ABSTRACT

Title of Dissertation: ANALYSIS OF DATA SECURITY VULNERABILITIES IN DEEP LEARNING

Liam Fowl
Doctor of Philosophy, 2022

Dissertation Directed by: Professor Wojciech Czaja
Department of Mathematics

Professor Thomas Goldstein
Department of Computer Science

As deep learning systems become more integrated into important application areas, the security of such systems becomes a paramount concern. Specifically, as modern networks require an increasing amount of data on which to train, the security of data that is collected for these models cannot be guaranteed. In this work, we investigate several security vulnerabilities and security applications of the data pipeline for deep learning systems. We systematically evaluate the risks and mechanisms of data security from multiple perspectives, ranging from users to large companies and third parties, and reveal several security mechanisms and vulnerabilities that are of interest to machine learning practitioners.

ANALYSIS OF DATA SECURITY VULNERABILITIES IN DEEP LEARNING

by

Liam Fowl

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2022

Advisory Committee:
Professor Wojciech Czaja, Chair/Advisor
Professor Thomas Goldstein, Co-Advisor
Professor Vincent Lyzinski
Professor John Dickerson
Professor William Gasarch, Dean's Representative

Acknowledgments

There are so many people who have made it possible for me to complete my PhD. Academically, I owe much of my progress to my advisors: Wojtek Czaja and Tom Goldstein. Wojtek made me feel welcome in the department in my early years, and introduced me to machine learning. Furthermore, he has been an invaluable resource for projects, as well as university-related issues. Likewise, Tom welcomed me into his group and has been incredibly helpful with project direction. His insight and feedback about deep learning research has allowed me to publish several projects. I could not havae navigated the field of deep learning research without the help of both Wojtek and Tom.

Also key to my research has been collaborators. They have provided invaluable help on projects, as well as camraderie and advice. These collaborators include Jonas Geiping, Micah Goldblum, Zeyad Emam, Avi Schwarzschild, Ping-yeh Chiang, Ronny Huang, Gowthami Somepalli, Arpit Bansal, Chen Zhu, Yuxin Wen, Steven Reich, Renkun Ni, Valeria Cherapanova, Eitan Borgnia, Amin Ghiasi, Gavin Taylor, Michael Moller, Ali Shafahi, Nathaniel Monson, and many others.

Outside of academia, my friends have made my years in grad school incredibly enjoyable and I cannot thank them enough for the support that they provide.

Finally, I would like to thank my family who have given me encouragement and support throughout my years at UMD. My girlfriend, Carolina, has been incredibly pa-

tient and compassionate with me - especially in the last months of my program. Additionally, I've cherished the support from my brother, Brendan, and sister-in-law, Maddie. My uncles and aunts, as well as my grandparents also deserve my thanks and acknowledgement for their support throughout this process. Last, but not least, I owe so much to my parents who made college possible for me, and have shown me nothing but love and support throughout all my years of schooling.

Introduction

Deep learning has rapidly become the state-of-the-art approach for both image processing tasks [137] as well as natural language processing tasks [162]. This has led to significant advances in applications such as object detection [133], medical imaging [134], and several other important tasks [50, 93, 115]. This state-of-the-art performance has been fueled by both the increased capacity of modern networks, and also the increased availability of a large amount of data on which to train modern networks. High performing vision models are often trained on millions of images, and large language models can be trained on hundreds of millions of tokens [32, 33]. These data-hungry models are fed data that is often not curated. The security of this data is often not guaranteed and security risks to practitioners arise because of this. The data-collection pipeline can be roughly described by the interaction between three parties: companies, users, third parties. Each party has different goals with regards to data collection and security, and each interacts with the other two in unique ways. In this work, we study different aspects of this pipeline.

In Chapter 1, we study the security of data collected from third parties by companies. Here we study the threat of *targeted data poisoning* attacks wherein a third party disseminates mmaliciously manipulated data that is then scraped and used in training by a practitioner. We are the first work to show that modern deep networks trained at an industrial scale are vulnerable to this type of attack. Previous attacks worked only in

toy settings on simple datasets. Furthermore, we achieve state-of-the-art results on previous benchmarks all while showing significant computational advantages to our method - *gradient alignment*.

In Chapter 2, we study the interaction between companies/practitioners and third parties from the *opposite* perspective. We begin by motivating the concept of secure dataset release as a way for companies, like social media companies, to release user data while also preventing third parties from scraping said data and training a high-utility model on such data. We formally phrase this problem as an *availability poisoning* attack and we show that adversarial examples, originally intended as test-time attacks on neural networks, make highly effective availability poisons. We show that such poisons can often degrade accuracy of a network trained on this data to below random accuracy levels. We analyze the mechanism by which this attack works, and compare to several existing poisoning methods, finding that our method achieves state-of-the-art results on several datasets.

In Chapter 3, we take the perspective of users and evaluate the security of federated learning systems implemented by companies and third parties. We find that secure aggregation, which was previously thought to be a sufficient mechanism to ensure user privacy in practice, can be circumvented with malicious servers who wish to breach user privacy. We accomplish this task by introducing malicious model and parameter modifications to the federated learning pipeline that create structured gradient entries in the shared model updates. This can reveal verbatim copies of user data to the server. We show that our method outperforms previous "honest-but-curious" attacks by orders of magnitude. Our work suggests that further privacy techniques are needed to ensure user

data is secure.

In Chapter 4, we investigate the safety of federated learning for text data. Text data is perhaps the most important application area for federated learning, and one where we know federated learning is actually practiced. We introduce a new threat model of *malicious parameters* where a server is allowed to send "snapshots" of malicious parameters, but is not allowed to modify the underlying architecture. We show that Transformer-based language models are especially vulnerable to privacy attacks due to the large linear layers included in Transformer blocks. We show that recovered embeddings can be matched to know positions and tokens, and we introduce an attention mechanism modification that introduces sequence coding into embeddings. This allows a malicious server to recover a large amount of data from mulitple users - a setting that had previously not been investigated. We vastly outperform the state-of-the-art for "honest-but-curious" attacks on federated language models and demonstrate the threat posed by such an attack on multiple commonly used architectures.

We believe that revealing such threats to the data-security pipeline is important for both practitioners and users, and subsequent investigation, including investigation into defenses, should be performed.

Table of Contents

Chapter 0: Preliminaries

0.1 Neural Networks

Historically, neural networks (sometimes referred to as artificial neural networks), arose in an attempt to understand the behavior of the brain [101]. The first implentation of a neural network, called a perceptron, was developed by Frank Rosenblatt in 1958 [135]. While on neural networks continued after the introduction of the perceptron, growth in the field of machine learning accelerated with increased computational capacity that emerged in the 21st century. This increased capacity led to the advent of *deep learning* which loosely comprises the area of research surrounding training, optimizing and deploying modern networks of increased depth and capacity [54]. Nowadays, there exist several flavors of neural networks. One of the most common types is the *feed-forward neural network*.

Broadly speaking, feed-forward neural networks are compositions of affine functions with non-linearities. For example, a three layer neural network could look like: $f^{(3)}(f^{(2)}(f^{(1)}(x)))$ where $f^{(i)}(x) = g(W^{(i)}x + b^{(i)})$ and g is a non-linearity such as ReLU ($ReLU(x) = \max(0, x)$) that operates pointwise on the output of the affine function parameterized by weights $W^{(i)}$ and biases $b^{(i)}$ [54]. The aim of such a composition is to approximate some function f^*. For example, f^* could be a classifier of images. Feed-

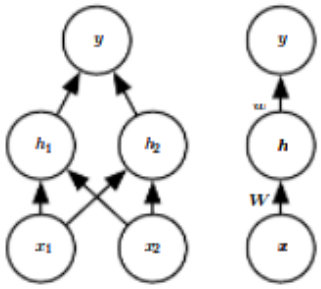


Figure 0.1: A feed-forward MLP with one hidden layer with two neurons trained to represent the "XOR" function [54]

.

forward networks where the weights $W^{(i)}$ are general linear functions are often called multi-layer perceptrons, or MLPs [54].

0.1.1 Convolutional Networks

There exist several variants of feed-forward networks. One of the most common variants is a convolutional feed-forward network [90]. A convolutional feed-forward network is a network wherein the linear maps determined by $W^{(i)}$ represent convolutions with some kernel rather than general linear maps. Convolution of a 2-d input I with a 2-d kernel K is defined as [54]:

$$(K * I)(i, j) = \sum_k \sum_l K(k, l) I(i + k, j + l) \tag{0.1}$$

The linear maps resulting from convolutions take the form of *doubly block circulant matrices* [54]. Convolutional filters enforce translationally invariant feature extraction, and are the backbone of many of the most successful classification and detection networks [59].

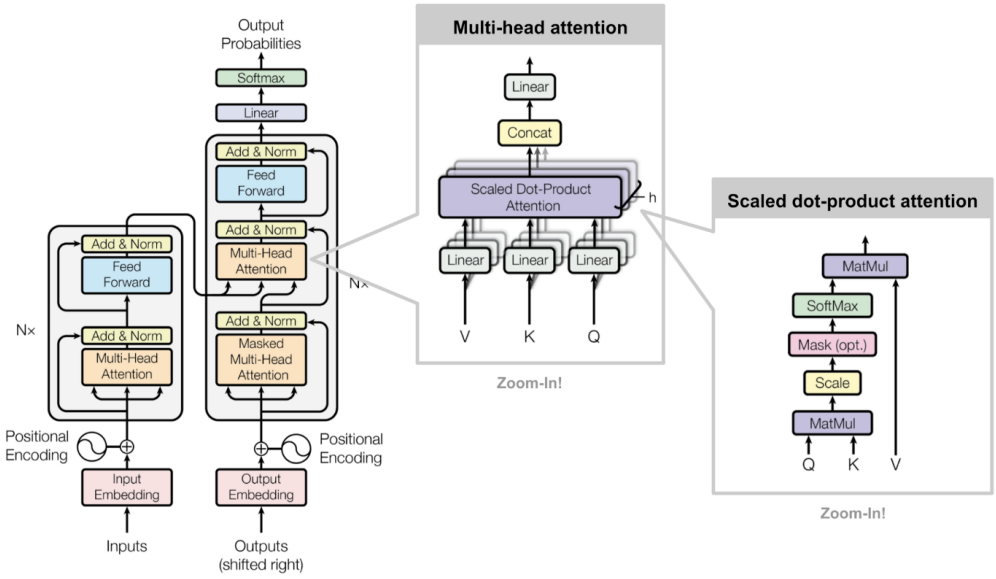


Figure 0.2: A Transformer model (diagram taken from [49, 162]).

0.1.2 Transformer Models

Recently, a new type of feed-forward network has emerged as a powerful tool for both image classification and language processing tasks. The *Transformer* model combines standard feed-forward layers with attention mechanisms to achieve competitive results in fields traditionally dominated by recurrent neural networks [162] or convolutional networks [33]. The attention mechanism of transformer models allows for efficient and parallel training of transformer models compared with recurrent neural networks. A diagram of the Transformer architecture can be found in Figure 0.2.

0.2 Training Neural Networks

Practitioners aim to find network parameters that minimize some loss $\mathcal{L}$ which evaluates the output of the network on some distribution $\mathcal{D}$ of interest. For example, the loss

for a model f with parameters θ can be expressed as:

$$\mathcal{L}^*(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} L(f(x;\theta), y).$$

However, when it comes to actually optimizing the parameters θ, this problem is usually converted to an *empirical risk minimization* problem wherein the practitioner minimizes

$$\sum_{(x,y)\in\mathcal{D}_{\text{train}}} \mathcal{L}(f(x;\theta), y).$$

Common loss functions include cross-entropy loss [54] for classification tasks, defined as:

$$\mathcal{L}(y^*, y) = -\sum_i y_i \log(y^*)_i,$$

for ground-truth label y^* and network output y (fed through a softmax function to produce prbabilities. Both y^*, y are discreet distributions over the label space. Another common loss function is regularized l_2 loss for regression tasks. This is defined as:

$$\mathcal{L}(y^*, y) = \sum_i (y_i - y_i^*)^2 + \alpha||\theta||_2^2,$$

or regularized l_2 loss for regression tasks.

Another piece of the training puzzle is the choice of optimization algorithm. One of the most common choices is the stochastic gradient descent (SGD) algorithm [54]. In this algorithm, the parameters θ are updated using the gradient direction from a *stochastic*

minibatch as follows:

$$\theta^{i+1} = \theta^i - \eta \nabla_\theta \sum_i \mathcal{L}(f(x^{(i)}; \theta), y^{(i)}),$$

for some minibatch $\{x^{(1)}, \ldots, x^{(m)}\}$ and some learning rate η [54]. Other algorithms, like SGD with momentum, and ADAM [80] have been shown to be successful in certain settings.

0.3 Datasets

Datasets are another fundamental component of neural network training. In vision, there exist several datasets of varying complexity and size. One common dataset is CIFAR-10 [86]. which consists of color images of size 32x32 coming from 10 classes. Usually, the dataset is split into 50,000 training examples, 5,000 from each class, and 10,000 testing images.

A more complex and much larger dataset commonly used for vision tasks is the ILSVRC2012 (ImageNet) dataset [137]. This dataset consists of over 1,000,000 color images from 1,000 classes. The images are of varying size, but a common preprocessing step is to crop the images to size 224x224. Still other datasets exist for different tasks like detection, segmentation, etc.

For natural language processing tasks, one common dataset is the WikiText dataset, which consists of over 100,000,000 tokens from curated Wikipedia articles [108]. We also utilize the Shakespeare dataset [77] consisting of 40,000 lines from different Shakespeare plays.

0.4 Attacks on Deep Learning Systems

Attacks on deep learning systems are an important area of research in the direction of deploying deep learning systems in security-critical applications. One avenue of attack is data manipulation. In this vein, attacks can generally be classified as train-time attacks or inference-time (or test-time) attacks.

0.4.1 Adversarial Examples

Adversarial examples are inference-time attacks against neural networks [55]. These adversarial examples are minimally perturbed inputs that cause an already trained network to mis-classify the modified examples. For example, an image of a dog could be minimally perturbed, maintaining the semantic label dog to human observers, while being classified as a cat by a network. The most common method for producing adversarial examples is projected gradient descent (PGD) which iteratively takes steps maximizing the loss with respect to the input pixels, and projecting onto an ℓ_p ball [99].

0.4.2 Data Poisoning

In contrast to adversarial examples, data poisoning attacks are train-time attacks. These attacks involve maliciously modifying data on which an unwitting practitioner trains a neural network. This is generally considered to be a more difficult problem than crafting adversarial examples for a network since the training procedure, and thus final parameters of a network are not known to the poisoner. Several attacks, and several goals exist within the data poisoning literature, including targeted data poisoning attacks which

aim to cause mis-classification of a pre-selected target datapoint, or availability attacks which aim to degrade overall performance of a victim network on some distribution [6].

0.5 Federated Learning

Federated learning (FL) is a system for training networks in a distributed fashion. Generally speaking, in standard training, a company could collect user data $\{(x_i, y_i)\}_{i=1}^N$ from users, and simply optimize

$$\arg\min_\theta \sum_i \mathcal{L}(f(x_i; \theta), y_i).$$

However in FL, the company, or central server, only receives user *updates* [14]. Formally, the server receives updates $\{\theta_i^*\}_{i=1}^N$ from users, and then computes the new parameters

$$\theta^{k+1} = \theta^k - \eta \sum_{i=1}^N \alpha_i \theta_i^*.$$

The main advantage of this setup is user privacy as the user data never gets sent to the central server.

Chapter 1: Integrity Poisoning Attacks

Data poisoning attacks modify training data to maliciously control a model trained on such data. Previous attacks against deep neural networks have been limited in scope and success, working only in simplified settings or being prohibitively expensive for large datasets. In this work, we focus on a particularly malicious poisoning attack that is both "from scratch" and "clean label", meaning we analyze an attack that successfully works against new, randomly initialized models, and is nearly imperceptible to humans, all while perturbing only a small fraction of the training data. The central mechanism of this attack is matching the gradient direction of malicious examples. We analyze why this works, supplement with practical considerations. and show its threat to real-world practitioners, finding that it is the first poisoning method to cause targeted misclassification in modern deep networks trained from scratch on a full-sized, poisoned ImageNet dataset. Finally we demonstrate the limitations of existing defensive strategies against such an attack, concluding that data poisoning is a credible threat, even for large-scale deep learning systems. This work was performed together with Jonas Geiping, Ronny Huang, Wojtek Czaja, Gavin Taylor, and Tom Goldstein. My contributions include: jointly conceiving of the mechanism of gradient alignment, a substantial amount of the large-scale experiments, jointly formulating the theoretical result, and writing a substantial portion of the paper.

1.1 Introduction

Machine learning models have quickly become the backbone of many applications from photo processing on mobile devices and ad placement to security and surveillance [91]. These applications often rely on large training datasets that aggregate samples of unknown origins, and the security implications of this are not yet fully understood [119]. Data is often sourced in a way that lets malicious outsiders contribute to the dataset, such as scraping images from the web, farming data from website users, or using large academic datasets scraped from social media [157]. *Data poisoning* is a security threat in which an attacker makes imperceptible changes to data that can then be disseminated through social media, user devices, or public datasets without being caught by human supervision. The goal of a poisoning attack is to modify the final model to achieve a malicious goal. Poisoning research has focuses on attacks that achieve mis-classification of some predetermined target data in Shafahi et al. [141], Suciu et al. [151], i.e. implementing a backdoor - but other potential goals of the attacker include denial-of-service [143, 149], concealment of users [142], and introduction of fingerprint information [97]. These attacks are applied in scenarios such as social recommendation [63], content management [37, 92], algorithmic fairness [146] and biometric recognition [96]. Accordingly, industry practitioners ranked data poisoning as the most serious attack on ML systems in a recent survey of corporations [88].

In this work we show that efficient poisoned data can be created even in the setting of deep neural networks trained on large image classification tasks, such as ImageNet [137]. Previous work on data poisoning has often focused on either linear classification

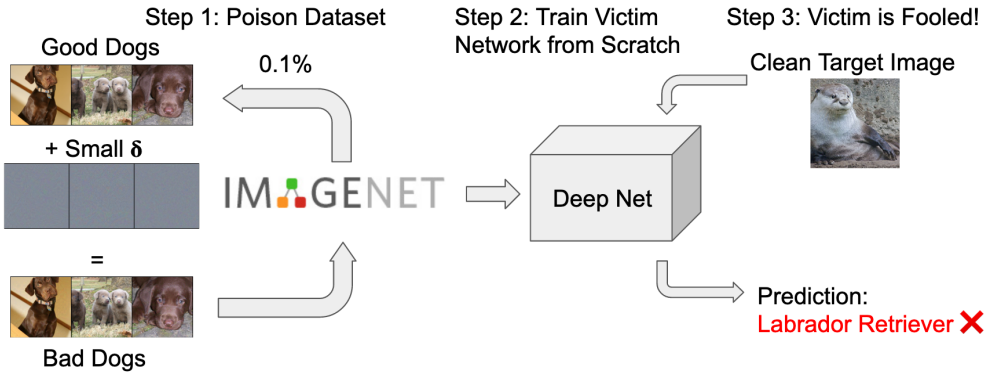


Figure 1.1: The poisoning pipeline. Poisoned images (labrador retriever class) are inserted into a dataset and cause a newly trained victim model to mis-classify a target (otter) image. We show successful poisons for a threat model where 0.1% of training data is changed within an ℓ_∞ bound of $\varepsilon = 8$. Further visualizations of poisoned data can be found in Section 1.6.

tasks [10, 84, 167] or poisoning of transfer learning and fine tuning [82, 141] rather than a full end-to-end training pipeline. Poison attacks on deep neural networks (and especially on ones trained from scratch) have proven difficult in Muñoz-González et al. [110] and Shafahi et al. [141]. Only recently were attacks against neural networks retrained from scratch shown to be possible in [68] for CIFAR-10 - however with costs that render scaling to larger datasets, like the ImageNet, prohibitively expensive.

