question for understanding memorization is:

> *Are the specific training samples encoded in the parameters of a trained classifier? Can they be recovered from the network parameters?*

In this work, we study this question, and devise a novel scheme which allows us to reconstruct a significant portion of the training data from the parameters of a trained neural network alone, without having any additional information on the data. Thus, we provide a proof-of-concept that the learning

*Equal contribution, alphabetically ordered

†Work done while the author was at the Weizmann Institute of Science

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

(a) Top 24 images reconstructed from a binary classifier trained on 50 CIFAR10 images

(b) Their corresponding nearest neighbours from the training-set of the model

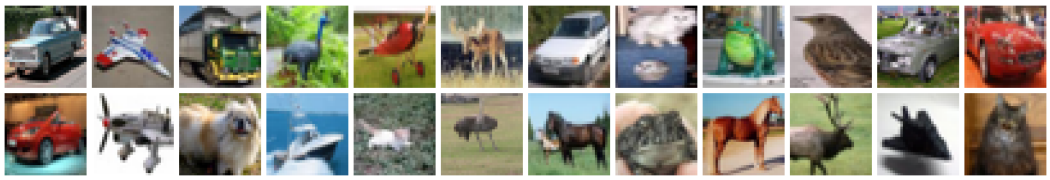


Figure 1: Reconstruction of training images from a pretrained binary classifier, trained on 50 CIFAR10 images. The two classes are "animals" and "vehicles". We calculate the nearest neighbor using the SSIM metric.

process can sometimes be reversed: That is, instead of learning a model given a training dataset, it is possible to find the training data given a trained model. In Figure 1 we show how our approach reconstructs images from the CIFAR10 dataset, given a simple trained binary classifier.

Many works try to "crack" neural networks by analyzing and visualizing either their learnt parameters or representations [Erhan et al., 2009, Mahendran and Vedaldi, 2015, Olah et al., 2017, 2020]. This is usually done by "inverting" the model, namely finding inputs that are strongly correlated with the model's activations [Mordvintsev et al., 2015, Yin et al., 2020, Fredrikson et al., 2015]. Unsurprisingly, the results are semantically correlated with the training dataset. However, one rarely sees an exact version of a training sample.

Our results have potential negative implications on privacy in deep learning. Our scheme can be viewed as a *training-data reconstruction attack*, since an adversary might recover sensitive training data. For example, if a medical device includes a model trained on sensitive medical records, an adversary might reconstruct this data and thus violate the privacy of the patients. Privacy attacks in deep learning have been widely studied in recent years (cf. Liu et al. [2021]), but as far as we are aware, the known attacks cannot reconstruct portions of the training data from a trained model.

Our approach relies on theoretical results about the implicit bias in training neural networks with gradient-based methods. The implicit bias has been studied extensively in recent years with the motivation of explaining generalization in deep learning (see Section 2). We use results by Lyu and Li [2019], Ji and Telgarsky [2020], which establish that, under some technical assumptions, if we train a neural network with the binary cross entropy loss, its parameters will converge to a stationary point of a certain margin-maximization problem. This result implies that the parameters of the trained network satisfy a set of equations w.r.t. the training dataset. In our approach, given a trained network, we find a dataset that solves this set of equations w.r.t. the trained parameters.

Our Contributions We show that large portions of the training samples are encoded in the parameters of a trained classifier. We also provide a practical scheme to decode the training samples, without any assumptions on the data. As far as we know, this is the first work that shows that reconstruction of actual training samples from a trained neural network classifier is possible.

2 Related Work

Understanding and Visualizing what is learnt by Neural Networks. The most common approach for analysing what is learnt by a neural network is by searching inputs that maximize the class output or the activations of neurons in intermediate layers [Erhan et al., 2009, Olah et al., 2020]. Oftentimes this is done via optimization with respect to the model input. Optimizing without any prior on the input usually results in noise inputs. Therefore, most approaches incorporate priors such as smoothness regularization or the use of pre-trained image generators [Mahendran and Vedaldi, 2015, Yosinski et al., 2015, Mordvintsev et al., 2015, Nguyen et al., 2016a,b, 2017] (see Olah et al. [2017] for a comprehensive summary). Optimization w.r.t. the input may also result in adversarial examples

[Szegedy et al., 2013, Goodfellow et al., 2014]. Recently, [Tsipras et al., 2018, Engstrom et al., 2019] showed that classifiers trained to be robust to adversarial examples tend to learn representations that are more aligned with human vision. This was later utilized by [Santurkar et al., 2019, Mejia et al., 2019] to generate class-conditional images from a trained classifier. While all those approaches indicate that, unsurprisingly, the learnt representations are strongly correlated with the datasets on which the model was trained, none of them demonstrate the reconstruction of exact training samples from the trained models.

Privacy Attacks in Deep Learning. Many methods deal with extracting sensitive information from trained models. Perhaps the closest to our approach is *model-inversion* that aims to reconstruct class representatives from the training data of a trained model [Fredrikson et al., 2015, He et al., 2019, Yang et al., 2019, Yin et al., 2020]. It is important to note that the reconstructed images, albeit semantically similar to some input images, are still not actual samples from the training set. Carlini et al. [2021, 2019] demonstrated reconstruction of training data from generative language models. By completing sentences, they reveal sensitive information from the training data. We note that this approach is specific to generative language models, while our approach considers classifiers and is less data specific. *Membership-inference* attacks [Shokri et al., 2017] aim to determine whether a given data point was used to train the model or not. For these methods to work, the adversary must be able to guess a specific input, whereas our approach does not assume such ability. Lastly, avoiding leakage of sensitive information on the training dataset is the motivation behind *differential privacy* in machine learning, which has been extensively studied [Abadi et al., 2016, Dwork et al., 2006, Chaudhuri et al., 2011]. For an elaborated discussion on the relation of these approaches to ours see Appendix A.

Implicit Bias. In overparameterized neural networks one might expect overfitting to occur, but it seems that gradient-based methods are biased towards networks that generalize well [Zhang et al., 2021, Neyshabur et al., 2017]. Mathematically characterizing this implicit bias is a major problem in the theory of deep learning. Our approach is based on a characterization of the implicit bias of gradient flow in homogeneous neural networks due to Lyu and Li [2019] and Ji and Telgarsky [2020] (see Section 3 for details). The implicit bias of gradient-based methods in neural networks was extensively studied in recent years both for classification tasks (e.g., Soudry et al. [2018], Gunasekar et al. [2018c], Ji and Telgarsky [2018], Nacson et al. [2019], Vardi et al. [2021], Chizat and Bach [2020], Gunasekar et al. [2018a], Moroshko et al. [2020]) and regression tasks (e.g., Gunasekar et al. [2018b], Arora et al. [2019], Azulay et al. [2021], Yun et al. [2020], Woodworth et al. [2020], Razin and Cohen [2020], Li et al. [2020], Vardi and Shamir [2021], Timor et al. [2022]). See Vardi [2022] for a survey.

3 Background and Reconstruction Scheme

In this section we present our training data reconstruction scheme, as well as provide a brief overview on the theoretical results about implicit bias, which motivate our approach.

3.1 On the Implicit Bias of Neural Networks

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^d \times \{-1, 1\}$ be a binary classification training dataset. Let $\Phi(\boldsymbol{\theta}; \cdot) : \mathbb{R}^d \to \mathbb{R}$ be a neural network parameterized by $\boldsymbol{\theta} \in \mathbb{R}^p$. For a loss function $\ell : \mathbb{R} \to \mathbb{R}$ the empirical loss of $\Phi(\boldsymbol{\theta}; \cdot)$ on the dataset S is $\mathcal{L}(\boldsymbol{\theta}) := \sum_{i=1}^n \ell(y_i \Phi(\boldsymbol{\theta}; \mathbf{x}_i))$. We focus on the *logistic loss* (a.k.a. *binary cross entropy*), namely, $\ell(q) = \log(1 + e^{-q})$.

Our approach is based on Theorem 3.1 below, which holds for *gradient flow* (i.e., gradient descent with an infinitesimally small step size). Before stating the theorem, we need the following definitions: (1) We say that gradient flow *converges in direction* to $\tilde{\boldsymbol{\theta}}$ if $\lim_{t \to \infty} \frac{\boldsymbol{\theta}(t)}{\|\boldsymbol{\theta}(t)\|} = \frac{\tilde{\boldsymbol{\theta}}}{\|\tilde{\boldsymbol{\theta}}\|}$, where $\boldsymbol{\theta}(t)$ is the parameter vector at time t; (2) We say that a network Φ is *homogeneous* w.r.t. the parameters $\boldsymbol{\theta}$ if there exists $L > 0$ such that for every $\alpha > 0$ and $\boldsymbol{\theta}, \mathbf{x}$ we have $\Phi(\alpha\boldsymbol{\theta}; \mathbf{x}) = \alpha^L \Phi(\boldsymbol{\theta}; \mathbf{x})$. Thus, scaling the parameters by any factor $\alpha > 0$ scales the outputs by α^L. We note that essentially any fully-connected or convolutional neural network with ReLU activations is homogeneous w.r.t. the parameters $\boldsymbol{\theta}$ if it does not have any skip-connections (i.e., residual connections) or bias terms, except possibly for the first layer.

Neural Networks: In deep learning, gradient descent has a bias towards low-rank or low-complexity solutions, which helps in generalization. This is why neural networks can perform well despite being heavily overparameterized.

Effect of Step Size and Learning Rate: The choice of step size affects the implicit bias. For instance, small learning rates tend to converge to solutions with a smaller norm, while large learning rates can push the model towards flatter minima.

Key Observation:
Training:
(x is fixed) theta converges to -Y
Reconstruction:
(theta fixed) Solve Eq.(1-4) for x

Theorem 3.1 (Paraphrased from Lyu and Li [2019], Ji and Telgarsky [2020]) *Let $\Phi(\boldsymbol{\theta};\cdot)$ be a homogeneous ReLU neural network. Consider minimizing the logistic loss over a binary classification dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ using gradient flow. Assume that there exists time t_0 such that $\mathcal{L}(\boldsymbol{\theta}(t_0)) < 1$[‡]. Then, gradient flow converges in direction to a first order stationary point (KKT point) of the following maximum-margin problem:*

$$\min_{\boldsymbol{\theta}'} \frac{1}{2}\left\|\boldsymbol{\theta}'\right\|^2 \quad \textit{s.t.} \quad \forall i \in [n] \;\; y_i \Phi(\boldsymbol{\theta}'; \mathbf{x}_i) \geq 1 \,. \tag{1}$$

Moreover, $\mathcal{L}(\boldsymbol{\theta}(t)) \to 0$ as $t \to \infty$.