We formulate data poisoning as the problem of solving a *gradient matching* problem and analyze the resulting novel attack algorithm that scales to unprecedented dataset size and effectiveness. Crucially, the new poisoning objective is orders-of-magnitude more efficient than a previous formulation based on on meta learning [68] and succeeds more often. We conduct an experimental evaluation, showing that poisoned datasets created by this method are robust and significantly outperform other attacks on CIFAR-10. We then demonstrate reliably successful attacks on common ImageNet models in realistic training scenarios. For example, the attack successfully compromises a ResNet-34 by

manipulating only 0.1% of the data points with perturbations less than 8 pixel values in ℓ_∞-norm. We close by discussing previous defense strategies and how strong differential privacy [1] is the only existing defense that can partially mitigate the effects of the attack.

1.2 Related Work

The task of data poisoning is closely related to the problem of adversarial attacks at test time, also referred to as evasion attacks [100, 155], where the attacker alters a target test image to fool an already-trained model. This attack is applicable in scenarios where the attacker has control over the target image, but not over the training data. An intermediary between data poisoning and adversarial attacks are backdoor trigger attacks [138, 160]. These attacks involve inserting a trigger – often an image patch – into training data, which is later activated by also applying the trigger to test images. Backdoor attacks require perturbations to both training and test-time data – a more permissive threat model than either poisoning or evasion.

In contrast to evasion and backdoor attacks, data poisoning attacks consider a setting where the attacker can modify training data, but does not have access to test data. Within this setting we focus on *targeted attacks* – attacks that aim to cause a specific target test image (or set of target test images) to be mis-classified. For example, an attack may cause a certain target image of a otter to be classified as a dog by victim models at test time. This attack is difficult to detect, because it does not noticeably degrade either training or validation accuracy [68, 141].

Two basic schemes for targeted poisoning are label flipping [6, 124], and water-

marking [141, 151]. In label flipping attacks, an attacker is allowed to change the label of examples, whereas in a watermarking attack, the attacker perturbs the training image, not label, by superimposing a target image onto training images. These attacks can be successful, yet they are easily detected by supervision such as Papernot and McDaniel [120]. This is in contrast to *clean-label* attacks which maintain the semantic labels of data.

Mathematically, data poisoning is a *bilevel* optimization problem [5, 10]; the attacker optimizes image pixels to enforce (malicious) criteria on the resulting network parameters, which are themselves the solution to an "inner" optimization problem that minimizes the training objective. Direct solutions to the bilevel problem have been proposed where feasible, for example, SVMs in Biggio et al. [10] or logistic regression in Demontis et al. [29]. However, direct optimization of the poisoning objective is intractable for deep neural networks because it requires backpropagating through the entire SGD training procedure, see [110]. As such, the bilevel objective has to be approximated. Recently, MetaPoison [68] proposed to approximately solve the bi-level problem based on methods from the meta-learning community [41]. The bilevel gradient is approximated by backpropagation through several unrolled gradient descent steps. This is the first attack to succeed against deep networks on CIFAR-10 as well as providing transferability to other models. Yet, [68] uses a complex loss function averaged over a wide range of models trained to different epochs and a single unrolling step necessarily involves both clean and poisoned data, making it roughly as costly as one epoch of standard training. With an ensemble of 24 models, [68] requires 3 (2 unrolling steps + 1 clean update step) x 2 (backpropagation through unrolled steps) x 60 (first-order optimization steps) x 24 (ensemble of models) equivalent epochs of normal training to attack, as well as ($\sum_{k=0}^{22} k = 253$)

epochs of pretraining. All in all, this equates to 8893 training epochs.

In contrast to bilevel approaches stand heuristics for data poisoning of neural networks. The most prominent heuristic is *feature collision*, as in Poison Frogs [141], which seeks to cause a target test image to be misclassified by perturbing training data to collide with the target image in feature space. Modifications surround the target image in feature space with a convex polytope [178] or collection of poisons [2]. These methods are efficient, but designed to attack fine-tuning scenarios where the feature extractor is nearly fixed. When applied to deep networks trained from scratch, their performance drops significantly.

1.3 Efficient Poison Brewing

In this section, we will discuss an intriguing weakness of neural network training based on first-order optimization and derive an attack against it. This attack modifies training images that so they produce a *malicious gradient signal* during training, even while appearing inconspicuous. This is done by matching the gradient of the target images within ℓ^∞ bounds. Because neural networks are trained by gradient descent, even minor modifications of the gradients can be incorporated into the final model.

This attack compounds the strengths of previous schemes, allowing for data poisoning as efficiently as in Poison Frogs [141], requiring only a single pretrained model and a time budget on the order of one epoch of training for optimization - but still capable of poisoning the from-scratch setting considered in [68]. This combination allow an attacker to ”brew” poisons that successfully attack realistic models on ImageNet.

1.3.1 Threat Model

We define two parties, the *attacker*, which has limited control over the training data, and the *victim*, which trains a model based on this data. We first consider a gray-box setting, where the attacker has knowledge of the model architecture used by its victim. The attacker is permitted to poison a fraction of the training dataset (usually less than 1%) by changing images within an ℓ_∞-norm ε-bound (e.g. with $\varepsilon \leq 16$). This constraint enforces *clean-label* attacks, meaning that the semantic label of a poisoned image is still unchanged. The attacker has no knowledge of the training procedure - neither about the initialization of the victim's model, nor about the (randomized) mini-batching and data augmentation that is standard in the training of deep learning models.

We formalize this threat model as bilevel problem for a machine learning model $F(x, \theta)$ with inputs $x \in \mathbb{R}^n$ and parameters $\theta \in \mathbb{R}^p$ (implicitly a vector-valued function of the perturbations), and loss function $\mathcal{L}$. We denote the N training samples by $(x_i, y_i)_{i=1}^N$, from which a subset of P samples are poisoned. For notational simplicity we assume the first P training images are poisoned by adding a perturbation Δ_i to the i^{th} training image. The perturbation is constrained to be smaller than ε in the ℓ_∞-norm. The task is to optimize Δ so that a set of T target samples $(x_i^t, y_i^t)_{i=1}^T$ is reclassified with the new adversarial labels y_i^{adv}:

$$\min_{\Delta \in \mathcal{C}} \sum_{i=1}^{T} \mathcal{L}\left(F(x_i^t, \theta(\Delta)), y_i^{\text{adv}}\right) \quad \text{s.t. } \theta(\Delta) \in \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(F(x_i + \Delta_i, \theta), y_i). \tag{1.1}$$

We subsume the constraints in the set $\mathcal{C} = \{\Delta \in \mathbb{R}^{N \times n} : ||\Delta||_\infty \leq \varepsilon, \Delta_i = 0 \; \forall i > P\}$. We call the main objective on the left the *adversarial loss*, and the objective that appears in the constraint on the right is the *training loss*. For the remainder, we consider a single target image ($T = 1$) as in Shafahi et al. [141], but stress that this is not a general limitation as shown in the appendix.

1.3.2 Motivation

What is the optimal alteration of the training set that causes a victim neural network $F(x, \theta)$ to mis-classify a specific target image x^t? We know that the expressivity of deep networks allows them to fit arbitrary training data [172]. Thus, if an attacker was unconstrained, a straightforward way to cause targeted mis-classification of an image is to insert the target image, with the incorrect label y^{adv}, into the victim networks training set. Then, when the victim minimizes the training loss they simultaneously minimize the adversarial loss, based on the gradient information about the target image. In our threat model however, the attacker is not able to insert the mis-labeled target. They can, however, still mimic the gradient of the target by creating poisoned data whose training gradient correlates with the adversarial target gradient. If the attacker can enforce

$$\nabla_\theta \mathcal{L}(F(x^t, \theta), y^{\text{adv}}) \approx \frac{1}{P} \sum_{i=1}^{P} \nabla_\theta \mathcal{L}(F(x_i + \Delta_i, \theta), y_i) \tag{1.2}$$

to hold for any θ encountered during training, then the victim's gradient steps that minimize the training loss on the poisoned data (right hand side) will also minimize the attackers adversarial loss on the targeted data (left side).

1.3.3 The Central Mechanism: Gradient Alignment

Gradient magnitudes vary dramatically across different stages of training, and so finding poisoned images that satisfy Equation 1.2 for all θ encountered during training is infeasible. Instead we *align* the target and poison gradients in the same direction, that is we minimize their negative cosine similarity. We do this by taking a clean model F with parameters θ, keeping θ fixed, and then optimizing

$$\mathcal{B}(\Delta, \theta) = 1 - \frac{\left\langle \nabla_\theta \mathcal{L}(F(x^t, \theta), y^{\text{adv}}), \sum_{i=1}^{P} \nabla_\theta \mathcal{L}(F(x_i + \Delta_i, \theta), y_i) \right\rangle}{\|\nabla_\theta \mathcal{L}(F(x^t, \theta), y^{\text{adv}})\| \cdot \| \sum_{i=1}^{P} \nabla_\theta \mathcal{L}(F(x_i + \Delta_i, \theta), y_i)\|}. \tag{1.3}$$

Algorithm 1 Poison Brewing via the discussed approach.

Require Pretrained clean network $\{F(\cdot, \theta)\}$, a training set of images and labels $(x_i, y_i)_{i=1}^N$, a target (x^t, y^{adv}), $P < N$ poison budget, perturbation bound ε, restarts R, optimization steps M
Begin
Select P training images with label y^{adv}
For $r = 1, \ldots, R$ restarts:
  Randomly initialize perturbations $\Delta^r \in \mathcal{C}$
  For $j = 1, \ldots, M$ optimization steps:
    Apply data augmentation to all poisoned samples $(x_i + \Delta_i^r)_{i=1}^P$
    Compute the average costs, $\mathcal{B}(\Delta^r, \theta)$ as in Equation 1.3, over all poisoned samples
    Update Δ^r with a step of signed Adam and project onto $||\Delta^r||_\infty \leq \varepsilon$
Choose the optimal Δ^* as Δ^r with minimal value in $\mathcal{B}(\Delta^r, \theta)$
Return Poisoned dataset $(x_i + \Delta_i^*, y_i)_{i=1}^N$

We optimize $\mathcal{B}(\Delta)$ using signed Adam updates with decaying step size, projecting onto $\mathcal{C}$ after every step. This produces an alignment between the averaged poison gradients and the target gradient.

In contrast to Poison Frogs, all layers of the network are included (via their parameters) in this objective, not just the last feature layer.

Each optimization step of this attack requires only a *single* differentiation of the parameter gradient w.r.t to its input, instead of differentiating through several unrolled steps as in MetaPoison. Furthermore, as in Poison Frogs we differentiate through a loss that only involves the (small) subset of poisoned data instead of involving the entire dataset, such that the attack is especially fast if the budget is small. Finally, the method is able to create poisons using only a single parameter vector, θ (like Poison Frogs in fine-tuning setting, but not the case for MetaPoison) and does not require updates of this parameter vector after each poison optimization step.

1.3.4 Making attacks that transfer and succeed "in the wild"

A practical and robust attack must be able to poison different random initializations of network parameters and a variety of architectures. To this end, we employ several techniques:

Differentiable Data Augmentation and Resampling: Data augmentation is a standard tool in deep learning, and transferable image perturbations must survive this process. At each step minimizing Equation 1.3, we randomly draw a translation, crop, and possibly a horizontal flip for each poisoned image, then use bilinear interpolation to resample to the original resolution. When updating Δ, we differentiate through this grid sampling operation as in Jaderberg et al. [70]. This creates an attack which is robust to data augmentation and leads to increased transferability.

Restarts: The efficiency we gained in Equation 1.3.3 allows us to incorporate restarts, a common technique in the creation of evasion attacks [109, 130]. We minimize Equation

1.3 several times from random starting perturbations, and select the set of poisons that give us the lowest alignment loss $\mathcal{B}(\Delta)$. This allows us to trade off reliability with computational effort.

Model Ensembles: A known approach to improving transferability is to attack an ensemble of model instances trained from different initializations [68, 95, 178]. However, ensembles are highly expensive, increasing the pre-training cost for only a modest, but stable, increase in performance.

We show the effects of these techniques via CIFAR-10 experiments (see Table 1.1 and Section 1.5.1). To keep the attack within practical reach, we do not consider ensembles for our experiments on ImageNet data, opting for the cheaper techniques of restarts and data augmentation. A summarizing description of the attack can be found in Algorithm 1. Lines 8 and 9 of Algorithm 1 are done in a stochastic (mini-batch) setting (which we omitted in Algorithm 1 for notational simplicity).

1.4 Theoretical Analysis

Can gradient alignment cause network parameters to converge to a model with low adversarial loss? To simplify presentation, we denote the adversarial loss and normal training loss of Equation 1.1 as

$$\mathcal{L}_{\text{adv}}(\theta) =: \mathcal{L}(F((x^t, \theta), y^{\text{adv}}),$$

and

$$\mathcal{L}(\theta) =: \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(x_i, y_i, \theta),$$

respectively. Also, recall that $1 - \mathcal{B}\left(\Delta, \theta^k\right)$, defined in Equation 1.3, measures the cosine similarity between the gradient of the adversarial loss and the gradient of normal training loss. We adapt a classical result of Zoutendijk [116, Thm. 3.2] to shed light on why data poisoning can work even though the victim only performs standard training on a poisoned dataset:

Proposition 1.4.1 (Adversarial Descent)**.** *Let $\mathcal{L}_{adv}(\theta)$ be bounded below and have a Lipschitz continuous gradient with constant $L > 0$ and assume that the victim model is trained by gradient descent with step sizes α_k, i.e. $\theta^{k+1} = \theta^k - \alpha_k \nabla\mathcal{L}(\theta^k)$. If the gradient descent steps $\alpha_k > 0$ satisfy*

$$0 < \alpha_k L < \beta\left(1 - \mathcal{B}(\Delta, \theta^k)\right)\frac{||\nabla\mathcal{L}(\theta^k)||}{||\nabla\mathcal{L}_{adv}(\theta^k)||}, \tag{1.4}$$

for some fixed $\beta < 1$, then $\mathcal{L}_{adv}(\theta^{k+1}) < \mathcal{L}_{adv}(\theta^k)$. If in addition $\exists\varepsilon > 0,\ k_0$ so that $\forall k \geq k_0$, $\mathcal{B}(\Delta, \theta^k) < 1 - \varepsilon$, then

$$\lim_{k\to\infty} ||\nabla\mathcal{L}_{adv}(\theta^k)|| \to 0. \tag{1.5}$$

Proof. Consider the gradient descent update

$$\theta^{k+1} = \theta^k - \alpha_k \nabla\mathcal{L}(\theta^k)$$

Firstly, due to Lipschitz smoothness of the gradient of the adversarial loss $\mathcal{L}_{\text{adv}}$ we can estimate the value at θ^{k+1} by the descent lemma

$$\mathcal{L}_{\text{adv}}(\theta^{k+1}) \leq \mathcal{L}_{\text{adv}}(\theta^k) - \langle \alpha_k \nabla \mathcal{L}_{\text{adv}}(\theta^k), \nabla \mathcal{L}(\theta^k) \rangle + \alpha_k^2 L ||\nabla \mathcal{L}(\theta^k)||^2$$

If we further use the cosine identity:

$$\langle \nabla \mathcal{L}_{\text{adv}}(\theta^k), \nabla \mathcal{L}(\theta^k) \rangle = ||\nabla \mathcal{L}(\theta^k)|| ||\nabla \mathcal{L}_{\text{adv}}(\theta^k)|| \cos(\gamma^k),$$

denoting the angle between both vectors by γ^k, we find that

$$\begin{aligned} \mathcal{L}_{\text{adv}}(\theta^{k+1}) &\leq \mathcal{L}_{\text{adv}}(\theta^k) - \alpha_k ||\nabla \mathcal{L}(\theta^k)|| ||\nabla \mathcal{L}_{\text{adv}}(\theta^k)|| \cos(\gamma^k) + \alpha_k^2 L ||\nabla \mathcal{L}(\theta^k)||^2 \\ &= \mathcal{L}_{\text{adv}}(\theta^k) - \left(\alpha_k \frac{||\nabla \mathcal{L}_{\text{adv}}(\theta^k)||}{||\nabla \mathcal{L}(\theta^k)||} \cos(\gamma^k) - \alpha_k^2 L \right) ||\nabla \mathcal{L}(\theta^k)||^2 \end{aligned}$$

As such, the adversarial loss decreases for nonzero step sizes if

$$\frac{||\nabla \mathcal{L}_{\text{adv}}(\theta^k)||}{||\nabla \mathcal{L}(\theta^k)||} \cos(\gamma^k) > \alpha_k L$$

i.e.

$$\alpha_k L \leq \frac{||\nabla \mathcal{L}_{\text{adv}}(\theta^k)||}{||\nabla \mathcal{L}(\theta^k)||} \frac{\cos(\gamma^k)}{c}$$

for some $1 < c < \infty$. This follows from our assumption on the parameter β in the statement of the proposition. Reinserting this estimate into the descent inequality reveals

that

$$\mathcal{L}_{\text{adv}}(\theta^{k+1}) < \mathcal{L}_{\text{adv}}(\theta^k) - ||\nabla \mathcal{L}_{\text{adv}}||^2 \frac{\cos(\gamma^k)}{c' L},$$

for $\frac{1}{c'} = \frac{1}{c} - \frac{1}{c^2}$. Due to monotonicity we may sum over all descent inequalities, yielding

$$\mathcal{L}_{\text{adv}}(\theta^0) - \mathcal{L}_{\text{adv}}(\theta^{k+1}) \geq \frac{1}{c' L} \sum_{j=0}^{k} ||\nabla \mathcal{L}_{\text{adv}}(\theta^j)||^2 \cos(\gamma^j)$$

As $\mathcal{L}_{\text{adv}}$ is bounded below, we may consider the limit of $k \to \infty$ to find

$$\sum_{j=0}^{\infty} ||\nabla \mathcal{L}_{\text{adv}}(\theta^j)||^2 \cos(\gamma^j) < \infty.$$

If for all, except finitely many iterates the angle between adversarial and training gradient is less than $180°$, i.e. $\cos(\gamma^k)$ is bounded below by some fixed $\epsilon > 0$, as assumed, then the convergence to a stationary point follows:

$$\lim_{k \to \infty} ||\nabla \mathcal{L}_{\text{adv}}(\theta^k)|| \to 0$$

□

In Figure 1.2 we visualize measurements of the computed bound from an actual poisoned training. The classical gradient descent converges only if $\alpha_k L < 1$, so we can find an upper bound to this value by 1, even if the actual Lipschitz constant of the neural network training objective is not known to us.

Put simply, our poisoning method aligns the gradients of training loss and ad-

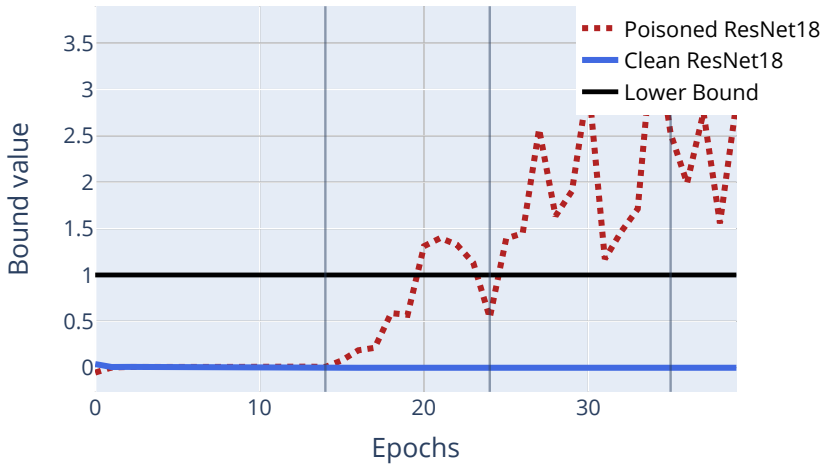
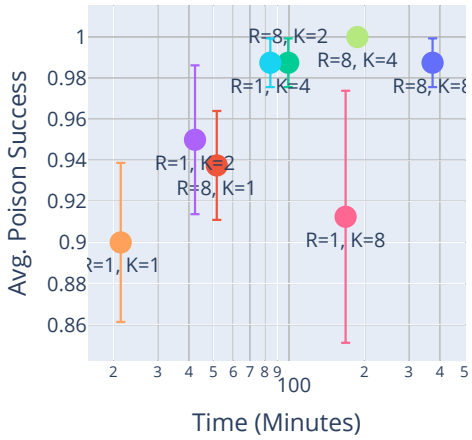


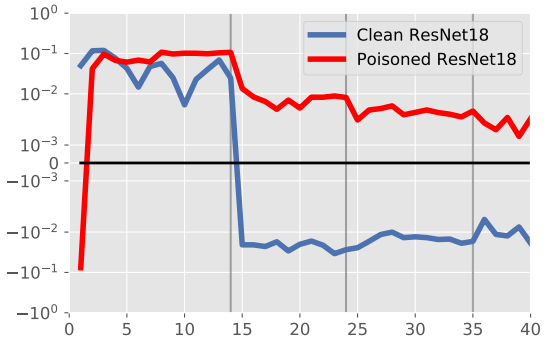
Figure 1.2: The bound considered in Prop. 1, evaluated during training of a poisoned and a clean model, using a practical estimation of the lower bound via $\alpha_k L \approx 1$. This is an upper bound of $\alpha_k L$ as $\alpha_k < \frac{1}{L}$ is necessary for the convergence of (clean) gradient descent.

versarial loss. This enforces that the gradient of the main objective is a descent direction for the adversarial objective, which, when combined with conditions on the step sizes, causes a victim to unwittingly converge to a stationary point of the adversarial loss, i.e. optimize *the original bilevel objective* locally.

The strongest assumption in Proposition 1.4.1 is that gradients are almost always aligned, $\mathcal{B}(\Delta, \theta^k) < 1 - \epsilon, k \geq k_0$. We directly maximize alignment during creation of the poisoned data, but only for a selected θ^*, and not for all θ^k encountered during gradient descent from any possible initialization. However, poison perturbations made from one parameter vector, θ, can transfer to other parameter vectors encountered during training. For example, if one allows larger perturbations, and in the limiting case, unbounded perturbations, our objective is minimal if the poison data is identical to the target image, which aligns training and adversarial gradients at every θ encountered. Empirically, we



(a) Crafting time versus poisons success for different hyperparameters.



(b) Average batch cosine similarity, per epoch, between the adversarial gradient $\nabla\mathcal{L}_{\text{adv}}(\theta)$ and the gradient of each mini-batch $\nabla\mathcal{L}(\theta)$ for a poisoned and a clean ResNet-18. Crucially, the gradient alignment is strictly positive after a small number of epochs.

see that the proposed "poison brewing" attack does indeed increase gradient alignment. In Figure 1.3b, we see that in the first phase of training all alignments are positive, but only the poisoned model maintains a positive similarity for the adversarial target-label gradient throughout training. The clean model consistently shows that these angles are negatively aligned - i.e. normal training on a clean dataset will increase adversarial loss. However, after the inclusion of poisoned data, the gradient alignment is modified enough to change the prediction for the target.

1.5 Experimental Evaluation

We evaluate poisoning approaches in each experiment by sampling 10 random poison-target cases. We compute poisons for each and evaluate them on 8 newly initialized victim models (see Section 1.7 for details of our methodology). We use the following hyperparameters for all our experiments: $\tau = 0.1$, $R = 8$, $M = 250$. We train victim models in a realistic setting, considering data augmentation, SGD with momentum,

weight decay and learning rate drops.

1.5.1 Evaluations on CIFAR-10

As a baseline on CIFAR-10, we compare the number of restarts R and the number of ensembled models K, showing that the proposed method is successful in creating poisons even with just a single model (instead of an ensemble). The inset figure shows poison success versus time necessary to compute the poisoned dataset for a budget of 1%, $\varepsilon = 16$ on CIFAR-10 for a ResNet-18. We find that as the number of ensemble models, K, increases, it is beneficial to increase the number of restarts as well, but increasing the number of restarts independently also improves performance. We validate the differentiable data augmentation discussed in Section 1.3.4 in Table 1.1, finding it crucial for scalable data poisoning, being as efficient as a large model ensemble in facilitating robustness.

Next, to test different poisoning methods, we fix our "brewing" framework of efficient data poisoning, with only a single network and diff. data augmentation. We evaluate the discussed gradient matching cost function, replacing it with either the feature-collision objective of Poison Frogs or the bullseye objective of Aghakhani et al. [2], thereby effectively replicating their methods, but in our context of from-scratch training.

Table 1.1: CIFAR-10 ablation. $\varepsilon = 16$, budget is 1%. Differentiable data augmentation is able to replace a large 8-model ensemble, without increasing computational effort.

Ensemble	**Diff. Data Aug.**	**Victim does data aug.**	**Poison Accuracy** ($\%(\pm$SE))
1	X	X	100.00% ($\pm$0.00)
1	X	✓	32.50% ($\pm$12.27)
8	X	X	78.75% ($\pm$11.77)
1	✓	✓	91.25% ($\pm$6.14)

Table 1.2: CIFAR-10 Comparison to other poisoning objectives with a budget of 1% within our framework (columns 1 to 3), for a 6-layer ConvNet and an 18-layer ResNet. MetaPoison* denotes the full framework of Huang et al. [68]. Each cell shows the avg. poison success and its standard error.

	Proposed	Bullseye	Poison Frogs	MetaPoison*
ConvNet ($\varepsilon = 32$)	*86.25%* ($\pm$9.43)	78.75% ($\pm$7.66)	52.50% ($\pm$12.85)	35.00% ($\pm$11.01)
ResNet-18 ($\varepsilon = 16$)	*90.00%* ($\pm$3.87)	3.75% ($\pm$3.56)	1.25% ($\pm$1.19)	42.50 % ($\pm$8.33)

The results of this comparison are collated in Table 1.2. While Poison Frogs and Bullseye succeeded in finetuning settings, we find that their feature collision objectives are only successful in the shallower network in the from-scratch setting. Gradient matching further outperforms MetaPoison on CIFAR-10, while faster (see Section 1.5.6), in particular as $K = 24$ for MetaPoison.

Benchmark results on CIFAR-10: To evaluate our results against a wider range of poison attacks, we consider the recent benchmark proposed in Schwarzschild et al. [140] in Table 1.3. In the category "Training From Scratch", this benchmark evaluates poisoned CIFAR-10 datasets with a budget of 1% and $\varepsilon = 8$ against various model architectures, averaged over 100 fixed scenarios. We find that the discussed gradient matching attack, even for $K = 1$ is significantly more potent in the more difficult benchmark setting. An additional feature of the benchmark is *transferability*. Poisons are created using a ResNet-18 model, but evaluated also on two other architectures. We find that the proposed attack transfers to the similar MobileNet-V2 architecture, but not as well to VGG11. However, we also show that this advantage can be easily circumvented by using an ensemble of different models as in Zhu et al. [178]. If we use an ensemble of $K = 6$, consisting of 2 ResNet-18, 2 MobileNet-V2 and 2 VGG11 models (last row), then the same poisoned dataset can compromise all models and generalize across architectures.

Table 1.3: Results on the benchmark of [140]. Avg. accuracy of poisoned CIFAR-10 (budget 1%, $\varepsilon = 8$) over 100 trials is shown. (*) denotes rows replicated from [140]. Poisons are created with a ResNet-18 except for the last row, where the ensemble consists of two models of each architecture.

Attack	ResNet-18	MobileNet-V2	VGG11	Average
Poison Frogs* [141]	0%	1%	3%	1.33%
Convex Polytopes* [178]	0%	1%	1%	0.67%
Clean-Label Backd.* [160]	0%	1%	2%	1.00%
Hidden-Trigger Backd.* [138]	0%	4%	1%	2.67%
Proposed Attack ($K = 1$)	45%	36%	8%	29.67%
Proposed Attack ($K = 4$)	$\mathbf{55\%}$	37%	7%	33.00%
Proposed Attack ($K = 6$, Heterogeneous)	49%	$\mathbf{38\%}$	$\mathbf{35\%}$	$\mathbf{40.67\%}$

1.5.2 Poisoning ImageNet models

The ILSVRC2012 challenge, "ImageNet", consists of over 1 million training examples, making it infeasible for most actors to train large model ensembles or run extensive hyperparameter optimizations. However, as the new gradient matching attack requires only a single sample of pretrained parameters θ, and operates only on the poisoned subset, it can poison ImageNet images using publicly available pretrained models without ever training an ImageNet classifier. Poisoning ImageNet with previous methods would be infeasible. For example, following the calculations in Section 1.2, it would take over 500 GPU days (relative to our hardware) to create a poisoned ImageNet for a ResNet-18 via MetaPoison. In contrast, the new attack can poison ImageNet in less than four GPU hours.

Figure 1.4 shows that a standard ImageNet models trained from scratch on a poisoned dataset "brewed" with the discussed attack, are reliably compromised - with examples of successful poisons shown (left). We first study the effect of varying poison budgets, and ε-bounds (top right). Even at a budget of 0.05% and ε-bound of 8, the attack poisons a

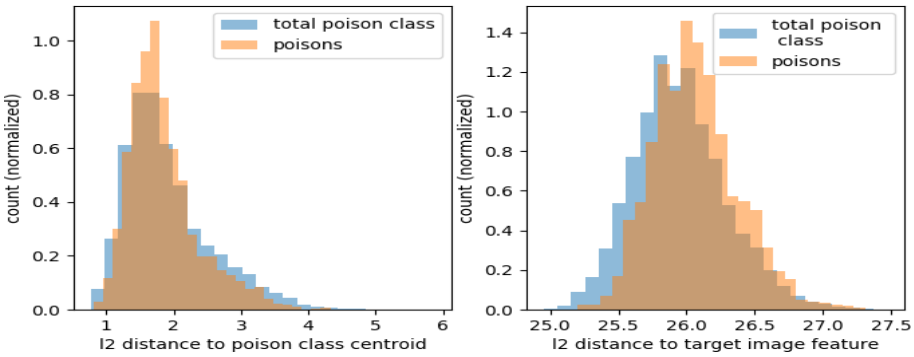
Figure 1.4: Poisoning ImageNet. **Left**: Clean images (above), with their poisoned counterparts (below) from a successful poisoning of a randomly initialized ResNet-18 trained on ImageNet for a poison budget of 0.1% and an ℓ_∞ bound of $\varepsilon = 8$. **Right Top**: ResNet-18 results for different budgets and varying ε-bounds. **Right Bot.**: More architectures [59, 139, 145] with a budget of 0.1% and $\varepsilon = 16$.

randomly initialized ResNet-18 80% of the time. These results extend to other popular models, such as MobileNet-v2 and ResNet50 (bottom right).

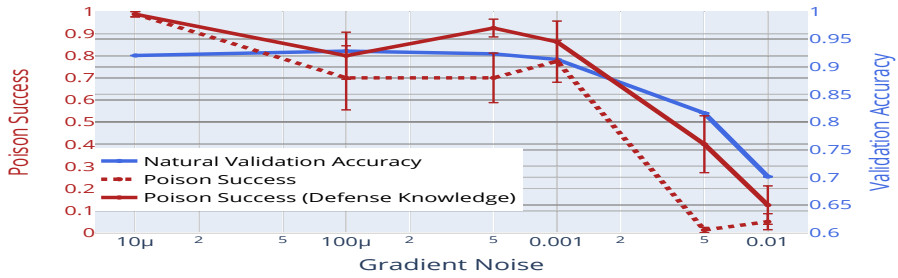
Poisoning Cloud AutoML: To verify that the discussed attack can compromise models in practically relevant *black-box setting*, we test against Google's Cloud AutoML. This is a cloud framework that provides access to black-box ML models based on an uploaded dataset. In Huang et al. [68] Cloud AutoML was shown to be vulnerable for CIFAR-10. We upload a poisoned ImageNet dataset (base: ResNet18, budget 0.1%, $\varepsilon = 32$) for our first poison-target test case and upload the dataset. Even in this scenario, the attack is measurably effective, moving the adversarial label into the top-5 predictions of the model in 5 out of 5 runs, and the top-1 prediction in 1 out of 5 runs.

1.5.3 Deficiencies of Defense Strategies

Previous defenses against data poisoning [123, 125, 149] have relied mainly on data sanitization, i.e. trying to find and remove poisons by outlier detection (often in feature



(a) Feature space distance to base class centroid, and target image feature, for victim model on CIFAR-10. 4.0% budget, $\varepsilon = 16$, showing sanitization defenses failing and no feature collision as in Poison Frogs.



(b) Defending through differential privacy. CIFAR-10, 1% budget, $\varepsilon = 16$, ResNet-18. Differential privacy is only able to limit the success of poisoning via trade-off with significant drops in accuracy.

Figure 1.5: Defense strategies against poisoning.

space). We demonstrate why sanitization methods fail in the face of the attack discussed in this work in Figure 1.5a. Poisoned data points are distributed like clean data points, reducing filtering based methods to almost-random guessing (see Table 1.4).

Differentially private training is a different defense. It diminishes the impact of individual training samples, in turn making poisoned data less effective [62, 98]. However, this come at a significant cost. Figure 1.5b shows that to push the Poison Success below 15%, one has to sacrifice over 20% validation accuracy, even on CIFAR-10. Training a diff. private ImageNet model is even more challenging. From this aspect, differentially private training can be compared to adversarial training [100] against evasion attacks.

Both methods can mitigate the effectiveness of an adversarial attack, but only by significantly impeding natural accuracy.

1.5.4 Deficiencies of Filtering Defenses

Defenses aim to sanitize training data of poisons by detecting outliers (often in feature space), and removing or relabeling these points [123, 125, 149]. In some cases, these defenses are in the setting of general performance degrading attacks, while others deal with targeted attacks. By in large, poison defenses up to this point are limited in scope. For example, many defenses that have been proposed are specific to simple models like linear classifiers and SVM, or the defenses are tailored to weaker attacks such as collision based attacks where feature space is well understood [123, 125, 149]. However, data sanitization defenses break when faced with stronger attacks. Table 1.4 shows a defense by anomaly filtering. averaged over 6 randomly seeded poisoning runs on CIFAR-10 (4% budget w/ $\varepsilon = 16$), we find that outlier detection is only marginally more successful than random guessing.

Table 1.4: Outlier detection is close to random-guessing for poison detection on CIFAR-10.

	10% filtering	20% filtering
Expected poisons removed (outlier method)	248	467
Expected clean removed (outlier method)	252	533
Expected poisons removed (random guessing)	200	400
Expected clean removed (random guessing)	300	600

1.5.5 Details: Defense by Differential Privacy

In Figure 1.5b we consider a defense by differential privacy. According to Hong et al. [62], gradient noise is the key factor that makes differentially private SGD [1] useful as a defense. As such we keep the gradient clipping fixed to a value of 1 and only increase the gradient noise in Figure 1.5b. To scale differentially private SGD, we only consider this gradient clipping on the mini-batch level, not the example level. This is reflected in the red, dashed line. A trivial counter-measure against this defense is shown as the solid red line. If the level of gradient noise is known to the attacker, then the attacker can brew poisoned data by the approach shown in Algorithm 1, but also add gradient noise and gradient clipping to the poison gradient. We use a naive strategy of redrawing the added noise every time the matching objective $\mathcal{B}(\Delta, \theta)$ is evaluated. It turns out that this yields a good baseline counter-attack against the defense through differential privacy.

1.5.6 Full-scale MetaPoison Comparisons on CIFAR-10

Removing all constraints for time and memory, we visualize time/accuracy of our approach against other poisoning approaches in Table 1.6. Note that attacks, like MetaPoison, which succeed on CIFAR-10 only after removing these constraints, cannot be used on ImageNet-sized datasets due to the significant computational effort required. For MetaPoison, we use the original implementation of Huang et al. [68], but add our larger models. We find that with the larger architectures and different threat model (original MetaPoison considers a color perturbation in addition to the ℓ^∞ bound), our gradient matching technique still significantly outperforms MetaPoison. Note that for the Con-

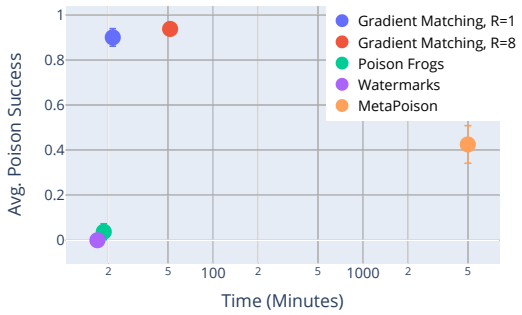
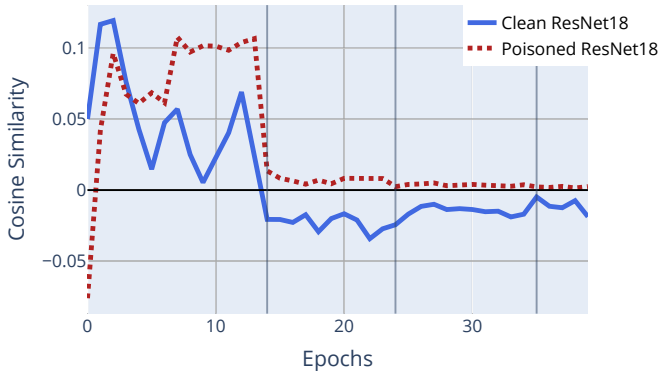


Figure 1.6: CIFAR-10 comparison without time and memory constraints for a ResNet18 with realistic training. Budget 1%, $\varepsilon = 16$. Note that the x-axis is logarithmic.

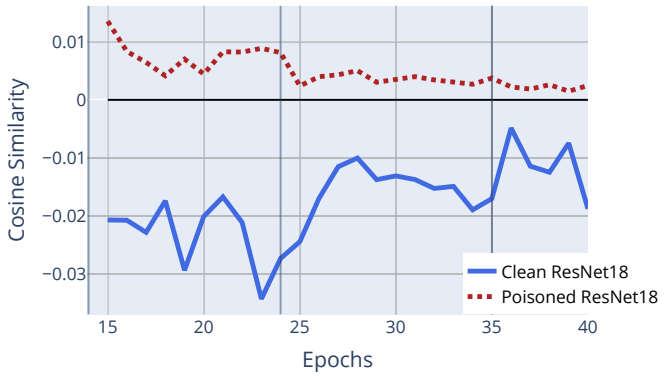
vNet experiment on MetaPoison in Table 1.2, we found that MetaPoison seems to overfit with $\varepsilon = 32$, and as such we show numbers running the MetaPoison code with $\varepsilon = 16$ in that column, which are about 8% better than $\varepsilon = 16$. This is possibly a hyperparameter question for MetaPoison, which was optimized for $\varepsilon = 8$ and a color perturbation.