The above theorem guarantees directional convergence to a first order stationary point (of the optimization problem (1)), which is also called *Karush–Kuhn–Tucker point*, or *KKT point* for short. The KKT approach allows inequality constraints, and is a generalization of the method of *Lagrange multipliers*, which allows only equality constraints.

The great virtue of Theorem 3.1 is that it characterizes the *implicit bias* of gradient flow with the logistic loss for homogeneous networks. Namely, even though there are many possible directions of $\frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|}$ that classify the dataset correctly, gradient flow converges only to directions that are KKT points of Problem (1). In particular, if the trajectory $\boldsymbol{\theta}(t)$ of gradient flow under the regime of Theorem 3.1 converges in direction to a KKT point $\tilde{\boldsymbol{\theta}}$, then we have the following: There exist $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ such that

$$\tilde{\boldsymbol{\theta}} = \sum_{i=1}^n \lambda_i y_i \nabla_{\boldsymbol{\theta}} \Phi(\tilde{\boldsymbol{\theta}}; \mathbf{x}_i) \qquad \text{(stationarity)} \tag{2}$$

$$\forall i \in [n], \;\; y_i \Phi(\tilde{\boldsymbol{\theta}}; \mathbf{x}_i) \geq 1 \qquad \text{(primal feasibility)} \tag{3}$$

$$\lambda_1, \ldots, \lambda_n \geq 0 \qquad \text{(dual feasibility)} \tag{4}$$

$$\forall i \in [n], \;\; \lambda_i = 0 \text{ if } y_i \Phi(\tilde{\boldsymbol{\theta}}; \mathbf{x}_i) \neq 1 \qquad \text{(complementary slackness)} \tag{5}$$

Our main insight is based on Eq. (2), which implies that the parameters $\tilde{\boldsymbol{\theta}}$ are a linear combinations of the derivatives of the network at the training data points. We say that a data point $\mathbf{x}_i$ is *on the margin* if $y_i \Phi(\tilde{\boldsymbol{\theta}}; \mathbf{x}_i) = 1$ (i.e. $|\Phi(\tilde{\boldsymbol{\theta}}; \mathbf{x}_i)| = 1$) . Note that Eq. (5) implies that only samples which are on the margin affect Eq. (2), since samples not on the margin have a coefficient $\lambda_i = 0$.

3.2 Dataset Reconstruction

Suppose we are given a trained neural network with parameters $\boldsymbol{\theta}$, and our goal is to reconstruct the dataset that the network was trained on. Although Theorem 3.1 holds asymptotically as the time t tends to infinity, it suggests that also after training for a finite number of iterations the parameters of the network might approximately satisfy Eq. (2), and the coefficients λ_i satisfy Eq. (4). Since n is unknown (and so is the number of samples on the margin) we set $m \geq 2n$ which represents the number of samples we want to reconstruct (thus, we only need to upper bound n), and fix $y_i = 1$ for $i = 1, \ldots, m/2$ and $y_i = -1$ for $i = m/2 + 1, \ldots, m$. We define the following losses:

$$L_{\text{stationary}}(\mathbf{x}_1, \ldots, \mathbf{x}_m, \lambda_1, \ldots, \lambda_m) = \left\| \boldsymbol{\theta} - \sum_{i=1}^m \lambda_i y_i \nabla_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta}; \mathbf{x}_i) \right\|_2^2 \tag{6}$$

$$L_\lambda(\lambda_1, \ldots, \lambda_m) = \sum_{i=1}^m \max\{-\lambda_i, 0\} \tag{7}$$

Note that the unknown parameters are the $\mathbf{x}_i$'s and λ_i's, and that $\boldsymbol{\theta}$ and the y_i's are given. The loss $L_{\text{stationary}}$ represents the stationarity condition that the parameters of the network satisfy, and L_λ represents the dual feasibility condition. We additionally define L_{prior} which represents some prior knowledge we might have about the dataset. For example, if we know that the dataset contains images, prior knowledge would be that each input coordinate (i.e. each pixel) is between 0 and 1. Given no prior knowledge on the data, we can define $L_{\text{prior}} \equiv 0$. Finally, we define the reconstruction loss as:

$$L_{\text{reconstruct}}(\{\mathbf{x}_i\}_{i=1}^m, \{\lambda_i\}_{i=1}^m) = \alpha_1 L_{\text{stationary}} + \alpha_2 L_\lambda + \alpha_3 L_{\text{prior}} \tag{8}$$

[‡]This ensures that $\ell(y_i \Phi(\boldsymbol{\theta}(t_0); \mathbf{x}_i)) < 1$ for all i, i.e. at some time Φ classifies every sample correctly.

where $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{R}$ are tunable hyperparameters of the different losses. To reconstruct the dataset, we can use any nonconvex optimization method (e.g. SGD) to find the $\mathbf{x}_1, \ldots, \mathbf{x}_m, \lambda_1, \ldots, \lambda_m$ which minimize Eq. (8). We note that the λ_i's are not part of the training data, but finding them is necessary in order to solve this optimization problem. Finally, we emphasize that there are many other possible options to formulate the KKT conditions Eq. (2)-(5) as an unconstrained optimization problem. However, this simple choice seemed to work quite well in practice.

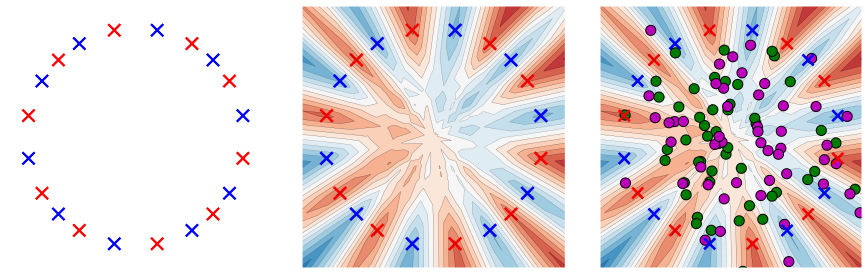
We note that if there exist $\{\mathbf{x}_i\}_{i=1}^n$ and $\{\lambda_i\}_{i=1}^n$ which satisfy the KKT conditions, then there are $\{\mathbf{x}_i\}_{i=1}^m$ and $\{\lambda_i\}_{i=1}^m$ which achieve zero loss in Eq. (8). Indeed, such a solution can be obtained by adding to $\{\mathbf{x}_i\}_{i=1}^n$ additional points $\mathbf{x}_j$ with $\lambda_j = 0$, or by duplicating some points in $\{\mathbf{x}_i\}_{i=1}^n$ and modifying the λ's accordingly. Also, note that since we choose $m \geq 2n$, then we set at least n labels y_i to 1 and at least n labels to -1. Hence, there is a solution to Eq. (8) even though we do not know the real distribution of labels in the actual training data.

We cannot simply use Eq. (3) and (5) in our reconstruction scheme, because they contain the constant "1" which corresponds to the margin (i.e., $\min_i |\Phi(\tilde{\boldsymbol{\theta}}; \mathbf{x}_i)|$). Namely, we only converge *in direction* to a point $\tilde{\boldsymbol{\theta}}$ that attains margin 1, but in practice we approach some point $\boldsymbol{\theta}$ which attains an unknown margin γ (i.e., $\min_i |\Phi(\boldsymbol{\theta}; \mathbf{x}_i)| = \gamma$), and we do not know in advance how to normalize it to attain a margin of exactly 1. On the other hand, Eq. (2) and (4) hold not only for $\tilde{\boldsymbol{\theta}}$ but also for any $\boldsymbol{\theta}$ that points at the direction of $\tilde{\boldsymbol{\theta}}$, and therefore in our loss in Eq. (8) we rely only on these conditions.

Intuitively, a reason to believe that there is enough information in Eq. (2) to reconstruct the data, is the following observation: Eq. (2) represents a set of p equations with $O(nd)$ unknown variables, where p is the number of parameters in the network. In practice, neural networks are often highly overparameterized (i.e., $p > nd$), suggesting more equations than variables.

Finally, since by Eq. (5) we have $\lambda_i = 0$ for every $\mathbf{x}_i$ that is not on the margin, then Eq. (2) implies that $\tilde{\boldsymbol{\theta}}$ is determined only by the gradients w.r.t. the data points that are on the margin. Hence, we can only expect to reconstruct training samples that are on the margin (see also Subsection 5.3).

4 A Simple Experiment in Two Dimensions



(a) Training Set (b) Model Landscape (c) Initialization

(d) Reconstruction (e) Removing Small λ's (f) Removing Duplicates

Figure 2: Exemplifying our reconstruction scheme on a simple 2D dataset (see text for explanation).

In this section we exemplify our dataset reconstruction scheme on a toy example of 2-dimensional data, i.e. we consider $(\mathbf{x}, y) \in \mathbb{R}^2 \times \{\pm 1\}$. We set $n = 20$ training samples on the unit circle, with alternating labels. For a visualization of the dataset see Figure 2a, blue and red "$\times$" represent the two classes. We trained a 3-layer model with 1000 neurons in each layer on this dataset. The model learns

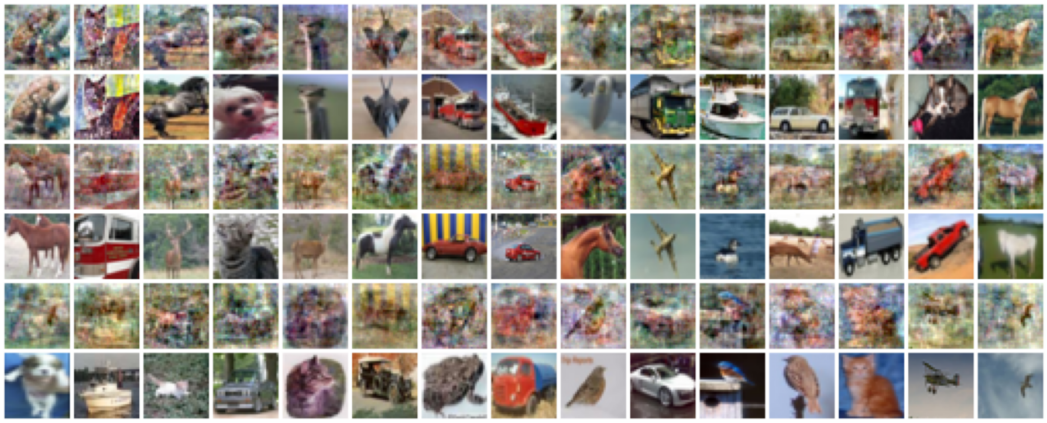
to correctly classify the training set. In Figure 2b, we visualize the output of the model as a function of its input. Blue and red regions correspond to smaller and larger outputs of the model, respectively.