1.5.7 Details: Gradient Alignment Visualization

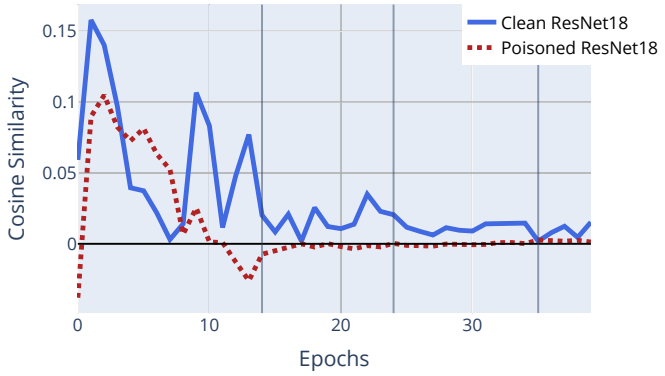
Figure 1.7 visualizes additional details regarding Figure 1.3b. Figure 1.7a replicates Figure 1.3b with linear scaling, whereas Figure 1.7b shows the behavior after epoch 14, which is the first learning rate drop. Note that in all figures each measurement is averaged over an epoch and the learning rate drops are marked with gray vertical bars. Figure 1.7c shows the opposite metric, that is the alignment of the original (non-adversarial) gradient. It is important to note for these figures, that the positive alignment is the crucial, whereas the magnitude of alignment is not as important. As this is the gradient averaged over the entire epoch, the contributions are from mini-batches can contain none or only a single poisoned example.



(a) Alignment of $\nabla\mathcal{L}_{\text{adv}}(\theta)$ and $\nabla\mathcal{L}(\theta)$



(b) Zoom: Alignment of $\nabla\mathcal{L}_{\text{adv}}(\theta)$ and $\nabla\mathcal{L}(\theta)$ from epoch 14.



(c) Alignment of $\nabla\mathcal{L}_{\text{t}}(\theta)$ (orig. label) and $\nabla\mathcal{L}(\theta)$

Figure 1.7: Average batch cosine similarity, per epoch, between the adversarial gradient and the gradient of each mini-batch (left), and with its clean counterpart $\nabla\mathcal{L}_{\text{t}}(\theta) := \nabla_\theta\mathcal{L}(x^t, y^t)$ (right) for a poisoned and a clean ResNet-18. Each measurement is averaged over an epoch. Learning rate drops are marked with gray vertical bars.

1.5.8 Ablation Studies - Reduced Brewing/Victim Training Data

In order to further test the strength and possible limitations of the discussed poisoning method, we perform several ablation studies, where we reduce either the training set known to the attacker or the set of poisons used by the victim, or both.

In many real world poisoning situations, it is not reasonable to assume that the victim will unwittingly add all poison examples to their training set, or that the attacker knows the full victim training set to begin with. For example, if the attacker puts 1000 poisoned images on social media, the victim might only scrape 300 of these. We test how dependent the method is on the victim training set by randomly removing a proportion of data (clean + poisoned) from the victim's training set. We then train the victim on the ablated poisoned dataset, and evaluate the target image to see if it is misclassified by the victim as the attacker's intended class. Then, we add another assumption - the brewing network does not have access to all victim training data when creating the poisons (see tab 1.5). We see that the attacker can still successfully poison the victim, even after a large portion of the victim's training data is removed, or the attacker does not have access to the full victim training set.

Table 1.5: Average poisoning success under victim training data ablation. In the first regime, victim ablation, a proportion of the victim's training data (clean + poisoned) is selected randomly and then the victim trains on this subset. In the second regime, pretrained + victim ablation, the pretrained network is trained on a randomly selected proportion of the data, and then the victim chose a new random subset of clean + poisoned data on which to train. All results averaged over 5 runs on ImageNet.

	70% data removed	50% data removed
victim ablation	60%	100%
pretrained + victim ablation	60%	80%

1.5.9 Ablation Studies - Method

Table 1.6 shows different variations of the proposed method. While using the Carlini-Wagner loss as a surrogate for cross entropy helped in Huang et al. [68], it does not help in our setting. We further find that running the proposed method for only 50 steps (instead of 250 as everywhere else in the paper) leads to a significant loss in avg. poison success. Lastly we investigate whether using euclidean loss instead of cosine similarity would be beneficial. This would basically imply trying to match eq. (2) directly. Euclidean loss amounts to removing the invariance to gradient magnitude, in comparison to cosine similarity, which is invariant. We find that this is not beneficial in our experiments, and that the invariance with respect to gradient magnitude does allow for the construction of stronger poisoned datasets. Interestingly the discrepancy between both loss functions is related to the width of the network. In Figure 1.8 on the left, we visualize avg. poison success for modified ResNet-18s. The usual base width of 64 is replaced by the width value shown on the x-axis. For widths smaller than 16, the Euclidean loss dominates, but its effectiveness does not increase with width. In contrast the cosine similarity is superior for larger widths and seems to be able to make use of the greater representative power of the wider networks to find vulnerabilities. Figure 1.8 on the right examines the impact of the pretrained model that is supplied to Algorithm 1. We compare avg. poison success against the number of pretraining epochs for a budget of 1%, first with $\varepsilon = 16$ and then with $\varepsilon = 8$. It turns out that for the easier threat model of $\varepsilon = 8$, even pretraining to only 20 epochs can be enough for the algorithm to work well, whereas in the more difficult scenario of $\varepsilon = 8$, performance increases with pretraining effort.

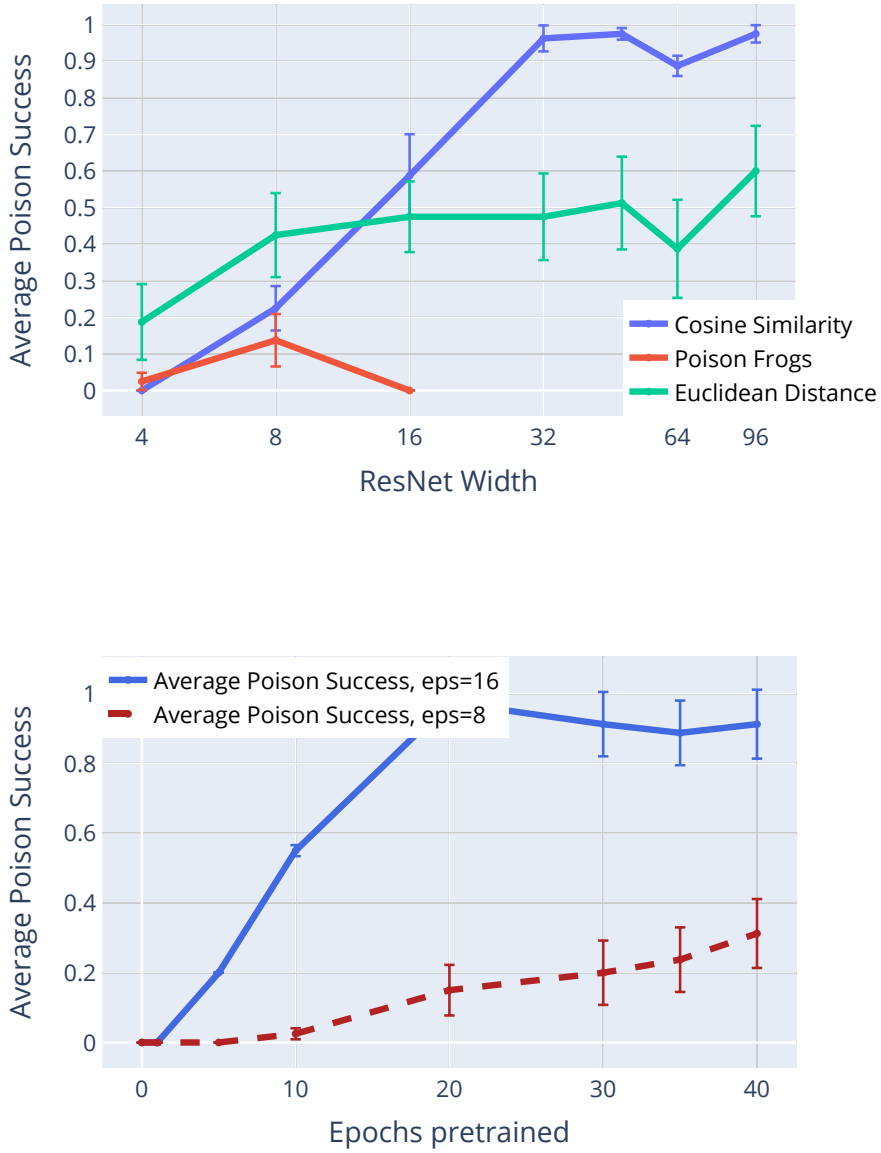


Figure 1.8: Ablation Studies. Left: avg. poison success for Euclidean Loss, cosine similarity and the Poison Frogs objective [141] for thin ResNet-18 variants. Right: Avg. poison success vs number of pretraining epochs.

Table 1.6: CIFAR-10 ablation runs. $\varepsilon = 16$, budget is 1%. All values are computed for ResNet-18 models.

Setup	Avg. Poison Success $\%(\pm$SE)	Validation Acc.%
Baseline (full data aug., $R = 8$, $M = 250$	91.25% ($\pm$6.14)	92.20%
Carlini-Wagner loss instead of $\mathcal{L}$	77.50% ($\pm$9.32)	92.08%
Fewer Opt. Steps ($M = 50$)	40.00% ($\pm$10.87)	92.05%
Euclidean Loss instead of cosine sim.	61.25% ($\pm$9.75)	92.09%

Table 1.7: CIFAR-10 ablation runs. $\varepsilon = 16$, budget is 1%. All values are computed for ResNet-18 models. Averaged over 5 runs.

Setup	Avg. Poison Success ($\%$ of total targets poisoned successfully)	Effective Budget / Target
1 target (Baseline)	90.00%	1%
5 targets	32.00%	0.2%
10 targets	14.00%	0.1%

1.5.10 Transfer Experiments

In addition to the fully black-box pipeline of the AutoML experiments in Section 1.7, we test the transferability of our poisoning method against other commonly used architectures. Transfer results on CIFAR-10 can be found in Table 1.3. On Imagenet, we brew poisons with a variety of networks, and test against other networks. We find that poisons crafted with one architecture can transfer and cause targeted mis-classification in other networks (see Figure 1.9).

1.5.11 Multi-Target Experiments

We also perform limited tests on poisoning multiple targets simultaneously. We find that while keeping the small poison budget of 1% fixed, we are able to successfully poison more than one target while optimizing poisons simultaneously, see Table 1.7. Effectively, however, every target image gradient has to be matched with an increasingly smaller

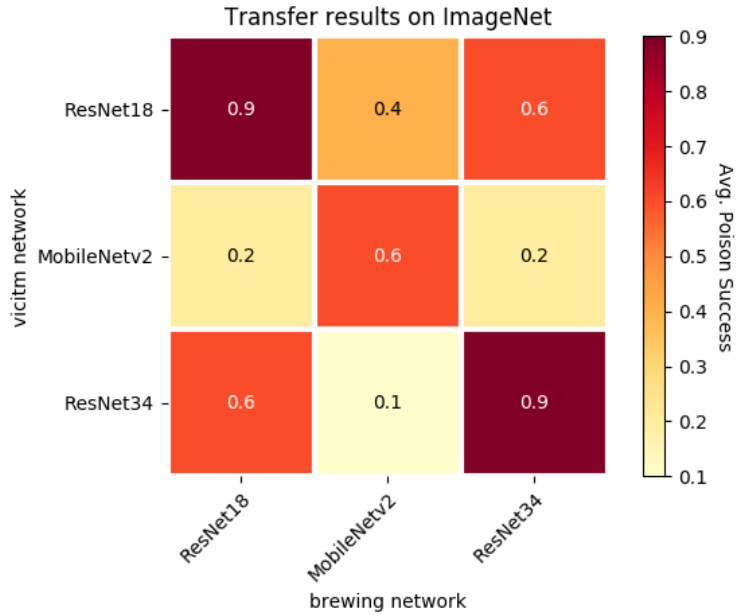


Figure 1.9: Direct transfer results on common architectures. Averaged over 10 runs with budget of 0.1% and ε-bound of 16. Note that for these transfer experiments, the model was *only* trained on the "brewing" network, without knowledge of the victim. This shows a transferability to unknown architectures.

budget. As the target images are drawn at random and not semantically similar (aside from their shared class), their synergy is limited - for example the 5 targets experiment reaches an accuracy of 32%, which is only 14% better than the naive baseline of $\frac{90\%}{5}$ one might expect. One encouraging result from the standpoint of poisoning though is that the multiple targets do not compete against each other, canceling out their respective different alignments.

1.6 Visualizations

We visualize poisoned sample from our ImageNet runs in Figures 1.13, 1.10, noting especially the "clean label" effect. Poisoned data is only barely distinguishable from clean data, even in the given setting where the clean data is shown to the observer. In a realistic

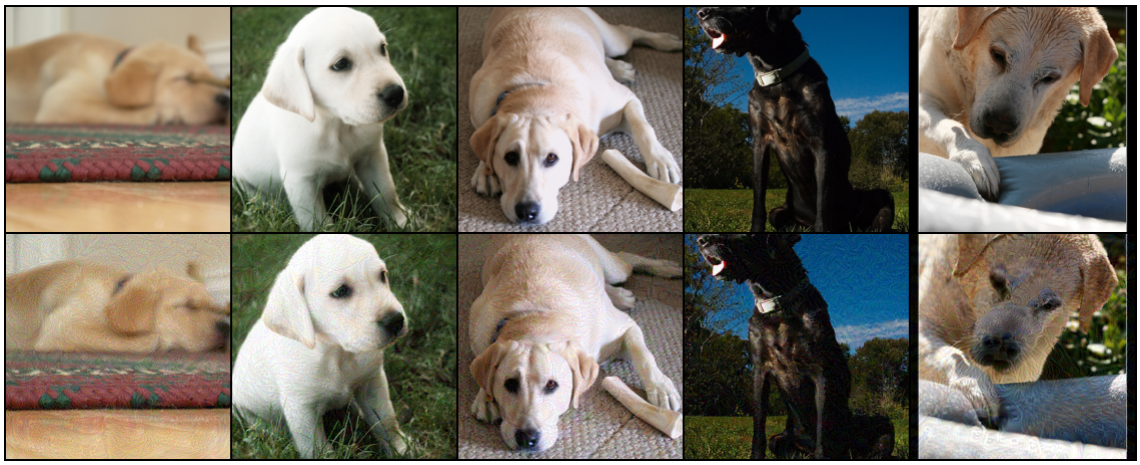


Figure 1.10: Clean images (above), with their poisoned counterparts (below) from a successful poisoning of a randomly initialized ResNet-18 trained on ImageNet. The poisoned images (taken from the Labrador Retriever class) successfully caused mis-classification of a target (otter) image under a threat model given by a budget of 0.1% and an ℓ_∞ bound of $\epsilon = 16$.

setting, this is significantly harder. A subset of poisoned images used to poison Cloud autoML with $\varepsilon = 32$ can be found in Figure 1.11.

We concentrate only on small ℓ^∞ perturbations to the training data as this is the most common setting for adversarial attacks. However, there exist other choices for attacks in practical settings. Previous works have already considered additional color transformations [68] or watermarks [141]. Most techniques that create adversarial attacks at test time within various constraints [35, 61, 76, 173] are likely to transfer into the data poisoning setting. Likewise, we do not consider hiding poisoned images further by minimizing perceptual scores and relate to the large literature of adversarial attacks that evade detection [19].

In Figure 1.12 we visualize how the adversarial loss and accuracy behave during an exemplary training run, comparing the adversarial label with the original label of the target image.

Figure 1.11: Clean images (above), with their poisoned counterparts (below) from a successful poisoning of a Google Cloud AutoML model trained on ImageNet. The poisoned images (taken from the Labrador Retriever class) successfully caused mis-classification of a target (otter) image. This is accomplished with a poison budget of 0.1% and an ℓ_∞ bound of $\varepsilon = 32$.

1.7 Experimental Setup

This section details our experimental setup for replication purposes. A central question in the context of evaluating data poisoning methods is how to judge and evaluate "average" performance. Poisoning is in general volatile with respect to poison-target class pair, and to the specific target example, with some combinations and target images being in general easier to poison than others. However, evaluating all possible combinations is infeasible for all but the simplest datasets, given that poisoned data has to created for each example and then a neural network has to be trained from scratch every time. Previous works [141, 178] have considered select target pairs, e.g. "birds-dogs" and "airplanes-frogs", but this runs the risk of mis-estimating the overall success rates. Another source of variability arises, especially in the from-scratch setting: Due to both the randomness of the initialization of the neural network, the randomness of the order in which images

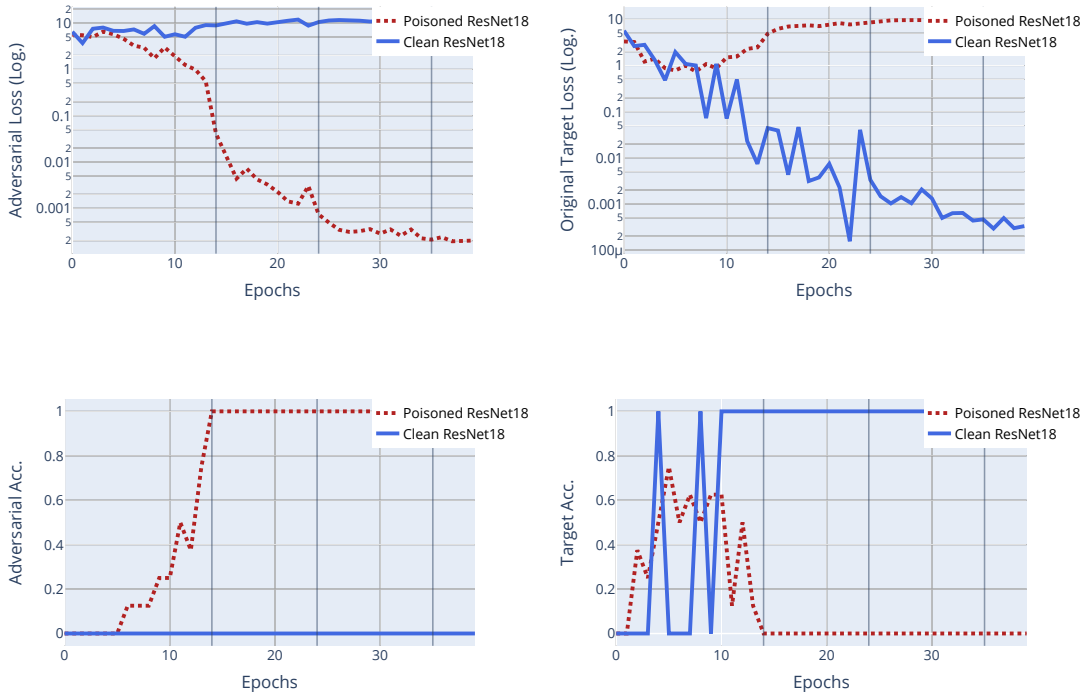


Figure 1.12: Cross entropy loss (Top) and accuracy (Bottom) for a given target with its adversarial label (left), and with its original label (right) shown for a poisoned and a clean ResNet-18. The clean model is used as victim for the poisoned model. The loss is averaged 8 times for the poisoned model. Learning rate drops are marked with gray horizontal bars.

are drawn during mini-batch SGD, and the randomness of data augmentations, a fixed poisoned dataset might only be effective some of the time, when evaluating it multiple times.

In light of this discussion, we adopt the following methodology: For every experiment we randomly select n (usually 10 in our case) settings consisting of a random target class, random poison class, a random target and random images to be poisoned. For each of these experiments we create a single poisoned dataset by the discussed or a comparing method within limits of the given threat model and then evaluate the poisoned datasets m times (8 for CIFAR-10 and 1 for ImageNet) on random re-initializations of the con-

sidered architecture. To reduce randomness for a fair comparison between different runs of this setup, we fix the random seeds governing the experiment and rerun different threat models or methods with the same random seeds. We have used CIFAR-10 with random seeds 1000000000-1111111111 hyperparameter tuning and now evaluate on random seeds 2000000000-2111111111 for CIFAR-10 experiments and 1000000000-1111111111 for ImageNet, with class pairs and target image IDs for reproduction given in Tables 1.8 1.9. For CIFAR-10, the target ID refers to the canonical order of all images in the dataset (as downloaded from `https://www.cs.toronto.edu/~kriz/cifar.html`); for ImageNet, the ID refers to an order of ImageNet images where the syn-sets are ordered by their increasing numerical value (as is the default in `torchvision`). However for future research we encourage the sampling of new target-poison pairs to prevent overfitting, ideally even in larger numbers given enough compute power.

For every measurement of *avg. poison success* in the paper, we measure in the following way: After retraining the given deep neural network to completion, we measure if the target image is successfully classified by the network as its adversarial class. We do not count mere misclassification of the original label (but note that this usually happens even before the target is incorrectly classified by the adversarial class). Over the m validation runs we repeat this measurement of target classification success and then compute the average success rate for a single example. We then aggregate this average over our 10 chosen random experiments and report the mean and standard error of these average success rates as *avg. poison success*. All error bars in the paper refer to standard error of these measurements.

Table 1.8: Target/poison class pairs generated from the initial random seeds for ImageNet experiments. Target ID relative to CIFAR-10 validation dataset.

Target Class	Poison Class	Target ID	Random Seed
dog	frog	8745	2000000000
frog	truck	1565	2100000000
frog	bird	2138	2110000000
airplane	dog	5036	2111000000
airplane	ship	1183	2111100000
cat	airplane	7352	2111110000
automobile	frog	3544	2111111000
truck	cat	3676	2111111100
automobile	ship	9882	2111111110
automobile	cat	3028	2111111111

Table 1.9: Target/poison class pairs generated from the initial random seeds for ImageNet experiments. Target Id relative to ILSVRC2012 validation dataset [137]

Target Class	Poison Class	Target ID	Random Seed
otter	Labrador retriever	18047	1000000000
warthog	bib	17181	1100000000
orange	radiator	37530	1110000000
theater curtain	maillot	42720	1111000000
hartebeest	capuchin	17580	1111100000
burrito	plunger	48273	1111110000
jackfruit	spider web	47776	1111111000
king snake	hyena	2810	1111111100
flat-coated retriever	alp	10281	1111111110
window screen	hard disc	45236	1111111111

1.7.1 Cloud AutoML Setup

For the experiment using Google's cloud autoML, we upload a poisoned ILSVRC2012 dataset into google storage, and then use `https://cloud.google.com/vision/automl/` to train a classification model. Due to autoML limitations to 1 million images, we only upload up to 950 examples from each class (reaching a training set size slightly smaller than $950\,000$, which allows for an upload of the $50\,000$ validation images). We use a ResNet-18 model as surrogate for the black-box learning within autoML,

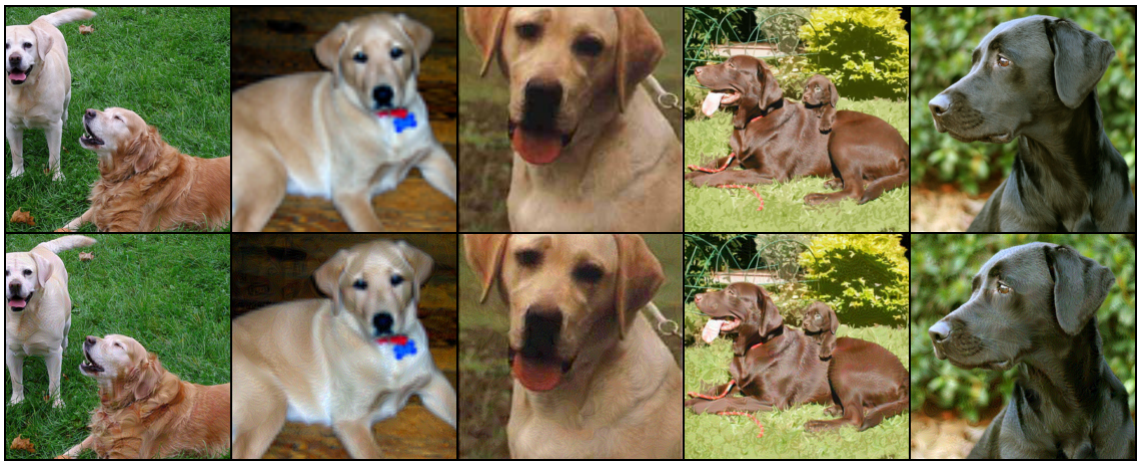


Figure 1.13: Clean images (above), with their poisoned counterparts (below) from a successful poisoning of a Google Cloud AutoML model trained on ImageNet. The poisoned images (taken from the Labrador Retriever class) successfully caused mis-classification of a target (otter) image under a threat model given by a budget 0.1% and an ℓ_∞ bound of $\varepsilon = 32$.

pretrained on the full ILSVRC2012 as before. We create a `MULTICLASS` autoML dataset and specify the vision model to be `mobile-high-accuracy-1` which we train to $10\,000$ milli-node hours, five times. After training the model, we evaluate its performance on the validation set and target image. The trained models all reach a 69% clean top-1 accuracy on the ILSVRC2012 validation set.

1.7.2 Hardware

We use a heterogeneous mixture of hardware for our experiments. CIFAR-10, and a majority of the ImageNet experiments, were run on NVIDIA GEFORCE RTX 2080 Ti gpus. CIFAR-10 experiments were run on 1 gpu, while ImageNet experiments were run on 4 gpus. We also use NVIDIA Tesla P100 gpus for some ImageNet experiments. All timed experiments were run using 2080 Ti gpus.

1.7.3 Models

For our experiments on CIFAR-10 in section 5 we consider two models. In table 2, the "6-layer ConvNet", - in close association with similar models used in Finn et al. [41] or Krizhevsky et al. [87], we consider an architecture of 5 convolutional layers (with kernel size 3), followed by a linear layer. All convolutional layers are followed by a ReLU activation. The last two convolutional layers are followed by max pooling with size 3. The output widths of these layers are given by $64, 128, 128, 256, 256, 2304$. In tables 1, 2, in the inset figure and Fig. 4 we consider a ResNet-18 model. We make the customary changes to the model architecture for CIFAR-10, replacing the stem of the original model (which requires ImageNet-sized images) by a convolutional layer of size 3, following by batch normalization and a ReLU. This is effectively equal to upsampling the CIFAR-10 images before feeding them into the model. For experiments on ImageNet, we consider ResNet-18, ResNet-34 [59], MobileNet-v2 [139] and VGG-16 [145] in standard configuration.

We train the ConvNet, MobileNet-v2 and VGG-16 with initial learning rate of 0.01 and the residual architectures with initial learning rate 0.1. We train for 40 epochs, dropping the learning rate by a factor of 10 at epochs 14, 24, 35. We train with stochastic mini-batch gradient descent with Nesterov momentum, with batch size 128 and momentum 0.9. Note that the dataset is shuffled in each epoch, so that where poisoned images appear in mini-batches is random and not known to the attacker. We add weight decay with parameter 5×10^{-4}. For CIFAR-10 we add data augmentations using horizontal flipping with probability 0.5 and random crops of size 32×32 with zero-padding of 4. For ImageNet

we resize all images to 256×256 and crop to the central 224×224 pixels. We also consider horizontal flipping with probability 0.5, and data augmentation with random crops of size 224×224 with zero-padding of 28.

When evaluating ImageNet poisoning from-scratch we use the described procedure. To create our poisoned datasets as detailed in Alg. 1, we download the respective pre-trained model from `torchvision`, see `https://pytorch.org/docs/stable/torchvision/models.html`.

1.8 Remarks

Remark (Validating the approach in a special case). Inner-product loss functions like Equation 1.3 work well in other contexts. In [45], cosine similarity between image gradients was minimized to uncover training images used in federated learning. If we disable our constraints, setting $\varepsilon = 255$, and consider a single poison image and a single target, then we minimize the problem of recovering image data from a normalized gradient as a special case. In [45], it was shown that minimizing this problem can recover the target image. This means that we can indeed return to the motivating case in the unconstrained setting - the optimal choice of poison data is insertion of the target image in an unconstrained setting for one image.

Remark (Transfer of gradient alignment). An analysis of how gradient alignment often transfers between different parameters and even between architectures has been conducted, e.g. in [22, 82] and [29]. It was shown in [29] that the performance loss when transferring an evasion attack to another model is governed by the gradient alignment of

both models. In the same vein, optimizing alignment appears to be a useful metric in the case of data poisoning. Furthermore [62] note that previous poisoning algorithms might already cause gradient alignment as a side effect, even without explicitly optimizing for it.

Remark (Poisoning is a Credible Threat to Deep Neural Networks). It is important to understand the security impacts of using unverified data sources for deep network training. Data poisoning attacks up to this point have been limited in scope. Such attacks focus on limited settings such as poisoning SVMs, attacking transfer learning models, or attacking toy architectures [10, 111, 141]. We demonstrate that data poisoning poses a threat to large-scale systems as well. The approach discussed in this work pertains only to the classification scenario, as a guinea pig for data poisoning, but applications to a variety of scenarios of practical interest have been considered in the literature, for example spam detectors mis-classifying a spam email as benign, or poisoning a face unlock based mobile security systems.

The central message of thedata poisoning literature can be described as follows: From a security perspective, the data that is used to train a machine learning model should be under the same scrutiny as the model itself. These models can only be secure if the entire data processing pipeline is secure. This issue further cannot easily be solved by human supervision (due to the existence of clean-label attacks) or outlier detection (see Figure 1.5a). Furthermore, targeted poisoning is difficult to detect as validation accuracy is unaffected. As such, data poisoning is best mitigated by fully securing the data pipeline.

So far we have considered data poisoning from the industrial side. From the per-

spective of a user, or individual under surveillance, however, data poisoning can be a means of securing personal data shared on the internet, making it unusable for automated ML systems. For this setting, we especially refer to an interesting application study in [142] in the context of facial recognition.

1.9 Conclusion

We investigate data poisoning via gradient matching and discover that this mechanism allows for data poisoning attacks against fully retrained models that are unprecedented in scale and effectiveness. We motivate the attack theoretically and empirically, discuss additional mechanisms like differentiable data augmentation and experimentally investigate modern deep neural networks in realistic training scenarios, showing that gradient matching attacks compromise even models trained on ImageNet. We close with discussing the limitations of current defense strategies.

Chapter 2: Availability Poisoning Attacks

The adversarial machine learning literature is largely partitioned into evasion attacks on testing data and poisoning attacks on training data. In this work, we show that adversarial examples, originally intended for attacking pre-trained models, are even more effective for data poisoning than recent methods designed specifically for poisoning. Our findings indicate that adversarial examples, when assigned the original label of their natural base image, cannot be used to train a classifier for natural images. Furthermore, when adversarial examples are assigned their adversarial class label, they are useful for training. This suggests that adversarial examples contain useful semantic content, just with the "wrong" labels (according to a network, but not a human). Our method, *adversarial poisoning*, is substantially more effective than existing poisoning methods for secure dataset release, and we release a poisoned version of ImageNet, ImageNet-P, to encourage research into the strength of this form of data obfuscation. This work was performed together with Micah Goldblum, Ping-yeh Chiang, Jonas Geiping, Wojtek Czaja, and Tom Goldstein. My contributions include: jointly conceiving of the targeted poisoning objective (the more powerful objective), performing a majority of experiments in the work, as well as writing a majority of the paper.

2.1 Introduction

Automated dataset scraping has become necessary to satisfy the exploding demands of cutting-edge deep models [14, 17], but the same automation that enables massive performance boosts exposes these models to security vulnerabilities [4, 23]. Recall, *data poisoning* attacks manipulate training data in order to cause the resulting models to misclassify samples during inference [81], while *backdoor attacks* embed exploits which can be triggered by pre-specified input features [25]. In this work, we focus on a flavor of data poisoning known as *availability attacks*, which aim to degrade overall testing performance [6, 9].

Adversarial attacks, on the other hand, focus on manipulating samples at test-time, rather than during training [154]. In this work, we connect adversarial and poisoning attacks by showing that adversarial examples form stronger availability attacks than any existing poisoning method, even though the latter were designed specifically for manipulating training data while adversarial examples were not. We compare our method, *adversarial poisoning*, to existing availability attacks for neural networks, and we exhibit consistent performance boosts (i.e. lower test accuracy). In fact, models trained on adversarial examples may exhibit test-time performance below that of random guessing.

Intuitively, adversarial examples look dramatically different from their natural base images in the eye of neural networks, despite the two looking similar to humans. Models trained only on such perturbed examples are completely unprepared for inference on the clean data. In support of this intuition, we observe that models trained on adversarially perturbed training data often fail to correctly classify the original clean training samples.

But does this phenomenon occur simply because adversarial examples are off the "natural image manifold" or because they actually contain informative features from other classes? Popular belief assumes that adversarial examples live off the natural image manifold, causing a catastrophic mismatch when digested by models trained on only clean data [79, 150, 177]. However, models trained on data with random additive noise (rather than adversarial noise) perform well on noiseless data, suggesting that there may be more to the effects of adversarial examples than simply moving off the manifold (see Table 2.2). We instead find that since adversarial attacks inject features that a model associates with incorrect labels, training on these examples is similar to training on *mislabeled* training data. After re-labeling adversarial examples with the "wrong" prediction of the network on which they were crafted, models trained on such label-flipped data perform substantially better than models trained on uncorrected adversarial examples and almost as well as models trained on clean images. While this label-correction is infeasible for a practitioner defending against adversarial poisoning, since it assumes possession of the crafting network which requires access to the clean dataset, this experiment strengthens the intuition that adversarial examples contain a strong training signal just from the "wrong" class.

2.2 Related Work

Data poisoning can generally be phrased as a bilevel optimization problem which minimizes loss with respect to parameters in the inner problem while maximizing some attack loss with respect to inputs in the outer problem [9, 67]. Poisoning comes in several

flavors, including *integrity attacks* and *availability attacks*. The former aims to cause targeted misclassification on a small number of pre-selected datapoints, while the latter aims to degrade the overall performance (generalization ability) of a victim network [6]. Classical approaches to poisoining attacks often focused on simple models, where the inner problem can sometimes be solved exactly [9, 21, 105, 167]. However, on neural networks, obtaining exact solutions is intractable. In this setting, Muñoz-González et al. [112] approximates a solution to the inner problem using a small number of descent steps, but the authors note that this method is ineffective against deep neural networks.

Modern poisoning attacks adopt new methods and approximations, like gradient alignment [47, 148], computation graph unrolling [68], etc., but for integrity attacks on deep networks. On the availability attack side, still other heuristics have been adopted. For example, gradient alignment [47] with a modified indiscriminate objective was used in Fowl et al. [42], while gradient explosion was suggested in Shen et al. [143]. Other availability attacks have used auto-encoder generated perturbations [39], as well as loss minimization objectives [65]. Notably, it was previously believed that adversarial (loss maximization) objectives were *not* suitable to availability attacks [65].

Still other related poisoning works harness *influence functions* which estimate the impact of each training sample on a resulting model [38, 81, 83]. However, influence functions are brittle on deep networks whose loss surfaces are highly irregular [7]. A general overview of data poisoning methods can be found in [52].

Adversarial examples. Adversarial attacks probe the blindspots of trained models where they catastrophically misclassify inputs that have undergone small perturbations [154]. Prototypical algorithms for adversarial attacks simply maximize loss with respect

to the input while constraining perturbations. The resulting adversarial examples exploit the fact that as inputs are even slightly perturbed in just the right direction, their corresponding deep features and logits change dramatically, and gradient-based optimizers can efficiently find these directions. The literature contains a wide array of proposed loss functions and optimizers for improving the effectiveness of attacks [18, 57]. A number of works suggest that adversarial examples are off the image manifold, and others propose methods for producing on-manifold attacks [79, 150, 177].

Adversarial training. The most popular method for producing neural networks that are robust to attacks involves crafting adversarial versions of each mini-batch and training on these versions [99]. On the surface, it might sound as if adversarial training is very similar to training on poisons crafted via adversarial attacks. After all, they both involve training on adversarial examples. However, adversarial training ensures that the robust model classifies inputs correctly within a ball surrounding each training sample. This is accomplished by updating perturbations to inputs *throughout* training. This process desensitizes the adversarially trained model to small perturbations to its inputs. In contrast, a model trained on adversarially poisoned data is only encouraged to fit the exact, fixed perturbed data.

2.3 Adversarial Examples as Poisons

In this section, we describe the central mechanism for crafting availability attack. We formally introduce the objective of the attacker, and describe our approach, and compare to several existing methods.

2.3.1 Threat Model and Motivation

We introduce two parties: the *poisoner* (sometimes called the attacker), and the *victim*. The poisoner has the ability to perturb the victim's training data but does not know the victim's model initialization, training routine, or architecture. The victim then trains a new model from scratch on the poisoned data. The poisoner's success is determined by the accuracy of the victim model on clean data.

Early availability attacks that worked best in simple settings, like SVMs, often modified only a small portion of the training data. However, recent availability attacks that work in more complex settings have instead focused on applications such as secure data release where the poisoner has access to, and modifies, *all* data used by the victim [39, 42, 65, 143].

These methods manipulate the entire training set to cause poor generalization in deep learning models trained on the poisoned data. This setting is relevant to practitioners such as social media companies who wish to maintain the competitive advantage afforded to them by access to large amounts of user data, while also protecting user privacy by making scraped data useless for training models. Practically speaking, companies could employ methods in this domain to imperceptibly modify user data *before* dissemination through social media sites in order to degrade performance of any model which is trained on this disseminated data.

To compare our method to recent works, we focus our experiments in this setting where the poisoner can perturb the entire training set. However, we also poison lower proportions of the data in Tables 2.5, 2.4. We find that on both simple and complex

datasets, our method produces poisons which are useless for training, and models trained on data including poisons would have performed just as well had they identified and thrown out the poisoned data altogether.

2.3.2 Problem Setup

Formally stated, availability poisoning attacks aim to solve the following bi-level objective in terms of perturbations $\Delta = \{\Delta_i\}$ to elements x_i of a dataset $\mathcal{T}$:

$$\max_{\Delta \in \mathcal{C}} \mathbb{E}_{(x,y)\sim\mathcal{D}} \Big[\mathcal{L}\left(F(x; \theta(\Delta)), y\right) \Big] \tag{2.1}$$

$$\text{s.t. } \theta(\Delta) \in \arg\min_{\theta} \sum_{(x_i,y_i)\in\mathcal{T}} \mathcal{L}(F(x_i + \Delta_i; \theta), y_i), \tag{2.2}$$

where $\mathcal{C}$ denotes the constraint set of the perturbations, and $\mathcal{D}$ denotes the distribution from which $\mathcal{T}$ was drawn. As is common in both the adversarial attack and poisoning literature, we employ an ℓ_∞ bound on each δ_i. Unless otherwise stated, our attacks are bounded by ℓ_∞-norm $\epsilon = 8/255$ as is standard practice on CIFAR-10, and ImageNet data in both adversarial and poisoning literature [47, 99]. Simply put, the attacker wishes to cause a network, F, trained on the poisons to generalize poorly to distribution $\mathcal{D}$ from which $\mathcal{T}$ was sampled.

Directly solving this optimization problem is intractible for neural networks as it requires unrolling the entire training procedure found in the inner objective (Equation (2)) and backpropagating through it to perform a single step of gradient descent on the outer objective. Thus, the attacker must approximate the bilevel objective. Approximations to this objective often involve heuristics, as previously described. For example, TensorClog

[143] aims to cause gradient vanishing in order to disrupt training, while more recent work aims to align poison gradients with an adversarial objective [47].