We now demonstrate our reconstruction scheme. We first randomly initialize $m = 100$ points in $\mathbb{R}^2$, and assign 50 points to each class. This is depicted in Figure 2c, where green points correspond to the blue class, and magenta points correspond to the red class. Next, we optimize the loss in Eq. (8), with $L_{\text{prior}} \equiv 0$. The results of our reconstruction scheme are in Figure 2d. Note that our approach reconstructed all the input samples, up to some noise.

To further improve our reconstruction results, we remove some of the extra points which did not converge to a training sample. In Figure 2e we removed points $\mathbf{x}_i$ with corresponding $\lambda_i < 5$. According to Eq. (2), points with $\lambda_i = 0$ should not affect the parameters, hence their corresponding $\mathbf{x}_i$ can take any value. In practice, it is sufficient to remove points with a small enough corresponding λ_i. Finally, to remove duplicates, we greedily remove points which are very close to other points. That is, we randomly order the points, and iteratively remove points that are at distance < 0.03 from another point. The final reconstruction result is depicted in Figure 2f.

5 Results

Top 45 images reconstructed from a model trained on CIFAR10 (rows $1, 3, 5$), and their corresponding nearest-neighbors from the training-set of the model (rows $2, 4, 6$)



Top 45 images reconstructed from a model trained on MNIST (rows $1, 3, 5$), and their corresponding nearest-neighbors from the training-set of the model (rows $2, 4, 6$)

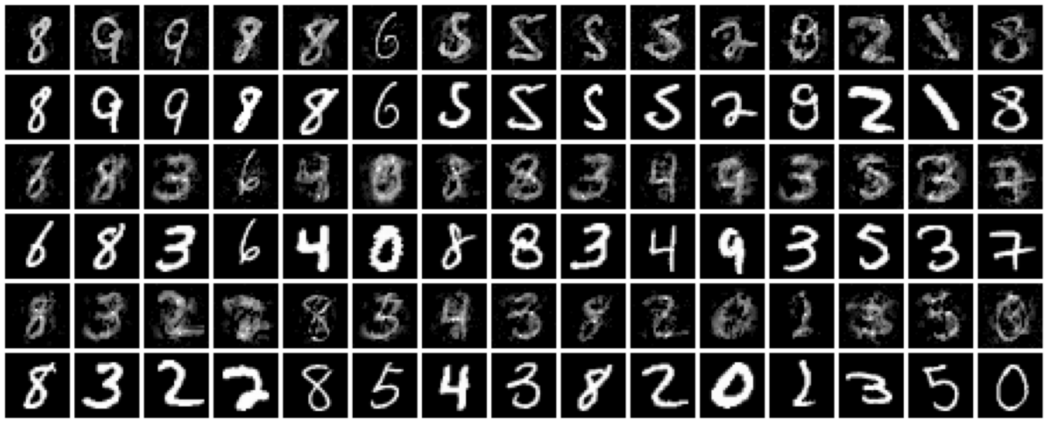


Figure 3: Reconstructing training samples from two binary classifiers – one trained on 500 images with labels animals/vehicles (CIFAR), and the other trained on 500 odd/even digit images (MNIST). Train errors are zero, test accuracies are $88.0\%/77.6\%$ for MNIST/CIFAR

5.1 Experimental Setup

Datasets. We conduct experiments on binary classification tasks where images are taken from the MNIST [LeCun et al., 2010] and CIFAR10 [Krizhevsky et al., 2009] datasets and the labels are set to odd vs. even digits (MNIST), and vehicles vs. animals[§] (CIFAR10). We make sure that the class distribution in the training and test sets is balanced, and normalize the train and test sets by reducing the mean of the training set from both.

Training. We consider MLP architectures. Unless stated otherwise, our models comprise of three fully-connected layers with dimensions d-1000-1000-1 (where d is the dimension of the input) with ReLU activations. Biases are set to zero except for the first layer, to line up with the theoretical assumption of homogeneous models in Section 3. The parameters are initialized using standard Kaiming He initialization [He et al., 2015] except for the weights of the first layer that are initialized to a Gaussian distribution with standard deviation 10^{-4} (see discussion in Subsection 5.2). We train our models using full batch gradient descent for 10^6 epochs with a learning rate of 0.01. All models achieve zero training error (i.e., all the train samples are labeled correctly), and a training loss $< 10^{-6}$. To compute the test accuracy, we use the original test sets of MNIST/CIFAR10 with 10000/8000 images respectively, and labeled accordingly.

5.2 Training Set Reconstruction

We minimize the loss defined in Eq. (8) with $\alpha_1 = 1,\ \alpha_2 = 5,\ \alpha_3 = 1$. We initialize $\mathbf{x}_i \sim \mathcal{N}(0, \sigma_x I)$, where σ_x is a hyperparameter, and $\lambda_i \sim \mathcal{U}[0, 1]$. We set the number of reconstructed samples to $m = 2n$ (where n is the size of the original training set). Note that our loss contains the derivative of ReLU Eq. (6). This derivative is a step function, containing only flat regions which are hard to optimize. We replace the derivative of the ReLU layer (backward function) with a sigmoid, which is the derivative of softplus (a smooth version of ReLU). We use the fact that our inputs are images to penalize values outside the range $[-1, 1]$. To this end we set $L_{\text{prior}}(z) = \max\{z - 1, 0\} + \max\{-z - 1, 0\}$ for each pixel z, and average over all dimensions (pixels) in $\mathbf{x}_i$. We optimize our loss for $100,000$ iterations using an SGD optimizer with momentum 0.9. We conduct a total of 100 runs using a random grid search on the hyperparameters (e.g. learning rate, σ_x. See Appendix B for full details). This results in $100m$ "reconstructed" inputs.

While some $\mathbf{x}_i$ end up converging to a training sample, some end as noise (similar phenomenon can be observed in 2D in Figure 2d). To identify the reconstructions that are most similar to a training image we use the SSIM metric [Wang et al., 2004].

In Figure 3 we show the best reconstruction results (in terms of SSIM) for models trained on n=500 samples from MNIST/CIFAR10 datasets (with test accuracy $88.0\%/77.6\%$ resp.). Note that the reconstructed images are very similar to the real input data, although a bit noisy. The source of this noise is not entirely clear. Possible reasons may be the complexity of the optimization problem, or the possibility that the trained model has not fully converged to the KKT point of Problem (1).

We observed that small initializations significantly improve the quality of the reconstructed samples. We conjecture that small initialization causes faster convergence to the direction of the KKT point. This is also theoretically implied in Moroshko et al. [2020] (for certain linear models). Similarly, training for more epochs also improves the quality of the reconstruction. In Appendix C we show results for reconstructions from networks trained with standard initialization or trained for much fewer epochs. During the training phase, we used full batch gradient descent, to remain as much aligned to the theoretical setting. In Appendix C we show that our approach can reconstruct training data also from models trained with mini-batch SGD.

5.3 Practice vs. Theory

In this section we analyze some relations between our experimental results to the theory laid down in Section 3. Given a trained model and its reconstructed samples, we match each training sample to its best reconstruction (in terms of SSIM score). We then plot this SSIM score against $\Phi(\boldsymbol{\theta}; \mathbf{x})$ (the value of the model's output on this training sample) – for all training samples. In Figure 4 each cell shows such plot for a given model. The top row shows models trained on the same architecture with

[§]Automobile, Truck, Airplane, Ship vs. Bird, Horse, Cat, Dog, Deer, Frog.

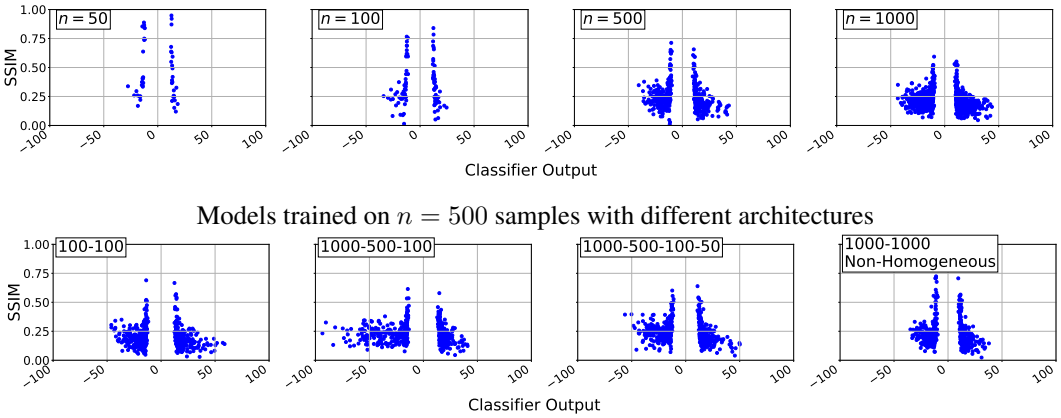


Figure 4: Each point represents a training sample. The y-axis is the highest SSIM score achieved by a reconstruction of this sample, the x-axis is the output of the model. **Top:** The effect of training the same model on different number of training samples (n). **Bottom:** The effect of training models with different architectures (on $n = 500$ training samples). The right-most plot shows a 3-layer non-homogeneous MLP (with bias terms in all hidden layers). See discussion in Section 5.3.

different number of training samples (n), where in the bottom row we show the results for models trained on $n = 500$ training samples, with different architectures (all results are on CIFAR10).

Recall that we do not expect to reconstruct samples that are far from the margin (Subsection 3.2). It is evident from Figure 4 that good reconstructions (e.g., SSIM> 0.4) are obtained for samples that lie on the margin, as expected from theory. The plots indicate that increasing training size makes reconstruction more difficult. Lastly, as seen from the rightmost plot in the bottom row, we manage to get high-quality reconstructions from a non-homogeneous model (trained with biases in all hidden layers). This indicates that our approach may work beyond the theoretical limitations of Theorem 3.1.

5.4 Comparison to other Reconstruction Schemes

Model Inversion. Given a trained model $\Phi(\boldsymbol{\theta}; \cdot)$, we search for $\mathbf{x}$ which maximizes or minimizes $\Phi(\boldsymbol{\theta}; \mathbf{x})$, corresponding to positive or negative labels. We initialize $\mathbf{x} \sim \mathcal{N}(0, \sigma I)$ for several values of σ and optimize w.r.t. the model output (see Appendix B for the choice of hyperparameters). In Figure 5a (left) it is apparent that in our two-dimensional experiment, model inversion successfully reconstructed 7 training samples, which indeed lie on a local minimum or maximum. However, note that our scheme reconstructs all 20 samples (Figure 2). In high dimensions, namely, in MNIST and CIFAR, while our scheme can reconstruct a large portion of the training set (Figure 3a), model inversion converges to noisy/blurry class representatives that correspond to high/low output values (Figure 5a, right). Such results are typical with model inversion since not all class members from the training set are visually similar (see discussions in Shokri et al. [2017], Melis et al. [2019]).