We opt for an entirely different strategy and instead replace the bi-level problem with *two* empirical *loss maximization* problems - an approach that was believed to be suboptimal for availability poisoning [65]. This turns the poison generation problem into an adversarial example problem. Specifically, we optimize the following untargeted (UT) objective:

$$\max_{\delta \in \mathcal{S}} \left[\sum_{(x_i, y_i) \in \mathcal{T}} \mathcal{L}\left(F(x_i + \delta_i; \theta^*), y_i\right) \right], \tag{2.3}$$

where θ^* denotes the parameters of a model trained on *clean* data, which is fixed during poison generation. We call this model the *crafting* model.

We also optimize an objective which defines a *class targeted* (CT) adversarial attack. This modified objective is defined by:

$$\min_{\delta \in \mathcal{S}} \left[\sum_{(x_i, y_i) \in \mathcal{T}} \mathcal{L}\left(F(x_i + \delta_i; \theta^*), g(y_i)\right) \right], \tag{2.4}$$

where g is a permutation (with no fixed points) on the label space of $\mathcal{S}$. Fittingly, we call our methods *adversarial poisoning*. Note that the class targeted objective was previously (independently) hypothesized to produce potent poisons in Nakkiran [113], and also tested in a work concurrent to ours [158].

Projected Gradient Descent (PGD) has become the standard method for generating adversarial examples for deep networks [99]. Accordingly, we craft our poisons with 250 steps of PGD on this loss-maximization objective. In addition to the adversarial attack introduced in Madry et al. [99], we also experiment with other attacks such as FGSM [55] and Carlini-Wagner [18] in Table 2.2. We find that while other adversaries do produce effective poisons, a PGD based attack is the most effective in generating poisons. Finally, borrowing from recent targeted data poisoning works, we also employ differentiable data augmentation in the crafting stage [47] (see section 2.3.3).

An aspect of note for our method is the ease of crafting perturbations - we use a straightforward adversarial attack on a fixed pretrained network to generate the poisons. This is in contrast to previous works which require pretraining an adversarial auto-encoder [40] (5 - 7 GPU days for simple datasets), or require iteratively updating the model and perturbations [65], which requires access to the entire training set all at once - an assumption that does not hold for practitioners like social media companies who acquire data sequentially. In addition to the performance boosts our method offers, it is also the most flexible compared to each of the availability attacks with which we compare.

2.3.3 Technical Details for Successful Attacks

As we will see in the following sections, adversarial objectives can indeed produce powerful poisons. As previously stated, such loss maximization approaches to availability attacks were thought to be suboptimal [65]. However, we find that differentiable data augmentation during crafting, along with more PGD steps, greatly improves the potency

of the generated poisons. In Figure 2.5, we see that augmentation is very helpful for the untargeted objective at higher number of PGD steps, but is less important for the class-targeted objective.

Furthermore, we find that the variability of the untargeted attack is generally higher than the class-targeted attack. That is, the untargeted attack is more sensitive to random initialization of the poisons. For purposes of comparison, we do modest hyperparameter searching to find a reasonable poisoned dataset on which to evaluate a victim model. We discuss this further in Section 2.4.9.

2.3.4 Baseline CIFAR-10 Results

We first experiment in a relatively simple setting, CIFAR-10 [86], consisting of $50,000$ low-resolution images from 10 classes. All poisons in this setting are tested on a variety of architectures in a setting adapted from a commonly used repository [1]. We find that our poisoning method reliably degrades the test accuracy of a wide variety of popular models including VGG19 [145], ResNet-18 [59], GoogLeNet [153], DenseNet-121 [64], and MobileNetV2 [139]. Even under very tight ℓ_∞ constraints, our poisons more than halve the test accuracy of these models. These results are presented in Table 2.1.

Additionally, we study the effect of the optimizer, data augmentation, number of steps, and crafting network on poison generation in Tables 2.9, 2.10, and 2.11. These tables demonstrate that a wide variety of adversarial attacks with various crafting networks and hyperparameters yield effective poisons. Also note that while the results we present here are for poisons generated with a ResNet18, we find that the poisons remain

[1] `https://github.com/kuangliu/pytorch-cifar`.

Table 2.1: **Comparison of different ε-bounds for our adversarial poisoning method.** All poisons generated by ResNet-18 crafted with 250 steps of PGD, with differentiable data augmentation. CT denotes poisons crafted with the class targeted adversarial attack. Note that we would expect random guessing to achieve 10% validation accuracy.

BOUND \VICTIM	VGG19	RESNET-18	GOOGLENET	DENSENET-121	MOBILENETV2
CLEAN	93.9 ± 0.16	95.53 ± 0.03	95.38 ± 0.11	95.51 ± 0.07	92.42 ± 0.06
$\varepsilon = 4/255$ (UT)	64.71 ± 0.76	56.79 ± 0.75	61.9 ± 0.42	59.10 ± 0.34	47.72 ± 0.23
$\varepsilon = 8/255$ (UT)	10.98 ± 0.27	6.25 ± 0.17	7.03 ± 0.12	7.16 ± 0.16	6.11 ± 0.17
$\varepsilon = 4/255$ (CT)	30.26 ± 0.55	26.49 ± 0.15	29.53 ± 0.51	26.63 ± 0.59	21.64 ± 0.54
$\varepsilon = 8/255$ (CT)	10.32 ± 0.35	8.69 ± 0.44	9.36 ± 0.26	7.85 ± 0.40	8.36 ± 0.31

effective when generated from other network architectures (see Table 2.11).

In this popular setting, we can compare our method to existing availability attacks including TensorClog [143], Loss Minimization [65], and a gradient alignment based method [42]. Our method widely outperforms these previous methods. Compared with the previous best method (loss minimization), we degrade the validation accuracy of a victim network by a factor of more than three.

Table 2.2: **Validation accuracies of models trained on data from different availability attacks.** Tested on randomly initialized ResNet-18 models on CIFAR-10. All crafted with $\varepsilon = 8/255$.

METHOD	VALIDATION ACCURACY (%, ↓)
NONE (CLEAN)	94.56
RANDOM NOISE	90.52
TENSORCLOG [143]	84.24
ALIGNMENT [42]	53.67
UNLEARNABLE EXAMPLES [65]	19.85
DEEPCONFUSE [40]	31.10
ADVERSARIAL POISONING UNTARGETED (OURS)	**11.94**
ADVERSARIAL POISONING CLASS-TARGETED (OURS)	**8.69**

Note that we test our method in a completely black-box setting wherein the poisoner has no knowledge of the victim network's initialization, architecture, learning rate scheduler, optimizer, etc. We find our adversarial poisons transfer across these settings

and reliably degrade the validation accuracy of all the models tested. See Section 2.4.1 for more details about training procedures.

2.3.5 Large Scale Poisoning

In addition to validating our method on CIFAR-10, we also conduct experiments on ImageNet (ILSVRC2012), consisting of over 1 million images coming from 1000 different classes [137]. This setting tests whether adversarial poisons can degrade accuracy on industrial-scale, high resolution datasets. We discover that while untargeted attacks successfully reduce the generalization, our class-targeted attack (CT) degrades clean validation accuracy significantly further (see Table 2.3). Furthermore, even at an almost imperceptible perturbation level of $4/255$, the class targeted poisons cripple the validation accuracy of the victim model to 3.57%. Visualizations of the poisons can be found in Figure 2.1.

Table 2.3: **Validation accuracies of models trained on poisoned ImageNet data**. Tested on randomly initialized ResNet-18 models.

METHOD	VALIDATION ACCURACY ($\%, \downarrow$)
NONE (CLEAN)	66.56
$\varepsilon = 4/255$ (UT)	45.07
$\varepsilon = 8/255$ (UT)	36.63
$\varepsilon = 4/255$ (CT)	3.57
$\varepsilon = 8/255$ (CT)	**1.45**

We hypothesize that the noteable superiority of the class targeted attack in this setting arises from the larger label-space of ImageNet. Specifically, we find that untargeted adversarial attacks can result in a concentrated attack matrix - i.e. attacks perturb data into a relatively small number of classes. This could lead to the perturbations being less effect-

Figure 2.1: Randomly selected clean image (left), with perturbed counterparts at level $\varepsilon = 4/255$ (center), and $\varepsilon = 8/255$ (right). The clean image is taken from class "hen" and poisons are generated via perturbations into class "ostrich".

ive at degrading generalization because they are not discriminatively useful for a network and are thus ignored. In contrast, a class-targeted attack ensures that the adversarially perturbed features can be associated with an incorrect class label when learned by a victim network.

2.3.6 Facial Recognition

In our third experimental setting, we test whether our poisons are effective against facial recognition models, along the lines of the motivations introduced in Section 2.3.1, where companies like social media sites could employ our method to protect user data from being scraped and used by facial recognition networks as demonstrated in [26]. For the purposes of comparison, we constrain our attack to the same setting as Huang et al. [65]. In this setting, we only assume the ability to modify a subset of the face images, specifically, the face images of the users that we want to protect. We test on the Webface dataset, and use pretrained Webface model from Yi et al. [168] to generate the class targeted adversarial examples with $\varepsilon = 8/255$. Our class targeted adversarial examples are exceedingly effective in this domain. Our method reduces the protected

classes' accuracy down to 8%, half the classification success of the unlearnable examples method of Huang et al. [65] (see Table 2.4). We visualize perturbations in Figure 2.4. Note that these images are produced using the same ε bound as unlearnable examples for comparison, and for more discrete perturbations, one can easily turn down the ε radius to ensure more high fidelity user images.

Table 2.4: **Comparison of poisoning methods for facial recognition**. All poisons generated with $\varepsilon = 8/255$.

POISON METHOD	AVERAGE ACCURACY ($\%, \downarrow$)
CLEAN ACCURACY	86.00
UNLEARNABLE EXAMPLES [65]	16.00
OURS (CT)	**8.00**

2.3.7 Less Data

So far, our comparisons to previous methods have been in the setting of full dataset poisoning, as was tested in Feng et al. [40], Fowl et al. [42], Huang et al. [65], Shen et al. [143]. This setting is relevant to practitioners like social media companies who wish to modify all user data to prevent others from using data scraped from their platforms.

However, even in this setting, it may be the case that an actor scraping data has access to an amount of unperturbed data - either through previous scraping before a poisoning method was employed, or data scraped from another source. A concern that follows is how this mixture affects model training. Thus, we test the effectiveness of our poisons when different proportions of clean and perturbed data are used to train the victim model. Poisons are then considered effective if their use does not significantly increase performance over training on the clean data alone. In Tables 2.5, 2.6 we find that, poisoned

data does not significantly improve results over training on only the clean data and often degrades results below what one would achieve using only the clean data. This is in comparison to similar experiments in Huang et al. [65] where poisoned data was observed to be slightly helpful when combined with clean data in training a model.

Table 2.5: Effects of adjusting the amount of clean data included in the poisoned CIFAR-10.

POISON METHOD\CLEAN PROPORTION	0.1	0.2	0.5	0.8
NONE (ONLY CLEAN DATA)	82.30 ± 0.08	87.90 ± 0.04	92.48 ± 0.07	93.90 ± 0.06
$\varepsilon = 8/255$	85.34 ± 0.09	88.23 ± 0.09	92.20 ± 0.08	93.65 ± 0.07

Table 2.6: Effects of adjusting the amount of clean data included in the poisoned ImageNet.

POISON METHOD\CLEAN PROPORTION	0.1	0.2	0.5	0.8
NONE (ONLY CLEAN DATA)	45.18	53.48	62.79	64.65
$\varepsilon = 8/255$	47.81	54.24	61.20	63.95

2.3.8 Defenses

We have demonstrated the effects of our poisons in settings where the victim trains "normally". However, there have been several defenses proposed against poisoning attacks that a victim could potentially utilize. Thus, in this section, we test the effectiveness of several popular defenses against our method.

Adversarial training: Because our poisons are constrained in an ℓ_∞ ball around the clean inputs, and because we find that clean inputs are themselves adversarial examples for networks trained on poisoned data (more on this in the next section), it is possible that adversarial training could "correct" the poisoned network's behavior on the clean

distribution. This is hypothesized in Tao et al. [158], where it is argued that adversarial training can be an effective defense against delusive poisoning. We find in Table 2.7 that this is indeed the case. In fact, it is known that adversarial training effectively mitigates the success of several previous availability attacks including Huang et al. [65], Feng et al. [40], Fowl et al. [42]. However, it is worth noting that this might not be an ideal solution for the victim as adversarial training is expensive computationally, and it degrades natural accuracy to a level well below that of standard training. For example, on a large scale dataset like ImageNet, adversarial training can result in a significant drop in validation accuracy. With a ResNet-50, adversarial training results in validation accuracy dropping from 76.13% to 47.91% - a drop that would be further exacerbated when adversarial training on poisoned data (cf. `https://github.com/MadryLab/robustness`).

Data Augmentation: Because our poisoning method relies upon slight perturbations to training data in order to degrade victim performance, it is conceivable that further modification of data during training could counteract the effects of the perturbations. This has recently been explored in Borgnia et al. [15] and Geiping et al. [48]. We test several data augmentations *not* known to the poisoner during crafting, although adaptive attacks have been shown to be effective against some augmentations [48]. Our augmentations range from straightforward random additive noise (of the same ε-bound as poisons) to Gaussian smoothing. We also include popular training augmentations such as Mixup, which mixes inputs and input labels during training [174], Cutmix [170], which mixes patches of one image with another, and Cutout [28] which excises certain parts of training images.

DPSGD: Differentially Private SGD (DPSGD) [1] was originally developed as a

differential privacy tool for deep networks which aims to inure a dataset to small changes in training data, which could make it an effective antitode to data poisoning. This defense adds noise to training gradients, and clips them. It was demonstrated to be successful in defending against targeted poisoning in some settings [47, 62]. However, this defense often results in heavily degraded accuracy. We test this defense with a clipping parameter 1.0 and noise parameter 0.005.

Table 2.7: **Effects of defenses against adversarial poisons**. All results averaged over 5 runs of a ResNet-18 victim model trained on class targeted, $\varepsilon = 8/255$ poisons. Although adversarial training improves results, none of these effectively recover the accuracy of a model trained on clean data.

DEFENSE	VALIDATION ACCURACY (%)
BASELINE (CLEAN)	94.56
ADV. TRAINING	83.01
GAUSSIAN SMOOTHING	11.94
RANDOM NOISE	6.55
MIXUP	15.86
CUTMIX	10.09
CUTOUT	8.11
DPSGD	24.61

Other than adversarial training, which has drawbacks discussed above, we find that previous proposed defenses are ineffective against our adversarial poisons, and can even degrade victim performance below the level of a model trained without any defense.

2.4 Additional Experiments and Details

2.4.1 Training/Crafting details

Code for this project can be found at: `https://github.com/lhfowl/adversarial_pois`

Below are descriptions of experimental settings and hyperparameters. Unless otherwise

stated, for CIFAR-10, we train the crafting network for 40 epochs before generating the attacks. For testing, we use two primary setups. First, for the CIFAR-10 comparison and baseline Tables (2.1, 2.2), to be as fair and objective as possible, we use a totally third party testing setup found in the popular repo `https://github.com/kuangliu/pytorch-cifar`. Note for these runs, we updated our crafting procedure to use a slightly smaller step size ($0.05 \cdot \frac{8}{255}$).

For other CIFAR-10 ablations, for efficiency, we train victim models for 100 epochs with 3 learning rate drops with SGD optimizer. We craft with a step size of $0.1 \cdot \frac{8}{255}$. Unless otherwise stated, for CIFAR-10 experiments, we use 8 restarts on poison crafting, and perturbation pixels are initialized from $\mathcal{N}(0, \varepsilon^2)$.

For ImageNet, for efficiency, we use a pretrained crafting network (ResNet18 unless otherwise stated) taken from `https://pytorch.org/vision/stable/models.html` to craft the poisons. For memory constraints, we craft in batches of $25,000$, but note that this has no effect on the perturbations since the crafting is independent. Unless otherwise specified, we then train a randomly initialized model for 100 epochs with SGD using standard ImageNet preprocessing (resizing, center crops, normalization) with three learning rate drops. For the ablation studies on smaller proportions of data, for efficiency, we only train the ImageNet models for 40 epochs.

For crafting class targeted attacks, we select a random permutation of the labels. For CIFAR-10, this amounted to label $i \to i + 3$, and for ImageNet, we simply chose $i \to i + 3$.

For the facial recognition, we conduct our experiments following the partially unlearnable setting in Huang et al. [65]. In this setting, we only assume ability to modify

a subset of the face images, specifically, the faces images of the users that we want to protect. We first randomly split the Webface dataset into 80% training data and 20% testing data, and then randomly select 50 identities to be the users who want to hide their identities. We use the pretrained Webface model from Yi et al. [168] to generate the class targeted adversarial examples with $\varepsilon = 8/255$. We then combine the protected data together with the the remaining 10525 identities to form the new training dataset. We then train an Inception-ResNet following the standard procedures in [156]. We visualize perturbations in Figure 2.4.

2.4.1.1 Hardware and time considerations

We run our experiments on a heterogeneous mixture of resources including Nvidia GeForce RTX 2080 Ti GPUs, as well as Nvidia Quadro GV100 GPUs. Crafting and training time vary widely depending on the dataset and hyperparameter choices (restarts). However, a typical crafting experiment for CIFAR-10 will take roughly 6 hours to train and craft with 4 2080 Ti GPUs. This can be reduced by a factor of roughly 8 if one utilizes pretrained models, and only performs 1 restart during poison creation.

2.4.2 Visualization

In Figures 2.2, 2.3, we visualize randomly selected perturbed images at different ε levels. As with adversarial attacks, and other poisoning attacks, there is a trade-off between visual similarity and potency of the perturbations.

In Figure 2.4, we visualize perturbed identities from the Webface dataset.

Figure 2.2: Randomly selected example perturbations to CIFAR-10 datapoint (class "frog"). **Left**: unaltered base image. **Middle**: $\varepsilon = 4/255$ perturbation. **Right**: $\varepsilon = 8/255$ perturbation. Networks trained on perturbations including the one on the right achieve below random accuracy.

Figure 2.3: Randomly selected example perturbations to ImageNet datapoints. Perturbations bounded by $\varepsilon = 8/255$.

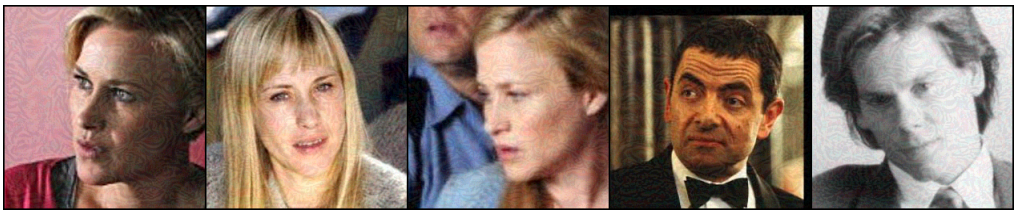


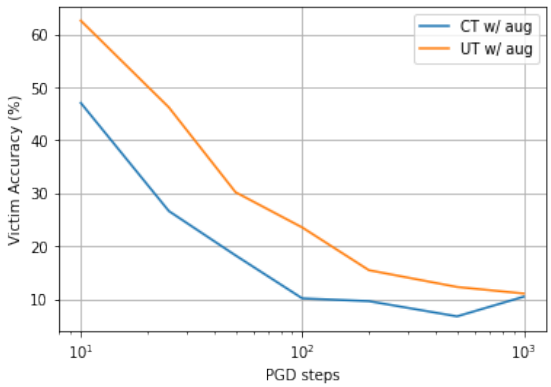
Figure 2.4: Samples of poisoned Webface images $\varepsilon = 8/255$

2.4.3 Adversary comparison

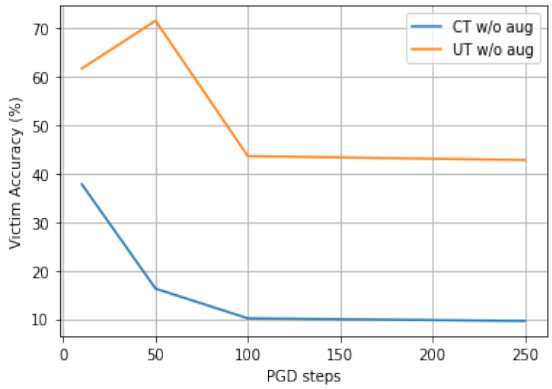
We find that a PGD based attack supersedes other common adversarial attacks in poison efficiency - a behavior that has been observed in classical adversarial attacks as well. These results can be found in Table 2.8.

Table 2.8: **Validation accuracies of victim models trained on data generated by different adversarial attacks.** Tested in the black-box setting on randomly initialized models on CIFAR-10.

METHOD \ VICTIM	VGG19	RESNET-18	GOOGLENET	DENSENET-121	MOBILENETV2
PGD W/ AUG	10.98 ± 0.27	6.25 ± 0.17	7.03 ± 0.12	7.16 ± 0.16	6.11 ± 0.17
CW	59.82 ± 1.36	48.40 ± 3.24	19.25 ± 1.15	43.40 ± 2.76	45.62 ± 4.87
FGSM	65.32 ± 0.58	76.35 ± 0.08	47.50 ± 1.06	59.44 ± 1.28	74.77 ± 0.43
FEATURE EXPLOSION	82.41 ± 0.38	83.26 ± 0.94	78.53 ± 0.49	81.83 ± 0.46	82.30 ± 0.88



(a) Ablating the number of PGD steps used for crafting. Step size is fixed at $0.05 \cdot \frac{8}{255}$



(b) Ablating the number of PGD steps used for crafting, without augmentation. Step size is fixed at $0.05 \cdot \frac{8}{255}$

Figure 2.5: Victim network accuracy as a function of PGD steps with and without augmentation. Augmentation appears necessary to produce powerful untargeted poisons.

2.4.4 Crafting Ablations

In Table 2.9, we find that the more steps we perform in the PGD optimization of adversarial poisons, the more effective they become. However, the poisons still degrade validation accuracy at lower numbers of steps.

2.4.5 Network Variation

Here we test how perturbations crafted using one network transfer to other networks. While it has been demonstrated that adversarial examples used as evasion attacks

Table 2.9: **Comparison of different number of crafting steps for our adversarial poisoning method.** All poisons generated by ResNet-18 with steps of PGD, with differentiable data augmentation, and class-targeted objective.

STEPS \VICTIM	VGG19	RESNET-18	GOOGLENET	DENSENET-121	MOBILENETV2
50 STEPS	25.10 ± 0.60	16.53 ± 0.26	18.51 ± 0.34	18.91 ± 0.34	15.29 ± 0.40
100 STEPS	16.06 ± 0.41	9.87 ± 0.26	11.66 ± 0.33	13.42 ± 0.28	10.38 ± 0.22
250 STEPS	10.98 ± 0.27	6.25 ± 0.17	7.03 ± 0.12	7.16 ± 0.16	6.11 ± 0.17

Table 2.10: **A comparison of different optimizers, crafting objectives, and use of differentiable data augmentation in crafting.** All poisons crafted with bound $\varepsilon = 8/255$. If not otherwise stated, poisons are crafted with PGD, differentiable data augmentation, and our class-targeted objective.

METHOD \VICTIM	VGG19	RESNET-18	GOOGLENET	DENSENET-121	MOBILENETV2
PGD W/ AUG	10.98 ± 0.27	6.25 ± 0.17	7.03 ± 0.12	7.16 ± 0.16	6.11 ± 0.17
SIGNADAM W/ AUG	8.92 ± 0.27	6.17 ± 0.27	6.75 ± 0.18	6.42 ± 0.23	7.15 ± 0.22
SIGNADAM W/O AUG	9.96 ± 0.22	6.96 ± 0.15	7.41 ± 0.15	7.84 ± 0.29	8.22 ± 0.23
CW LOSS	59.82 ± 1.36	48.40 ± 3.24	19.25 ± 1.15	43.40 ± 2.76	45.62 ± 4.87
FEATURE EXPLOSION LOSS	82.41 ± 0.38	83.26 ± 0.94	78.53 ± 0.49	81.83 ± 0.46	82.30 ± 0.88

can often transfer across architectures, we find that adversarial examples as poisons also transfer across different architectures in Table 2.11.

Table 2.11: **Results varying the *crafting* network for the poisons.** All poisons crafted with bound $\varepsilon = 8/255$, PGD, differentiable data augmentation, class-targeted objective.

CRAFTING \TESTING	VGG19	RESNET-18	GOOGLENET	DENSENET-121	MOBILENETV2
RESNET-18	10.98 ± 0.27	6.25 ± 0.17	7.03 ± 0.12	7.16 ± 0.16	6.11 ± 0.17
RESNET-50	17.86 ± 0.57	9.71 ± 0.12	11.44 ± 0.21	10.64 ± 0.46	6.82 ± 0.18
VGG19	20.88 ± 0.84	18.53 ± 1.15	21.48 ± 0.67	23.77 ± 0.74	17.59 ± 0.56
MOBILENETV2	21.66 ± 0.26	12.42 ± 0.17	14.84 ± 0.29	15.95 ± 0.18	9.60 ± 0.24
CONVNET	15.43 ± 0.34	10.05 ± 0.20	11.88 ± 0.17	10.30 ± 0.11	7.95 ± 0.26

In addition to experimenting with different crafting network architectures, we also vary other factors of the crafting network. For example, we experiment how using an adversarially robust crafting network affects performance of the poisons. We find that adversarially trained models produce ineffective poisons. This is in line with the findings of [69] that robust models leverage a different set of discriminatory features - ones less

brittle to perturbation - during classification. These results can be found in Table 2.12.

Table 2.12: Poison generation using a robust crafting model.

CRAFT MODEL \ VICT. MODEL	VGG19	RESNET-18	GOOGLENET	DENSENET-121	MOBILENETV2
ROBUST RESNET-18	85.58 ± 0.05	81.79 ± 0.57	80.99 ± 0.17	80.39 ± 0.27	77.70 ± 1.27

2.4.6 Relabeling Trick

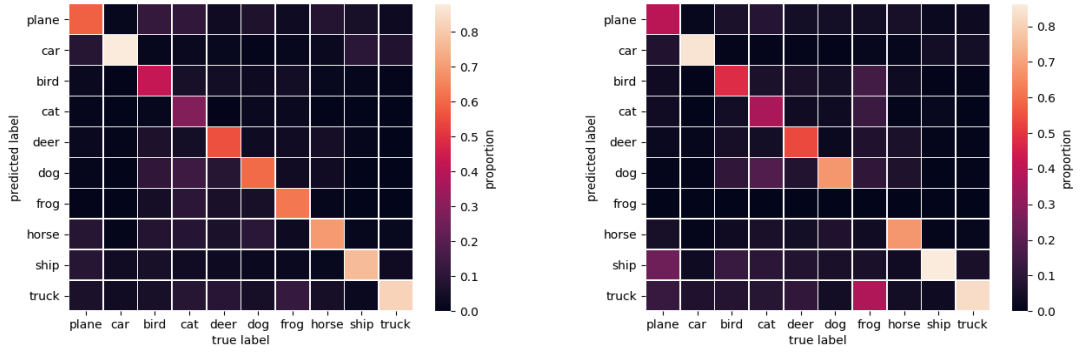
In Table 2.13, we find that the poisons generated via our method cause class "confusion" when the victim network trains. That is to say, the victim network incorrectly associates the target label with the non-robust features found in the perturbed data.

Table 2.13: **Training the victim with labels corrected to the "adversarial labels".** Poisons are crafted on a CIFAR-10 trained ResNet-18 with 250 steps of PGD and differentiable data augmentation.

DATA \ VICTIM	VGG19	RESNET-18	GOOGLENET	DENSENET-121	MOBILENETV2
UNCORRECTED CIFAR-10	10.98 ± 0.27	6.25 ± 0.17	7.03 ± 0.12	7.16 ± 0.16	6.11 ± 0.17
CORRECTED ONE-HOT CIFAR-10	74.95 ± 0.31	78.98 ± 0.25	77.72 ± 0.37	78.55 ± 0.36	74.33 ± 0.24
CORRECTED SOFTMAX CIFAR-10	75.88 ± 0.25	75.69 ± 0.25	73.57 ± 0.47	70.26 ± 0.32	69.46 ± 0.32
CORRECTED ONE-HOT SVHN	31.13 ± 0.49	30.19 ± 0.16	40.71 ± 0.12	40.31 ± 0.43	34.18 ± 0.29

2.4.7 Learning Without Seeing

We have seen that it is not necessary to have "correctly" labeled data (labeled with ground truth labels) in order for a network to achieve good performance at test-time. We can extend this to the question: does one need ground truth data *at all* to learn how to classify? For example, can a network learn how to classify cats without ever seeing an image of a cat, but instead only seeing images of dogs perturbed to look like cats? We find



(a) Test-time predictions of network trained on label-corrected, class-targeted poisons.

(b) Test-time predictions of network trained on label-corrected, class-targeted poisons without any images of "cats".

Figure 2.6: Classification heatmaps. **Left** - heatmap of clean test predictions from network trained on label-corrected, class based targeted poisons. **Right** - heatmap of clean test predictions from network trained on same poisons, without any "cat" images.

the answer is yes. Specifically, we conduct an experiment where we craft targeted, class-based poisons (i.e. all images of one class are perturbed via targeted attacks into another class). We then train on the label-corrected full data, and also train on a label-corrected pruned training set without any images from the cat class. Interestingly, we find that the network trained on the ablated set is able to classify clean, test-time images of cats having never seen an example during training! Moreover, the network fails to classify clean images from the class into which cats were perturbed (class 6, "frog") under the targeted attack. Instead, clean test-time frogs are more likely to be classified as class 9 ("truck") - the class where frog images were perturbed into under the crafting attack. Interestingly, this demonstrates that the network not only learns to associate the perturbed features of frogs with the label "truck", but also features of clean frog images even though the network never encountered "clean" frog features during training because the cat class was ablated. These results can be found in Figure 2.6.

2.4.8 ImageNet Comparison

While our main ImageNet results are in a realistic setting of training for 100 epochs, previous art has poisoned ImageNet victims trained for 40 epochs. We compare our method to their's here (Table 2.14) and find that our clas targeted attack far outperforms their attack based on gradient alignment.

Table 2.14: Effects of adjusting the amount of clean data included in the poisoned ImageNet. The same number of gradient updates were used for each model (equivalent to 40 epochs for full sized ImageNet).

POISON METHOD\POISON BOUND	$\varepsilon = 8/255$
CLEAN	65.70
ALIGNMENT	37.58
OURS	**1.57**

2.4.9 Instability of Untargeted Attacks

While untargeted adversarial poisons are able to significantly degrade the validation accuracy of a victim models, it is a weaker attack than our class targeted attack, and it can be more brittle to poison initialization. Some poisons generated with untargeted attacks achieve much worse performance than others. We hypothesize this is because the poison initialization has a large effect on which class the untargeted attack is perturbed into, thus affecting the feature confusion aspect to poisoning. As mentioned in the main body, we perform modest hyperparameter searching (poison initialization) for the untargeted CIFAR-10 attack. Specifically, for comparison to other methods, for the untargeted attack, we seed the poisons and model initialization for poison crafting, and take the best run from

a modest number of trials (< 15). This is not unreasonable for a practitioner like a social media company as they potentially have access to the entire dataset which they intend to poison, and can choose the best set of perturbations. In these trials, we observe that the average success ($\%$) of these poisons (now varying over initialization of the poisons) is 19.17 with standard deviation 2.87. The most potent poisons in these runs degrade victim accuracy to 11.94%, and it is on these poisons that we run comparisons. Note that these do not represent the *most* potent untargeted poisons we have found, but rather a "reasonable" best set a poisoner can hope to find. Even more potent poisons can be found linked through our Github `https://github.com/lhfowl/adversarial_poisons`.

On the other hand, our class-targeted objective is much more stable to poison/model initialization, and thus we do not perform hyperparameter searches for these runs. The relative weakness of the untargeted objective is evident in our ImageNet results in Table 2.3. We hypothesize the success of the untargeted variant depends on how "well distributed" the attack matrix is (which in turn depends on poison initialization). We have included a heatmap of an untargeted attack in Figure 2.7.

2.5 Analysis

Why do adversarial examples make such potent poisons? In Section 2.1, we motivate the effectiveness of adversarial poisons with the explanation that the perturbed data contains semantically useful information, but for the wrong class. For instance, an adversarially perturbed "dog" might be labeled as a "cat" by the crafting network because the perturbations contain discriminatory features useful for the "cat" class. In Ilyas et al.

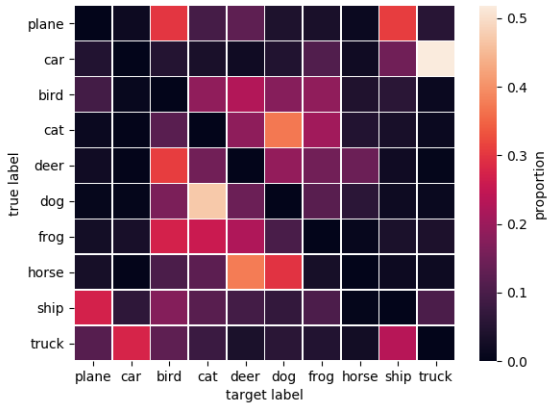


Figure 2.7: An example heatmap of an untargeted poisoning measuring the attack distribution on the crafting network. Note how this attack "well distributes" the target labels (i.e. not every class is perturbed into the same class). We hypothesize this is important for a successful attack.

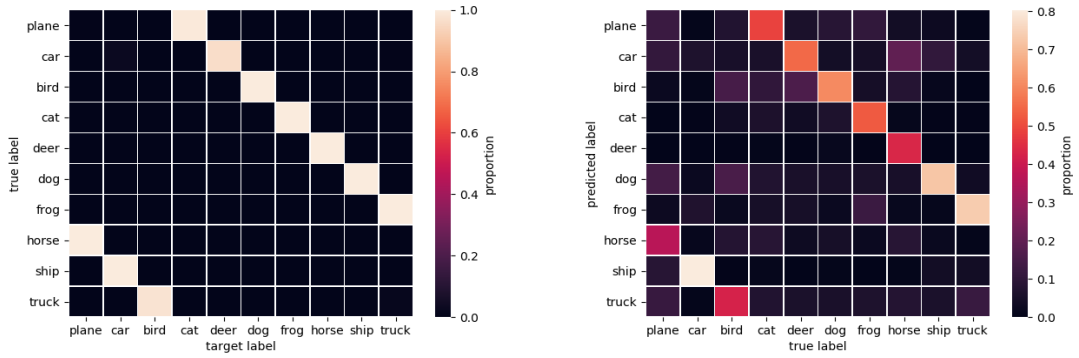
[69], the authors discover that there exist image features which are both brittle under adversarial perturbations and useful for classification. Ilyas et al. [69] explores these implications for creating robust models, as well as "mislabeled" datasets which produce surprisingly good natural accuracy. We investigate the implications that the existence of non-robust features have on data-poisoning attacks. We hypothesize that adversarial examples might poison networks so effectively because, for example, they teach the network to associate "cat" features found in perturbed data with the label "dog" of the original, unperturbed sample. Then, when the network tries to classify clean test-time data, it leverages the "mislabeled" features found in the perturbed data and displays low test-time accuracy. We confirm this behavior by evaluating *how* data is misclassified at test-time. We find that the distribution of predictions on *clean* data closely mimics the distributions of labels assigned by the network used for crafting after adversarial attacks, even though these patterns are generated from different networks.

To tease apart these effects, we conduct several experiments. First, we verify that the victim network does indeed train - i.e. reach a region of low loss - on the adversarial examples. This is in contrast to the motivation of [65, 143] which try to *prevent* the network from training on poisoned data. We find that the victim network is able to almost perfectly fit the adversarial data. However, the accuracy of the victim network on the original, unperturbed training data is just as low as accuracy on the clean validation data, revealing an interesting duality - clean training data are adversarial examples for networks trained on their perturbed counterparts (see Table 2.15). But are adversarial examples simply so different from clean examples that learning on one is useless for performing inference on the other? Or do adversarial examples contain useful features but for the *wrong* class?

To investigate these hypotheses, we train models on the adversarial poisons crafted with the class targeted objective found in Equation 2.4 so that every image from a given class is perturbed into the same target class. We then observe the classification patterns of the victim network and compare these to the attack patterns of the crafting network. We find that poisoned networks confuse exactly the same classes as the crafting network. Specifically, a network trained on dog images perturbed into the cat class will then misclassify clean cat images as dogs at test time as the network learns to associate "clean" cat features found in the perturbations to the training dog images with the label dog. This behavior is illustrated in Figure 2.8.

To further confirm our hypothesis, we employ the re-labeling trick introduced in Ilyas et al. [69], and we train the victim network on the "incorrect" labels assigned to the poisons by the *crafting* network. For example, if a dog image is perturbed into the "cat"

class by an adversarial attack on the crafting network, we train on the perturbed dog image but assign it the "cat" label. We find that this simple re-labeling trick boosts validation accuracy significantly, for example from 6.25% to 75.69% on a victim ResNet-18 model (see Table 2.13). This confirms the finding of Ilyas et al. [69] concerning non-robust features. One might think that this observation is useful for defending against adversarial poisons. However, in order to perform this label correction, the victim must have access to the crafting model which requires the victim to already possess the original non-poisoned dataset.



(a) Adversarial attack predictions of network used to craft adversarial poisons.

(b) Test-time predictions of network trained on adversarial poisons.

Figure 2.8: Classification heatmaps. **Left** - heatmap of predictions after the adversarial attack on the network used for crafting. **Right** - heatmap of clean test predictions after training a new network on adversarial poisons.

One further intricacy remains; even if adversarial examples contained no useful features, they may still encode decision boundary information that accounts for the increased validation accuracy - behaviour that has previously been demonstrated with out-of-distribution data in Nayak et al. [114]. However, we find that training on data from a drastically different distribution (SVHN), labeled with the CIFAR-10 crafting network's predictions, fails to achieve comparable CIFAR-10 validation accuracy. This confirms

that the adversarial CIFAR-10 images contain useful features for the CIFAR-10 distribution but are simply mislabeled.

Table 2.15: **Testing the victim on clean vs. adversarial training images.** Poisons are crafted on a CIFAR-10 trained ResNet-18 with 250 steps of PGD and differentiable data augmentation.

MEASUREMENT \ VICTIM	VGG19	RESNET-18	GOOGLENET	DENSENET-121	MOBILENETV2
TRAINING ACC. ON POISONS	99.95 ± 0.00	99.99 ± 0.00	99.99 ± 0.00	99.99 ± 0.00	99.94 ± 0.00
ACC. ON CLEAN TRAIN DATA	10.98 ± 0.32	6.16 ± 0.16	6.80 ± 0.08	7.06 ± 0.13	6.15 ± 0.17

2.6 Conclusion

There is a rising interest in availability attacks against deep networks due to their potential use in protecting publicly released user data. Several techniques have been introduced leveraging heuristics such as loss minimization, and auto-encoder based noise. However, we find that adversarial attacks against a fixed network are more potent availability poisons than existing methods, often degrading accuracy below random guess levels. We study the effects of these poisons in multiple settings and analyze why these perturbations make such effective poisons. Our observations confirm a fundamental property of adversarial examples; they contain discriminatory features but simply for the wrong class. Our class-targeted attack leverages this property to effectively poison models on a variety of datasets.