Weights Visualization. The weights of the first fully-connected layer have the same dimension as that of the input. One may wonder whether training samples are directly encoded there. In Figure 5b we show the weights that are most similar (SSIM) to a training sample, or all of them in the 2D case. As seen in the 2D case, most weights are in the general direction of a training sample, however the scale is unknown without prior knowledge on the data. For images (MNIST/CIFAR10), not more than 3 or 4 of the weights have resemblance to training samples, while our scheme manages to reconstruct dozens of samples. See Appendix B and C for details and all 1000 weights of the models.

6 Discussion and Conclusion

Even though our results are shown for relatively small-scale models, they are the first to show that the parameters of trained networks may contain enough information to fully reconstruct training samples,

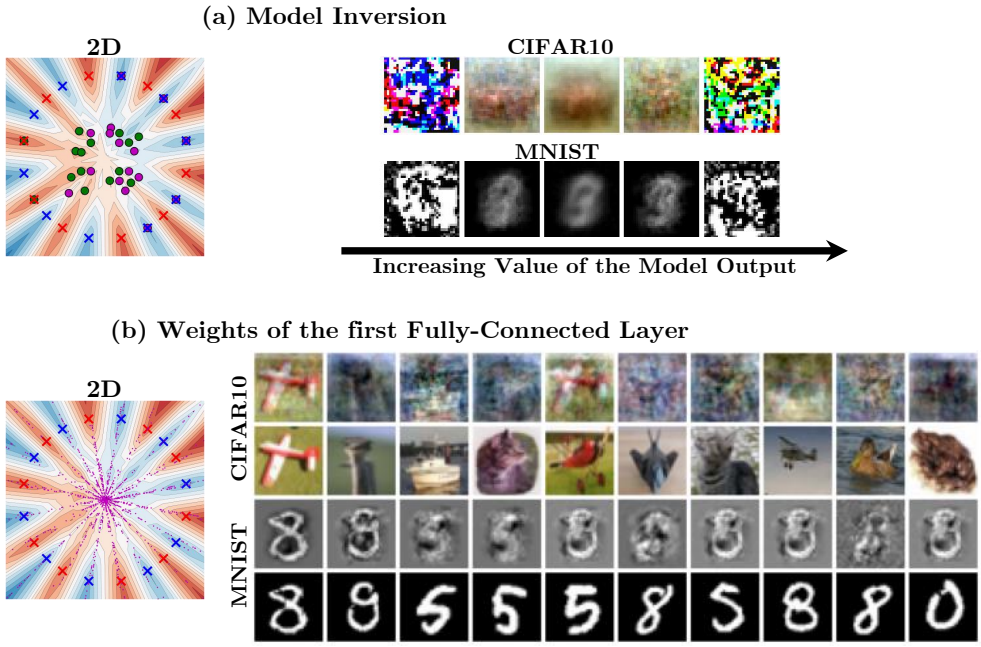


Figure 5: Comparison to other reconstruction schemes. **Top:** Model inversion on the 2D experiment (left), on CIFAR10 (top right) and MNIST (bottom right). The CIFAR and MNIST images are ordered by the value of their output from left (smallest) to right (largest). **Bottom:** Weights of the first (fully-connected) layer for the 2D experiment (left), CIFAR10 (top right) and MNIST (bottom right). The weights for the 2D experiment are the small purple dots. For the CIFAR and MNIST experiments we show the 10 weights with the highest SSIM score.

and the first to *reconstruct a substantial amount of training samples*. Moreover, the theoretical basis of the implicit bias in neural networks provides an analytic explanation to this phenomenon.

Solving our optimization problem for convolutional neural networks turned out to be more challenging and is therefore a subject of future research. We note that the theoretical results that we rely on (i.e., Theorem 3.1) also covers convolutional neural networks. We believe that the homogeneity restriction might be relaxed, and showed reconstructions also from a non-homogeneous model (Figure 4, bottom-rightmost). We also believe that our method may be extended to multi-class classifiers using an extension of Theorem 3.1. Finally, showing reconstructions on larger models and datasets, or on tabular or textual data are interesting future directions.

On the theoretical side, it is not entirely clear why our optimization problem in Eq. (8) converges to actual training samples, even though there is no guarantee that the solution is unique, especially when using no prior (other than simple bounding to $[-1, 1]$). As a final note, our work brings up the question: *are samples on margin the only ones that can be recovered from a trained classifier?* or there exist better reconstruction schemes to reconstruct even more training samples from a trained neural network.

Acknowledgements

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 788535), and ERC grant 754705, and from the D. Dan and Betty Kahn Foundation, and was supported by the Carolito Stiftung.

References

M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

S. Arora, N. Cohen, W. Hu, and Y. Luo. Implicit regularization in deep matrix factorization. In *Advances in Neural Information Processing Systems*, pages 7413–7424, 2019.

S. Azulay, E. Moroshko, M. S. Nacson, B. Woodworth, N. Srebro, A. Globerson, and D. Soudry. On the implicit bias of initialization shape: Beyond infinitesimal mirror descent. In *International Conference on Machine Learning*, pages 468–477, 2021.

B. Balle, G. Cherubin, and J. Hayes. Reconstructing training data with informed adversaries. *arXiv preprint arXiv:2201.04845*, 2022.

L. Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

G. Brown, M. Bun, V. Feldman, A. Smith, and K. Talwar. When is memorization of irrelevant training data necessary for high-accuracy learning? In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 123–132, 2021.

N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 267–284, 2019.

N. Carlini, S. Deng, S. Garg, S. Jha, S. Mahloujifar, M. Mahmoody, S. Song, A. Thakurta, and F. Tramer. Is private learning possible with instance encoding? *arXiv preprint arXiv:2011.05315*, 2020a.

N. Carlini, M. Jagielski, and I. Mironov. Cryptanalytic extraction of neural network models. In *Annual International Cryptology Conference*, pages 189–218. Springer, 2020b.

N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021.

K. Chaudhuri, C. Monteleoni, and A. D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(3), 2011.

S. Chen, A. Klivans, and R. Meka. Efficiently learning one hidden layer relu networks from queries. *Advances in Neural Information Processing Systems*, 34, 2021.

L. Chizat and F. Bach. Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss. In *Conference on Learning Theory*, pages 1305–1338. PMLR, 2020.

C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, B. Tran, and A. Madry. Adversarial robustness as a prior for learned representations. *arXiv preprint arXiv:1906.00945*, 2019.

D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

V. Feldman. Does learning require memorization? a short tale about a long tail. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 954–959, 2020.

M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1322–1333, 2015.

I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

S. Gunasekar, J. Lee, D. Soudry, and N. Srebro. Characterizing implicit bias in terms of optimization geometry. In *International Conference on Machine Learning*, pages 1832–1841. PMLR, 2018a.

S. Gunasekar, J. Lee, D. Soudry, and N. Srebro. Implicit bias of gradient descent on linear convolutional networks. *arXiv preprint arXiv:1806.00468*, 2018b.

S. Gunasekar, J. D. Lee, D. Soudry, and N. Srebro. Implicit bias of gradient descent on linear convolutional networks. In *Advances in Neural Information Processing Systems*, pages 9461–9471, 2018c.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Z. He, T. Zhang, and R. B. Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 148–162, 2019.

B. Hitaj, G. Ateniese, and F. Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 603–618, 2017.

Y. Huang, Z. Song, K. Li, and S. Arora. Instahide: Instance-hiding schemes for private distributed learning. In *International Conference on Machine Learning*, pages 4507–4518. PMLR, 2020.

Y. Huang, S. Gupta, Z. Song, K. Li, and S. Arora. Evaluating gradient inversion attacks and defenses in federated learning. *Advances in Neural Information Processing Systems*, 34:7232–7241, 2021.

M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1345–1362, 2020.

M. Jegorova, C. Kaul, C. Mayor, A. Q. O'Neil, A. Weir, R. Murray-Smith, and S. A. Tsaftaris. Survey: Leakage and privacy at inference time. *arXiv preprint arXiv:2107.01614*, 2021.

Z. Ji and M. Telgarsky. Gradient descent aligns the layers of deep linear networks. In *International Conference on Learning Representations*, 2018.

Z. Ji and M. Telgarsky. Directional convergence and alignment in deep learning. *Advances in Neural Information Processing Systems*, 33:17176–17186, 2020.

A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

J. P. Lewis. Fast normalized cross-correlation. 1995. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.6062`.

Z. Li, Y. Luo, and K. Lyu. Towards resolving the implicit bias of gradient descent for matrix factorization: Greedy low-rank learning. In *International Conference on Learning Representations*, 2020.

B. Liu, M. Ding, S. Shaham, W. Rahayu, F. Farokhi, and Z. Lin. When machine learning meets privacy: A survey and outlook. *ACM Computing Surveys (CSUR)*, 54(2):1–36, 2021.

Y. Long, V. Bindschaedler, L. Wang, D. Bu, X. Wang, H. Tang, C. A. Gunter, and K. Chen. Understanding membership inferences on well-generalized learning models. *arXiv preprint arXiv:1802.04889*, 2018.

K. Lyu and J. Li. Gradient descent maximizes the margin of homogeneous neural networks. *arXiv preprint arXiv:1906.05890*, 2019.

A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.

F. A. Mejia, P. Gamble, Z. Hampel-Arias, M. Lomnitz, N. Lopatina, L. Tindall, and M. A. Barrios. Robust or private? adversarial training makes models more vulnerable to privacy attacks. *arXiv preprint arXiv:1906.06449*, 2019.

L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706. IEEE, 2019.

S. Milli, L. Schmidt, A. D. Dragan, and M. Hardt. Model reconstruction from model explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 1–9, 2019.

A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks. 2015.

E. Moroshko, B. E. Woodworth, S. Gunasekar, J. D. Lee, N. Srebro, and D. Soudry. Implicit bias in deep linear classification: Initialization scale vs training accuracy. *Advances in neural information processing systems*, 33:22182–22193, 2020.

M. S. Nacson, S. Gunasekar, J. Lee, N. Srebro, and D. Soudry. Lexicographic and depth-sensitive margins in homogeneous and non-homogeneous deep models. In *International Conference on Machine Learning*, pages 4683–4692. PMLR, 2019.

B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.

A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *Advances in neural information processing systems*, 29, 2016a.

A. Nguyen, J. Yosinski, and J. Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint arXiv:1602.03616*, 2016b.

A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4467–4477, 2017.

S. J. Oh, B. Schiele, and M. Fritz. Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 121–144. Springer, 2019.

C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2(11):e7, 2017.

C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

N. Razin and N. Cohen. Implicit regularization in deep learning may not be explainable by norms. *Advances in Neural Information Processing Systems*, 2020.

D. Rolnick and K. Kording. Reverse-engineering deep relu networks. In *International Conference on Machine Learning*, pages 8178–8187. PMLR, 2020.

A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *arXiv preprint arXiv:1806.01246*, 2018.

S. Santurkar, A. Ilyas, D. Tsipras, L. Engstrom, B. Tran, and A. Madry. Image synthesis with a single (robust) classifier. *Advances in Neural Information Processing Systems*, 32, 2019.

R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

L. Song and P. Mittal. Systematic evaluation of privacy risks of machine learning models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2615–2632, 2021.

D. Soudry, E. Hoffer, M. S. Nacson, S. Gunasekar, and N. Srebro. The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*, 19(1):2822–2878, 2018.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

N. Timor, G. Vardi, and O. Shamir. Implicit regularization towards rank minimization in relu networks. *arXiv preprint arXiv:2201.12760*, 2022.

F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.

D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

G. Vardi. On the implicit bias in deep-learning algorithms. *arXiv preprint arXiv:2208.12591*, 2022.

G. Vardi and O. Shamir. Implicit regularization in relu networks with the square loss. In *Conference on Learning Theory*, pages 4224–4258. PMLR, 2021.

G. Vardi, O. Shamir, and N. Srebro. On margin maximization in linear and relu networks. *arXiv preprint arXiv:2110.02732*, 2021.

B. Wang and N. Z. Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 36–52. IEEE, 2018.

Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

B. Woodworth, S. Gunasekar, J. D. Lee, E. Moroshko, P. Savarese, I. Golan, D. Soudry, and N. Srebro. Kernel and rich regimes in overparametrized models. In *Conference on Learning Theory*, pages 3635–3673. PMLR, 2020.

Z. Yang, J. Zhang, E.-C. Chang, and Z. Liang. Neural network inversion in adversarial setting via background knowledge alignment. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 225–240, 2019.

S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, pages 268–282. IEEE, 2018.

H. Yin, P. Molchanov, J. M. Alvarez, Z. Li, A. Mallya, D. Hoiem, N. K. Jha, and J. Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8715–8724, 2020.

H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov. See through gradients: Image batch recovery via gradinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16337–16346, 2021.

J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

C. Yun, S. Krishnan, and H. Mobahi. A unifying view on implicit bias in training linear neural networks. In *International Conference on Learning Representations*, 2020.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song. The secret revealer: Generative model-inversion attacks against deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 253–261, 2020.

L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. *Advances in Neural Information Processing Systems*, 32, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [N/A] We rely on known theoretical results, so proofs are not required.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] We do not have new assets.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We used only publicly available assets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We used only publicly available data.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

Table of Contents

A More Details on Privacy Attacks in Deep Learning

Below we discuss several privacy attacks that have been extensively studied in recent years (see Liu et al. [2021], Jegorova et al. [2021] for surveys).

Membership Inference. In *membership-inference attacks* [Shokri et al., 2017, Long et al., 2018, Salem et al., 2018, Yeom et al., 2018, Song and Mittal, 2021] the adversary determines whether a given data point was used to train the model or not. For example, if the model was trained on records of patients with a certain disease, the adversary might learn that an individual's record appeared in the training set and thus infer that the owner of the record has the disease with high chance. Note that membership inference attacks are significantly different from our attack, as the adversary must choose a specific data point. E.g., if the inputs are images, then the adversary must be able to guess a specific image.

Model Extraction. In *model-extraction attacks* [Tramèr et al., 2016, Oh et al., 2019, Wang and Gong, 2018, Carlini et al., 2020b, Jagielski et al., 2020, Milli et al., 2019, Rolnick and Kording, 2020, Chen et al., 2021] the adversary aims to steal the trained model functionality. In this attack, the adversary only has black-box access with no prior knowledge of the model parameters or training data, and the outcome of the attack is a model that is approximately the same as the target model. It was shown that in certain cases the adversary can reconstruct the exact parameters of the target model. We note that such attacks might be combined with our attack in order to allow extraction of the training dataset in a black-box setting. Namely, in the first stage the model is extracted using model-extraction attacks, and in the second stage the training dataset is reconstructed using our attack.

Model Inversion. *Model-inversion attacks* [Fredrikson et al., 2015] are perhaps the closest to our attack, as they consider reconstruction of input data given a trained model. These attacks aim to infer class features or construct class representatives, given that the adversary has some access (either black-box or white-box) to a model.

Fredrikson et al. [2015] showed that a face-recognition model can be used to reconstruct images of a certain person. This is done by using gradient descent for obtaining an input that maximizes the output probability that the face-recognition model assigns to a specific class. Thus, if a class contains only images of a certain individual, then by maximizing the output probability for this class we obtain an image that might be visually similar to an image of that person. It is important to note that the reconstructed image is not an actual example from the training set. Namely, it is an image that contains features which the classifier identifies with the class, and hence it might be visually similar to any image of the individual (including images from the training set). If the class members are not

all visually similar (which is generally the case), then the results of model inversion do not look like the training data (see discussions in Shokri et al. [2017] and Melis et al. [2019]). For example, if this approach is applied to the CIFAR-10 dataset, it results in images which are not human-recognizable [Shokri et al., 2017]. In Zhang et al. [2020], the authors leverage partial public information to learn a distributional prior via generative adversarial networks (GANs) and use it to guide the inversion process. That is, they generate images where the target model outputs a high probability for the considered class (as in Fredrikson et al. [2015]), but also encourage realistic images using GAN. We emphasize that from the reasons discussed above, this method does not reconstruct any specific training data point. Another approach for model inversion is training a model that acts as an inverse of the target model [Yang et al., 2019]. Thus, the inverse model takes the predicted confidence vectors of the target model as input, and outputs reconstructed data. A recent paper Balle et al. [2022] shows a reconstruction attack where the attacker has information about all the data samples except for one. On the theoretical side, Brown et al. [2021] prove that in certain settings, models memorize information about training examples, and show reconstruction attacks on some synthetic datasets.

Model inversion and information leakage in *collaborative deep learning* was studied in, e.g., He et al. [2019], Melis et al. [2019], Hitaj et al. [2017], Zhu et al. [2019], Yin et al. [2021], Huang et al. [2021]. Extraction of training data from language models was studied in Carlini et al. [2021, 2019], where they use the ability of language models to complete a given sentence in order to reveal sensitive information from the training data. We note that this attack is specific to language models, which are generative models, while our approach considers classifiers and is less specific.

Defences against Training Data Reconstruction. Avoiding leakage of sensitive information on the training dataset is the motivation behind *differential privacy* in machine learning, which has been extensively studied in recent years [Abadi et al., 2016, Dwork et al., 2006, Chaudhuri et al., 2011]. This approach allows provable guarantees on privacy, but it typically comes with high cost in accuracy. Other approaches for protecting the privacy of the training set, which do not allow such provable guarantees, have also been suggested (e.g., Huang et al. [2020], Carlini et al. [2020a]).

B Implementation Details

B.1 Hardware, Software and Running Time

A typical reconstruction runs for about 30 minutes on a GPU Tesla V-100 32GB, for reconstructing $m = 1000$ samples from a model with architecture d-1000-1000-1, and for $100,000$ epochs (running times slightly differ with the number of samples m, number of epochs and the size of the model, but it still takes about this time to run). Our code is implemented in PyTorch [Paszke et al., 2019]. We will release the code.

B.2 Hyperparameters

Our reconstruction scheme has 4 hyperparameters. Already discussed in the paper are the learning rate and $\sigma_{\mathbf{x}}$ (discussed in Subsection 5.2). In Subsection 5.2 we discuss the modification in the derivative of a ReLU layer $y = \max\{0, x\}$. The backward function of a ReLU layer works as follows: given the "gradient from above" $\frac{\partial L}{\partial y}$, the backward gradient is $\frac{\partial L}{\partial y} \cdot \mathbb{I}\{\mathbf{x} > 0\}$. Our modification to the backward gradient is $\frac{\partial L}{\partial y} \cdot \sigma(\alpha \mathbf{x})$, where $\sigma(z) = \frac{1}{1+e^{-z}}$ and α is a hyperparameter. As noted in the paper, this derivative is essentially the derivative of a SoftReLU, where the derivative is the same as ReLU for $\alpha \to \infty$ and is the derivative of the identity function for $\alpha \to 0$. Note that this is done only in the backward function, while the forward function remains that of a ReLU function. We also add an extra hyperparameter λ_{min} to our L_λ loss from Eq. (7):

$$L_\lambda(\lambda_1, \ldots, \lambda_m) = \sum_{i=1}^{m} \max\{-\lambda_i + \lambda_{\text{min}}, 0\}$$

The intuition behind is to encourage as many samples to lie on a margin, and thus try and reconstruct some sample from the training set.

To sum it all, the hyperparameters of our reconstruction scheme are:

1. Reconstruction learning rate
2. $\sigma_{\mathbf{x}}$, the initial scale of x_i initialization

3. α, of the derivative of the modified ReLU
4. λ_{min}

To find the set of hyperparamerers we used Weights&Biases [Biewald, 2020] using a random grid search where the parameters are sampled from the following distributions:

- Learning rate, log-uniform in $[10^{-5}, 1]$
- $\sigma_{\mathbf{x}}$, log-uniform in $[10^{-6}, 1]$
- ReLU derivative α, uniform in $[10, 500]$
- λ_{min}, log-uniform in $[10^{-4}, 1]$

When searching for hyperparameters for the model inversion results in Subsection 5.4 we use the following:

- Learning rate, log-uniform in $[10^{-6}, 1]$
- $\sigma_{\mathbf{x}}$, log-uniform in $[10^{-7}, 1]$

B.3 Post-Processing of Reconstructed Samples

After the reconstruction run ends we want to match the reconstructed samples to samples from the training set. This is done in the following manner:

1. **Scaling.** Each reconstructed sample is stretched to fit into the range $[0, 1]$ (by linear transformation of its minimal/maximal values).
2. **Searching Nearest Neighbours.** For each training sample from the training set we compute the distance to all reconstructed outputs using NCC [Lewis, 1995].
3. **Voting.** For each training sample we compute the mean of all the closest nearest neighbours (all reconstructed samples with NCC score largest than 0.9 of the distance to the closest nearest neighbour). Now we have pairs of trainig-sample and its reconstruction.
4. **Sorting.** For each pair we compute its SSIM [Wang et al., 2004], and sort the results by descending order.

C Supplementary Results

C.1 Results for Models in Figure 4

In this subsection we provide more details and experiments on each model presented in Figure 4. In Table 1 we show the train loss, test error and test loss of each model from Figure 4. All the models achieved a train accuracy of 100%. We note that adding more training samples improves the test accuracy, while adding more layers keeps the test accuracy approximately the same. In Figures 6-11 we show the best 45 extracted images (sorted by SSIM score) for the models presented in Figure 4. The reconstructions for the 50 and 500 samples with a d-1000-1000-1 architecture is presented in Figure 1 and Figure 3 (top) respectively.

C.2 All Comparisons for Subsection 5.4

In this subsection we provide more detailed results on the comparison to other methods as presented in Subsection 5.4. In Figure 12 and Figure 13 we provide more results from the model inversion attack on models trained on CIFAR10 and MNIST respectively. These are the same models from Figure 5 (a). In this attack, we either minimize or maximize the model's output w.r.t. a randomly initialized input. In this experiment, half of the initializations were maximized and the other half is minimized. The images are ordered by output of the model, in an increasing order. The results indicate that the model inversion attack mostly converge to similar reconstructions, even with many different initializations and different hyperparameters. Also, these reconstruction are mostly blurry, and probably represent the averages of each class.

Figure 6: **Architecture:** d-1000-1000-1, **Samples:** 100
Odd rows (1,3,5) are reconstructions, even rows (2,4,6) are the original data.

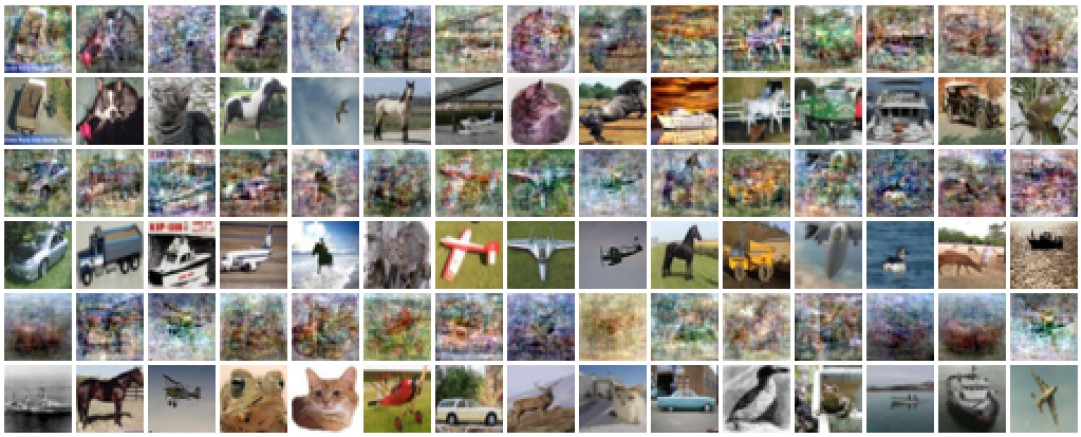


Figure 7: **Architecture:** d-1000-1000-1, **Samples:** 1000
Odd rows (1,3,5) are reconstructions, even rows (2,4,6) are the original data.

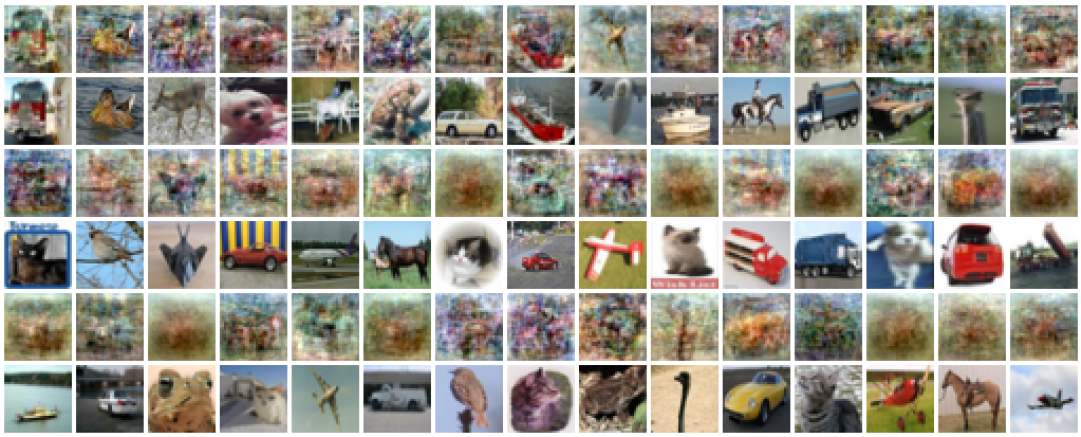


Figure 8: **Architecture:** d-100-100-1, **Samples:** 500
Odd rows (1,3,5) are reconstructions, even rows (2,4,6) are the original data.

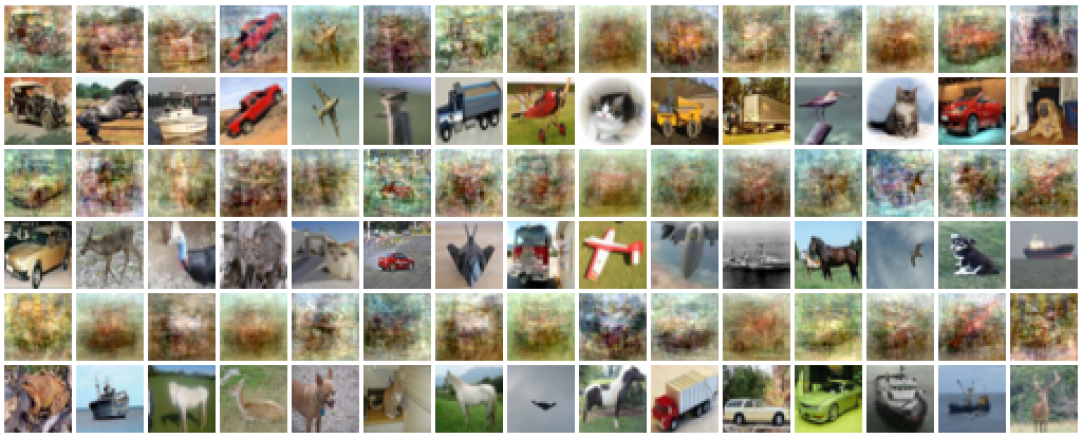


Figure 9: **Architecture:** d-1000-500-100-1, **Samples:** 500
Odd rows (1,3,5) are reconstructions, even rows (2,4,6) are the original data.

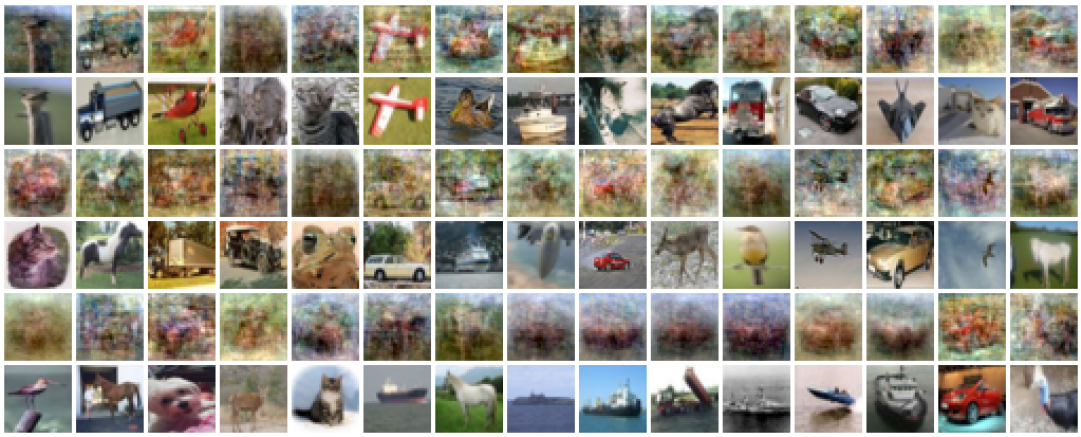


Figure 10: **Architecture:** d-1000-500-100-50-1, **Samples:** 500
Odd rows (1,3,5) are reconstructions, even rows (2,4,6) are the original data.

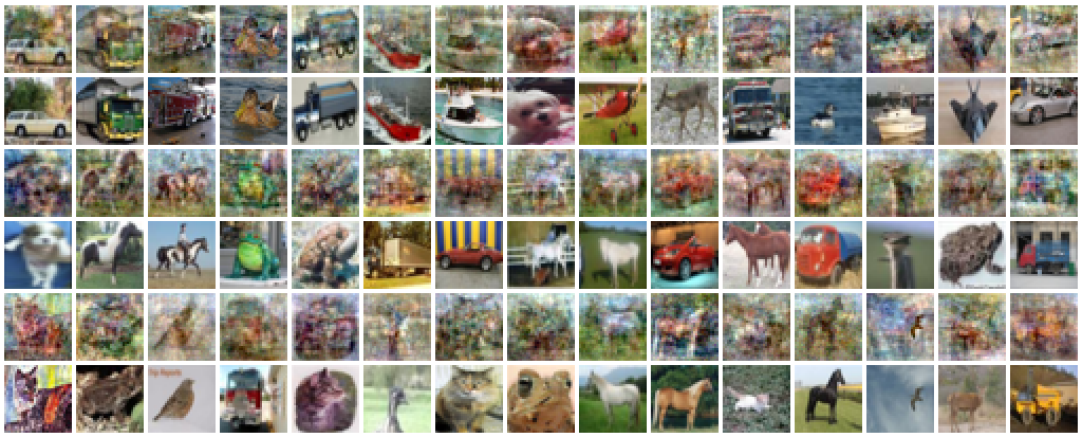


Figure 11: **Architecture:** d-1000-1000-1 (non-homogeneous), **Samples:** 500
Odd rows (1,3,5) are reconstructions, even rows (2,4,6) are the original data.

Architecture	Training Set Size (n)	Train Loss	Test Accuracy	Test Loss
1000-1000	50	$1.5 \cdot 10^{-6}$	72%	2.14
1000-1000	100	$2.0 \cdot 10^{-6}$	74%	2.41
1000-1000	500	$4.0 \cdot 10^{-6}$	78%	2.09
1000-1000	1000	$5.5 \cdot 10^{-6}$	79%	1.96
100-100	500	$3.0 \cdot 10^{-7}$	77%	2.72
1000-500-100	500	$1.2 \cdot 10^{-7}$	78%	3.14
1000-500-100-50	500	$8.4 \cdot 10^{-7}$	77%	2.57
1000-1000 (Non-Homogeneous)	500	$4.3 \cdot 10^{-6}$	77%	2.12

Table 1: Train/Test loss and Test Error for models shown in Figure 4

In Figure 14 and Figure 15 we show all the weights, as images, of the first fully-connected layer of models trained on CIFAR10 and MNIST respectively. These are the same models as in Figure 3, i.e., there are 1000 weights. Some weights are indicative of several input samples, e.g., a plane from CIFAR10 and the digits 8 and 5 from MNIST. We note that our reconstruction scheme is able to reconstruct much more samples, and in better quality than is represented in these weights.

C.3 Stretching the Theoretical Limitations

In this section we show results from several experiments which go beyond the theoretical limitations of Theorem 3.1.

Experiment	Training Set Size (n)	Train Loss	Test Accuracy	Test Loss
Standard Initialization	10	$8.3 \cdot 10^{-7}$	71%	1.68
Standard Initialization	50	$1.5 \cdot 10^{-6}$	74%	1.72
SGD	500	$4.0 \cdot 10^{-6}$	77%	2.21
10k Epochs (CIFAR10)	500	0.0039	77%	1.22
10k Epochs (MNIST)	500	0.014	87%	0.55

Table 2: Train/Test loss and Test Error for models shown in Figure 16

C.3.1 Standard Initialization Scale

In this subsection we consider networks trained with standard initialization scales. We recall that in the experiments presented in Section 5 the first fully-connected layer is initialized to a Gaussian distribution with mean 0 and standard deviation 10^{-4}, while the other layers are initialized by standard Kaiming initialization [He et al., 2015]. In Figure 17 and Figure 18 we show reconstructions of a model trained on CIFAR10 on 10 and 50 samples respectively, where the all the layers of the model are initialized by standard Kaiming initialization. The architecture of the model is d-1000-1000-1. We note that although the quality of the reconstructions is lower than when initializing the first layer with a small scale, there is still a strong signal that some of reconstructions correlate with training samples. It is an interesting future direction to improve the reconstruction quality for models with standard initialization.

In Figure 16 (a,b) we plot the SSIM score of each training sample against the output of the model. Note that indeed in these experiments the best SSIM score is lower than from other experiments presented in Figure 4. This corresponds to the lower quality of reconstructions when using standard initialization.

C.3.2 Less Epochs

In the experiments from Section 5 we trained each model for 10^6 epochs. The reason for this long training time is that Theorem 3.1 gives guarantees only when converging to KKT point. Such a convergence happens only after training until infinity, and longer training time may converge closer to the KKT point. In this section we provide reconstruction results for models trained for only 10^4 epochs. Figure 19 and Figure 20 show reconstructions for models trained on 500 samples from

Figure 12: Model inversion attack on a model trained on CIFAR10, with 500 samples. We reconstructed a total of $40,000$ images using different initializations and hyperparameters. We sorted the results according to the model's output, and selected 500 representative with index $i = 0, 80, 160, ..., 40000$.

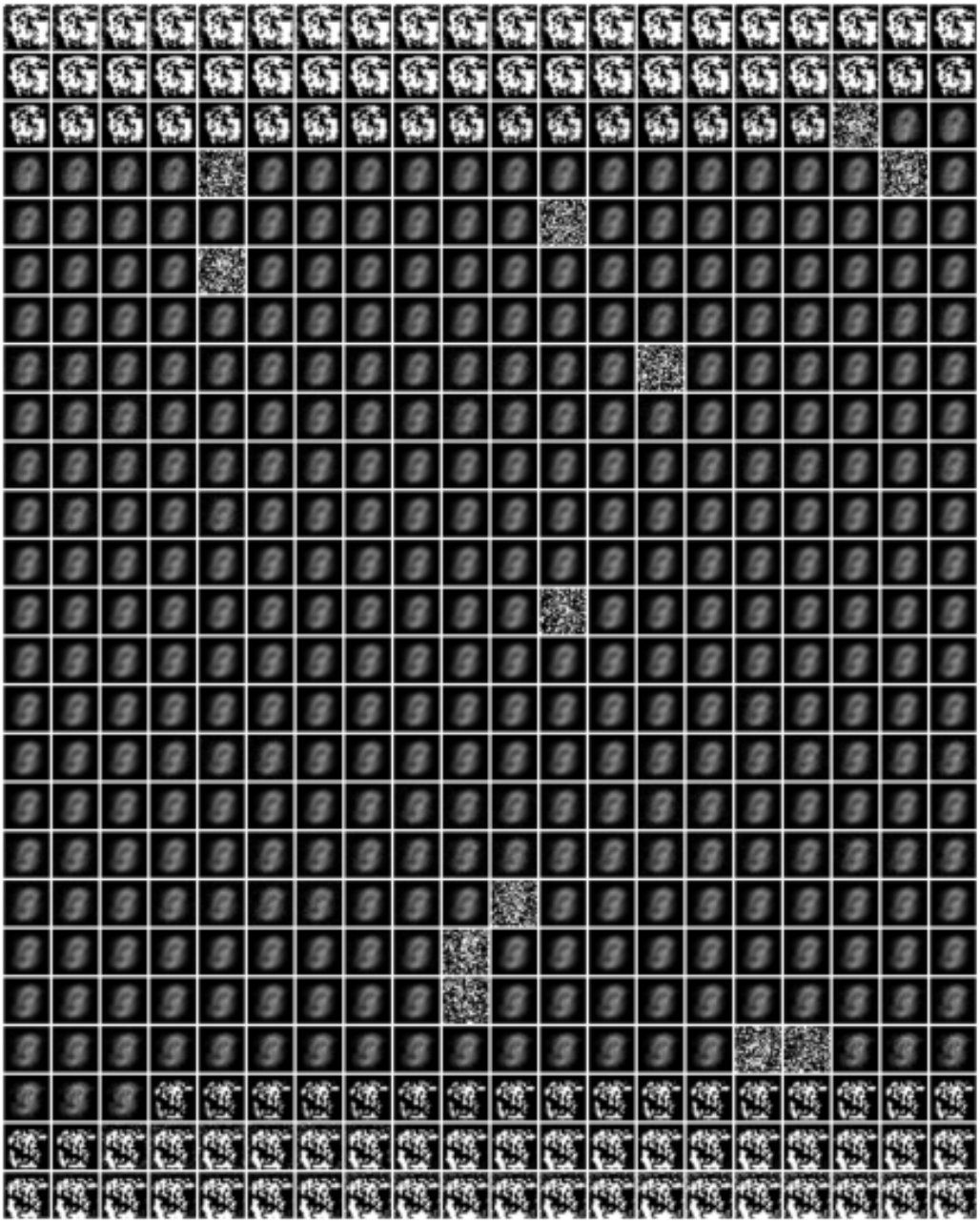


Figure 13: Model inversion attack on a model trained on MNIST, with 500 samples. We reconstructed a total of $40,000$ images using different initializations and hyperparameters. We sorted the results according to the model's output, and selected 500 representative with index $i = 0, 80, 160, ..., 40000$.

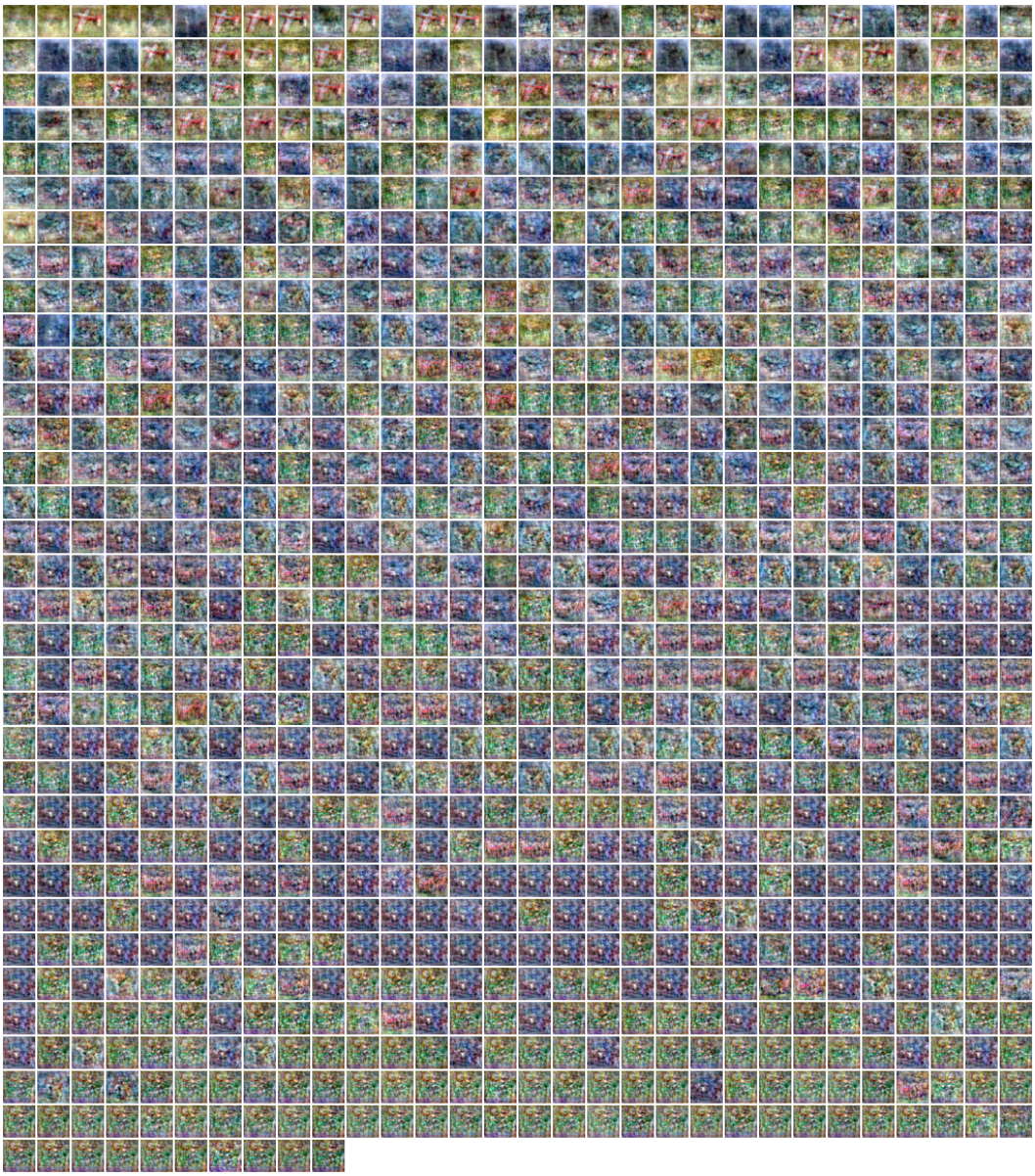


Figure 14: All the 1000 weights, shown as images, of the first fully-connected layer of a model trained on 500 samples on CIFAR10.

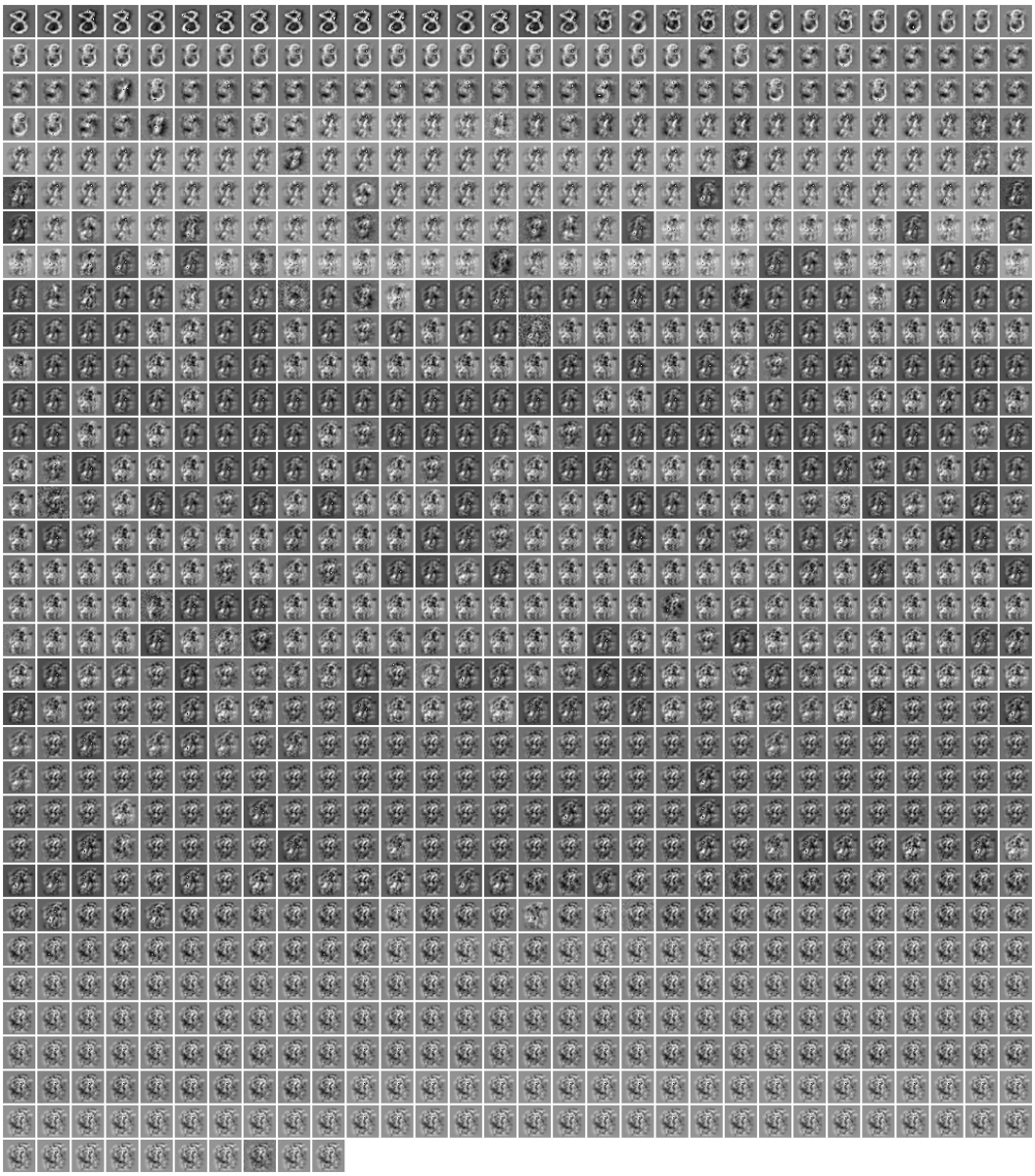


Figure 15: All the 1000 weights, shown as images, of the first fully-connected layer of a model trained on 500 samples on MNIST.

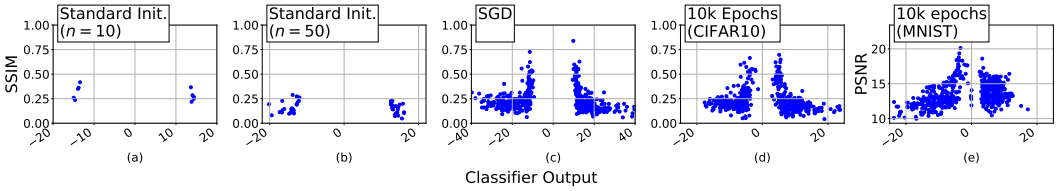


Figure 16: Each point represents a training sample. The y-axis is the highest SSIM score achieved by a reconstruction of this sample, the x-axis is the output of the model. From left to right: (a,b) models trained with standard Kaiming initialization in all layers on 10 and 50 CIFAR samples. (c) A model trained using SGD with a batch size of 50. (d,e) Models trained for 10^4 epochs on 500 samples from CIFAR and MNIST respectively.

Figure 17: Reconstructions from a model trained on 10 CIFAR10 images with labels animals vs. vehicles. In the first row are the reconstructions, and in the second row are their corresponding nearest neighbor from the dataset (sorted by SSIM score).

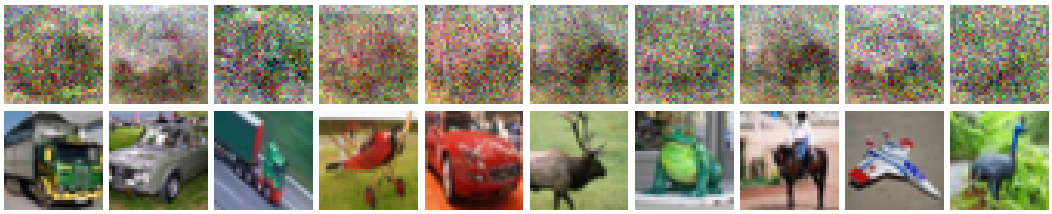


Figure 18: Top 10 reconstructions from a model trained on 50 CIFAR10 images with labels animals vs. vehicles. Top row shows reconstructions, and bottom row shows their corresponding nearest neighbor.

CIFAR10 and MNIST datasets respectively, with an architecture of d-1000-1000-1. It is clear that the quality of the reconstruction is very similar to when training for more epochs, this may indicate that even after significantly less training epochs the model converge sufficiently close to a KKT point.

In Figure 16 (d,e) we plot the SSIM score of each training sample against the output of the model. We note that we are able to reconstruct samples which appear approximately on the margin for both MNIST and CIFAR. In addition, the model for MNIST did not achieve 0 train error, and the margin is still very small. With that said, we are still able to reconstruct a large portion of the data with high quality. This goes beyond our theoretical limitations which have guarantees only for models which successfully label the entire training set.

C.3.3 Mini-batch SGD

In the experiments from Section 5 we trained the models using full-batch gradient descent. This was done to align with the theoretical guarantees of Theorem 3.1, which assume training with gradient flow. In Figure 21 we show reconstructions from a model trained with mini-batch SGD, using a batch size of 50. The model is trained on 500 images from CIFAR10, and with an architecture of d-1000-1000-1.

In Figure 16 (c) we plot the SSIM score of each training sample against the output of the model. This plot shows that we indeed reconstruct samples that lie on the margin.

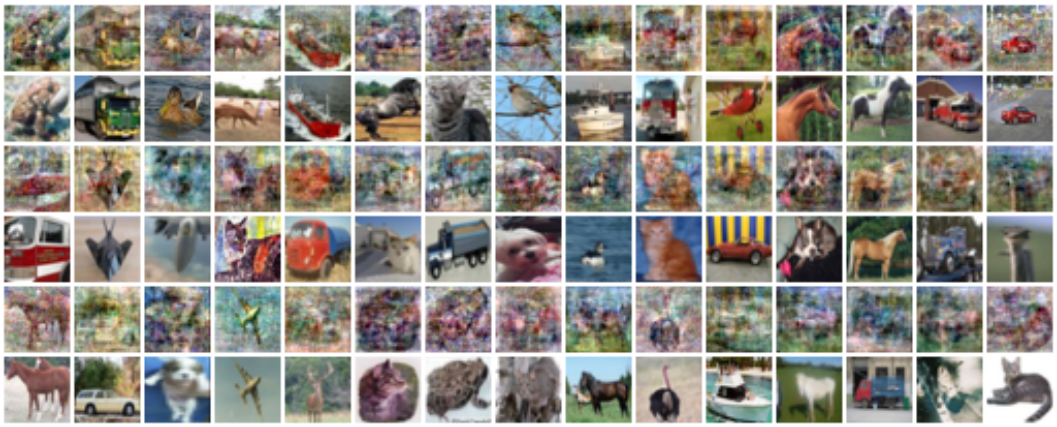


Figure 19: Reconstructions from a model trained for 10^4 epochs on CIFAR10 with labels animals vs. vehicles. Odd rows (1,3,5) are reconstruction, and even rows (2,4,6) are their nearest neighbor from the training samples.

Figure 20: Reconstructions from a model trained for 10^4 epochs on MNIST with labels odd vs. even. Odd rows (1,3,5) are reconstruction, and even rows (2,4,6) are their nearest neighbor from the training samples.

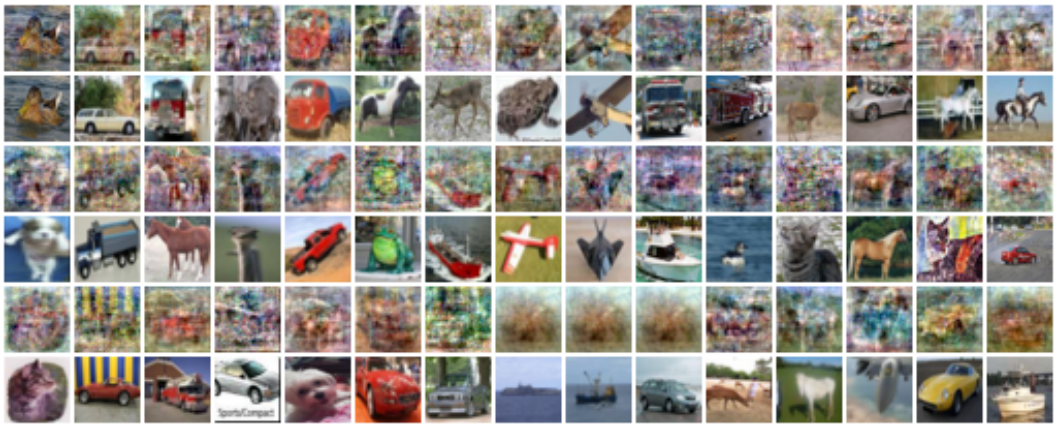


Figure 21: Reconstructions from a model trained using SGD with a batch size of 50. The model trained on 500 images from CIFAR10 with labels animals vs. vehicles. Odd rows (1,3,5) are reconstructions and even rows (2,4,6) are their nearest neighbor from the training dataset.