Chapter 3: Malicious Model Modifications for Federated Learning

Federated learning has quickly gained popularity with its promises of increased user privacy and efficiency. Previous works have shown that federated gradient updates contain information that can be used to approximately recover user data in some situations. These previous attacks on user privacy have been limited in scope and do not scale to gradient updates aggregated over even a handful of data points, leaving some to conclude that data privacy is still intact for realistic training regimes. In this work, we introduce a new threat model based on minimal but malicious modifications of the shared model architecture which enable the server to directly obtain a verbatim copy of user data from gradient updates without solving difficult inverse problems. Even user data aggregated over large batches – where previous methods fail to extract meaningful content – can be reconstructed by these minimally modified models. This work was conducted with Jonas Geiping, Wojtek Czaja, Micah Goldblum, and Tom Goldstein. My contributions include jointly conceiving of the mechanism of the imprint module, implementing the first variant of the module, and writing a substantial portion of the paper.

3.1 Introduction

Federated learning [85], also known as collaborative learning [144], is a mechanism for training machine learning models in a distributed fashion on multiple user devices. In the simplest setting, a central *server* sends out model states to a group of *users*, who compute an update to the model based on their local data. These updates are then returned to the server, aggregated, and used to train the model. Over multiple rounds, this protocol can train a machine learning model, distributed over all users, without exchanging local data – only model updates are exchanged. Two central goals of federated learning are to improve training efficiency by decreasing communication overhead and to side-step issues of user-level privacy and data access rights that have become a focus of public attention in recent years [163].

Accordingly, many organizations, ranging from large tech companies [102] to medical institutions with especially strict privacy laws, such as hospitals [73], have utilized federated learning to train machine learning models. However, in practice, data privacy is not guaranteed in general, but is dependent on a large number of interdependent settings and design choices specific to each federated learning system. In this work, we focus on the user perspective of privacy, and we study federated learning systems in which the central server is not able to directly view user data.

The key privacy concern for users is whether model updates reveal too much about the data on which they were calculated. Although Kairouz et al. [75] discuss that "model updates are more focused on the learning task at hand than is the raw data (i.e. they contain strictly no additional information about the user, and typically significantly less,

compared to the raw data)", scenarios can be constructed in which the model updates themselves can be inverted to recover their input user information [106, 165]. Simple knowledge of the shared model state and model update can be sufficient for such an attack [46, 180]. These inversion attacks are particularly fruitful if a user's model update is based on a single data point or only a small batch. Accordingly, a strong defense against these attacks is aggregation. The user only reports model updates aggregated over a significant number of local data points, and data from multiple users can be combined with secure aggregation protocols [12] before being passed to the server. This ability to aggregate user updates while maintaining their utility is thought to be the main source of security in federated learning. Averaging raw local data in similar amounts would make it unusable for training, but model updates can be effectively aggregated.

Previous inversion attacks typically focus on a threat model in which the server (server here is a stand-in for any party with root access to the server or its incoming and outgoing communication) is interested in uncovering user information by examining updates, but without modifying the federated learning protocol, a behavior also referred to as *honest-but-curious* or *semi-honest* [53]. In our case, where the party intending to recover user data is the server, this "honest" scenario appears contrived, as the server can modify its behavior to obtain private information. In this work, we are thus interested in explicitly *malicious servers* that may modify the model architecture and model parameters sent to the user.

We focus on scenarios in which an agent obtains data without making suspicious changes to the client code or learning behavior. One scenario which enables this threat model involves recently introduced APIs that allow organizations to train their own mod-

els using established federated learning protocols [20]. In this environment, a malicious API participant can change their model's architecture and parameters but cannot force unsuspecting edge devices to send user data directly.

We introduce minimal changes to model architectures that enable servers to breach user privacy, even in the face of large aggregations that have been previously deemed secure. These changes induce a structured pattern in the model update, where parts of the update contain information only about a fixed subset of data points. The constituent data points can then be recovered exactly, while evading existing aggregation defenses.

3.2 Limitations of Existing Attack Strategies

A range of possible attacks against privacy in federated learning have been proposed in recent literature. In the simplest case of *analytic attacks*, Phong et al. [127] were among the first to discuss that the input to a learnable affine function can be directly computed from the gradient of its weights and bias, and additional analysis of this case can be found in Fan et al. [36], Qian and Hansen [128], and in section 3.3.2. However, analytic recovery of this kind only succeeds for a single data point. For multiple data points, only the average of their inputs can be recovered, leading the attack to fail in most realistic scenarios.

Recursive attacks as proposed in Zhu and Blaschko [179] can extend analytic attacks to models with more than only linear layers - a construction also mentioned in [36]. However, these attacks still recover only the average of inputs in the best case. Improvements in Pan et al. [118] transform linear layers with ReLU activations into systems of

linear equations that allow for a degree of recovery for batched inputs to these linear layers, although preceding convolutional layers still have to be deconvolved by recursion or numerical inversion techniques.

Surprisingly, *optimization-based attacks* turn out to be highly effective in inverting model updates. Wang et al. [165] propose the direct recovery of input information in a setting where the users' model update is the model parameter gradient averaged over local data. In a supervised learning setting, we define this update by g and the loss function over this model by $\mathcal{L}$ with model parameters θ and data points $(x, y) \in [0, 1]^n \times \mathbb{R}^m$. The server can then attempt recovery by solving the gradient matching problem of

$$\min_{x \in [0,1]^n} ||\nabla_\theta \mathcal{L}(x, y, \theta) - g||^2 \tag{3.1}$$

and solve this optimization objective using first-order methods or any nonlinear equation solver. Subsequent work in Zhao et al. [176], Zhu et al. [180] and Wainakh et al. [164] proposes solutions that also handle recovery of targets y and variants of this objective are solved for example in Geiping et al. [46] with cosine similarity and improved optimization and in Jeon et al. [72] with additional generative image priors. Reconstruction of input images can be further boosted by additional regularizers as in Yin et al. [169] and Qian et al. [129].

Most attacks in the literature focus on the described *fedSGD* setting [85] in which the users return gradient information to the server, but numerical attacks can also be performed against local updates with multiple local steps [46], for example against *fedAVG* [103]. In this work, we will discuss both update schemes, noting that gradient aggregation

in “time”, with multiple local update steps, is not fundamentally more secure than aggregation over multiple data points. Previous attacks also focus significantly on learning scenarios where the user data is comprised of images. This is an advantage to the attacker, given that image data is highly structured, and a multitude of image priors are known and can be employed to improve reconstruction. In contrast, data types with weaker structure, such as tabular data, do not lend themselves to regularization based on strong priors, and we will show that our approach, on the other hand, does not rely on such tricks, and is therefore more data-agnostic.

The central limitation of these attack mechanisms is the degradation of attack success when user data is aggregated over even moderately large batches of data (either by the user themselves or by secure aggregation). State-of-the-art attacks such as Yin et al. [169] recover only 28% of the user data (given a charitable measure of recovery) on a batch size of 48 for a ResNet-50 [59] model on ImageNet (ILSVRC2012 [137]) with unlikely label collisions. The rate of images that can be successfully recovered drops drastically with increased batch sizes. Even without label collisions, large networks such as a ResNet-32-10 [171] leak only a few samples for a batch size of 128 in Geiping et al. [46]. These attacks further reconstruct only approximations to the actual user data which can fail to recover parts of the user data or replace it with likely but unrelated information in the case of strong image priors.

Further, although all of the previous works nominally operate under an *honest-but-curious* server model, they do often contain model adaptations on which reconstruction works especially well, such as large vision models with large gradient vectors, models with many features [165, 179], special activation functions [179, 180], wide models [46],

or models trained with representation learning [24, 169]. These may be seen as *malicious* models with architectural choices that breach user privacy. In the same vein, we ask, what is the worst-case (but small) modification that can be applied to a neural network to break privacy?

3.3 Model Modifications

In this section, we detail an example of a small model modification that has a major effect on user privacy, even allowing for the direct recovery of verbatim user data from model updates.

3.3.1 Threat Model

We define two parties: the server $\mathcal{S}$ and the users $\mathcal{U}$. The server could be a tech company, a third party app using a federated learning framework on a mobile platform, or an organization like a hospital. The server S defines a model architecture and distributes parameters θ for this architecture to the users, who compute local updates and return them to the server. The server cannot deviate from standard federated learning protocol in ways beyond changes to model architecture (within limits imposed by common ML frameworks) and model parameters. We measure the strength of a malicious modification of the architecture using the number of additional parameters inserted into the model. While it is clear that models with more parameters can leak more information, we will see that clever attacks can have a disproportionate effect on attack success, compared to more benign increases in parameter count, such as when model width is increased.

3.3.2 A Simple Example

To motivate the introduction of malicious modifications, we start with the simple case of a fully connected layer. A forward pass on this layer is written as $y = Wx + b$ where W is a weight matrix, b is a bias, and x is the layer's input. As seen in [36, 126, 128], when the parameters of the network are updated according to some objective $\mathcal{L}$, the i^{th} row of the update to W:

$$\nabla_{W^i}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial y^i} \cdot \nabla_{W^i} y^i = \frac{\partial \mathcal{L}}{\partial y^i} \cdot x,$$

where we use the shorthand $\mathcal{L} = \mathcal{L}(x; W, b)$. Similarly,

$$\frac{\partial \mathcal{L}}{\partial b^i} = \frac{\partial \mathcal{L}}{\partial y^i}\frac{\partial y^i}{\partial b^i} = \frac{\partial \mathcal{L}}{\partial y^i}.$$

So as long as there exists some index i with $\frac{\partial \mathcal{L}}{\partial b^i} \neq 0$, the single input x is recovered perfectly as:

$$x = \nabla_{W^i}\mathcal{L} \oslash \frac{\partial \mathcal{L}}{\partial b^i}, \tag{3.2}$$

where $\oslash$ denotes entry-wise division.

However, for batched input x, all derivatives are summed over the batch dimension n and the same computation can only recover $\sum_{t=1}^{n} \nabla_{W_l^i}\mathcal{L}_t \oslash \sum_{t=1}^{n} \frac{\partial \mathcal{L}_t}{\partial b_l^i}$ from each row where $\sum_{t=1}^{n} \frac{\partial \mathcal{L}_t}{\partial b_l^i} \neq 0$, which is merely proportional to $\sum_{t=1}^{n} x_t$. If $\mathcal{L}$ is a linear regression, then this shows that only the average can be recovered. However, data points x_t only appear in the average if $\frac{\partial \mathcal{L}_t}{\partial y_t^i}$ is non-zero, a phenomenon also discussed in Sun et al. [152].

If $\mathcal{L}$ has a sparse gradient, e.g. in a multinomial logistic regression, then this structured gradient weakens the notion of averaging: Let x be a batch of data with unique labels $1, \ldots, n$. In this setting $\frac{\partial \mathcal{L}_t}{\partial y_t^i} = 0$ for all $i \neq t$, so that each row i actually recovers

$$x_t = \sum_{t=1}^{n} \frac{\partial \mathcal{L}_t}{\partial y_t^i} x_t \oslash \sum_{t=1}^{n} \frac{\partial \mathcal{L}_t}{\partial y_t^i} = \frac{\partial \mathcal{L}_t}{\partial y_t^i} x_t \oslash \frac{\partial \mathcal{L}_t}{\partial y_t^i}. \tag{3.3}$$

For a batch of n data points with unique labels, we could thus recover all data points exactly for this multinomial logistic regression. We visualize this in Appendix fig. 3.11 for ImageNet data Russakovsky et al. [137] (image classification, 1000 classes), where we could technically recover up to 1000 unique data points in the optimal case. However, this setup is impractical and suffers from several significant problems:

- **Averaging**: Multiple image reconstruction as described above is only possible in the linear setting, and with a logistic regression loss, a loss that depends on sparse logits is used. Even in this restrictive setting, reconstruction fails as soon as labels are repeated in a user update (which is the default case and outside the control of the server), especially if the accumulation size of a user update is larger than the underlying label space of the data. In this case, the server reconstructs the *average* of repeated classes. In Appendix fig. 3.11, we see that in the worst-case scenario where all data points fall into the same class, each piece of user data contributes to the gradient equally, resulting in a mashup reconstruction that leaks little private information.

- **Integration**: As stated above, the naive reconstruction is only guaranteed to work

only if the linear (single-layer) model is a standalone model, and not within a larger network. If the naive linear model was placed before another network, like a ResNet-18, then gradient entries for the linear layer contain elements averaged over all labels, as the combined network ostensibly depends on each output of the linear layer.

- **Scalability**: on an industrial scale dataset like ImageNet, the naive logistic regression model would require $> 150M$ parameters to retrieve an image from each label, which is of course far from any practical application.

3.3.3 Imprinting User Information into Model Updates

Nonetheless, the perfect reconstruction afforded by the linear model remains an attractive feature. To this end, we introduce the *imprint module* class of modifications which overcome the previously described issues, while maintaining the superior reconstruction abilities of an analytic reconstruction as described above. Further, the imprint module can be constructed from a combination of commonly used architectural features with maliciously modified parameters that can create structured gradient entries for large volumes of data.

The imprint module can be constructed with a single linear layer (with bias), together with a ReLU activation. Formally, let $\{x_i\}_{i=1}^{n} = X \in \mathbb{R}^{n \times m}$ be a batch of size n of user data, then a malicious server can define an imprint module whose forward pass (on a

single datapoint, x) looks like

$$M(x) = f(W_* x + b_*),$$

where f is a standard ReLU nonlinearity. The crux of the imprint module lies in the construction of $W_* \in \mathbb{R}^{k \times m}$ and $b_* \in \mathbb{R}^k$. We denote the i^{th} row (or channel) of W_* and the i^{th} entry of b_* as W_*^i and b_*^i, respectively. We then construct $W_*^{(i)}$ so that

$$\langle W_*^i, x \rangle = h(x),$$

where h is *any* linear function of the data where the server can estimate the distribution of values $\{h(x)\}_{x \sim \mathcal{D}}$ of this function on the user data distribution. For example, if the user data are images, h could be average brightness, in which case W_*^i is simply the row vector with entries identically equal to $\frac{1}{m}$.

In order to define the entries of the bias vector, we assume that the server knows the cumulative density function (CDF), assumed to be continuous for the quantity measured by h. For brightness this is straightforward as image data is often normalized. For other quantities, e.g. red-channel intensity, it would not be difficult for a malicious server to estimate this distribution. We stress that the choice of h here is not important to our method. For the purpose of explanation, we will assume that the quantity measured by h is distributed normally, with $\mu = 0$, $\sigma = 1$. Then, the biases of the imprint module are determined by

$$b_*^i = -\Phi^{-1}(\frac{i}{k}) = -c_i,$$

where Φ^{-1} is the inverse of the standard Gaussian CDF. In plain language, we first measure some quantity, like brightness, with the matrix W_*. We duplicate this measurement along the k channels (rows) of W_*. In the meantime, we create k "bins" for the data corresponding to intervals of equal mass according to the CDF of h. Then, the measurement for a given datapoint will land somewhere in the distribution of h.

For example, consider the case when the brightness of some image x_t lands between two values: $c_l \leq h(x_t) \leq c_{l+1}$, and no other image in the same batch has brightness in this range. In this situation, we say that x_t alone activates bin l. Then, if the image x_t is passed through the imprint module, we have

$$\left(\nabla_{W_*^l}\mathcal{L} - \nabla_{W_*^{l+1}}\mathcal{L}\right) \oslash \left(\frac{\partial \mathcal{L}}{\partial b_*^l} - \frac{\partial \mathcal{L}}{\partial b_*^{l+1}}\right) = x_t + \sum_{s=1}^{p} x_{i_s} - \sum_{s=1}^{p} x_{i_s} = x_t, \tag{3.4}$$

where images $\{x_{i_s}\}$ are images from the batch with brightness $> c_l$. That is, the difference in successive rows $l, l+1$ of the gradient entry for W_* correspond to all elements with brightness $c_l \leq h(x) \leq c_{l+1}$ (in this case, assumed to be just x_t). This is because all of $x_t \cup \{x_{i_s}\}$ activate the non-linearity for layer l, since all these images have brightness $\geq c_l$, however, only images $\{x_s\}$ have brightness $\geq c_{l+1}$, so only these images activate the non-linearity for layer $l+1$. An interesting biproduct of this setup is that it would be difficult even for hand inspection of the parameters to reveal the inclusion of this module as the server could easily permute the bins, and add random rows to W_* which do not correspond to actual bins and only contribute to model performance. The gradient does not directly contain user data, so that the leak is also difficult to find by analyzing the gradient data and checking for matches with user data therein.

How successful will this attack be? Recall the parameter k defined in the construction of the imprint module. This corresponds to the number of bins the malicious server can create to reconstruct user data. If a *batch* of data is passed through the imprint module, depending on the batch size n used to calculate the update sent to the server, and number of bins, k, the server can expect several bins to activate for *only* one datapoint. And the corresponding entries of the gradient vector can be appropriately combined, and inverted easily. The following result quantifies the user vulnerability in terms of the number of imprint bins, k, and the amount of data, n, averaged in a given update.

Proposition 3.3.1. *If the server knows the CDF (assumed to be continuous) of some quantity associated with user data that can be measured with a linear function $h : \mathbb{R}^m \to \mathbb{R}$, then for a batch of size n and a number of imprint bins $k > n > 2$, by using an appropriate combination of linear layer and ReLU activation, the server can expect to exactly recover*

$$\frac{1}{\binom{k+n-1}{k-1}}\left[\sum_{i=1}^{n-2} i \cdot \binom{k}{i} \cdot \left(\sum_{j=1}^{\lfloor \frac{n-i}{2} \rfloor} \binom{k-i}{j}\binom{n-i-j-1}{j-1}\right)\right] + r(n,k),$$

samples of user data (where the data is in $\mathbb{R}^m$) perfectly. Note: $r(n,k) = \frac{n}{\binom{k+n-1}{k-1}}\binom{k}{n} - \frac{n}{k}$ is a correction term (see proof for full expansion).

Proof. By construction of the imprint module, given a random sample (batch) $X_1, \ldots, X_n$ (iid) the server perfectly recovers data whenever an imprint bin has exactly 1 element of the batch. Because we know the CDFs, we can create partitions of equal mass corresponding to imprint bins $\{b_j\} = \{[a_j, b_j]\}$ where $P(X_i \in [a_j, b_j]) = 1/k \; \forall i, j$.

We can then phrase the problem of expected number of perfectly recovered samples as a modified "stars and bars" problem. For a given batch of data, to calculate the amount of data recovered, we first calculate:

$$\sum_{i=1}^{n} i \cdot \binom{k}{i} \cdot N_i,$$

where $\binom{k}{i}$ is the number of ways to select the i bins that have exactly 1 element, and N_i is the number of orientations of the remaining data into the remaining bins so that no bin has exactly 1 element. Note that we can do this because the bins all have equal mass, and thus we can factor out the (uniform) probability of any configuration from the sum.

Simply put, we first take the configuration where there is only 1 bin with exactly 1 element, and weight it by 1, then we take the number of configurations with 2 bins with exactly 1 element, and weight it by 2, and so on.

In order to calculate N_i, we notice that in our construction, once the i bins with exactly 1 element are chosen, every other bin has either 0 or ≥ 2 elements. We focus on the bins that have ≥ 2 elements. By a simple "reverse" pigeon hole argument, we can now have at most $\lfloor \frac{n-i}{2} \rfloor$ of the remaining bins containing any elements, as otherwise, one bin would be guaranteed to contain exactly 1 element.

So we further select any $1 \leq j \leq \lfloor \frac{n-i}{2} \rfloor$ number of the remaining $k - i$ bins all to contain at least 2 elements. Formally, this is equivalent to calculating the number of orientations of integers $\{x_l\}_{l=1}^{j}$ so that $x_1 + \cdots + x_j = n - i$ constrained with $x_l \geq 2 \; \forall l$

Now, we make a change of variables to instead calculate the number of orientations of integers $\{p_l\}_{l=1}^{j}$ so that $p_1 + \cdots + p_j = n - i - 2j$ constrained with $p_l \geq 0 \; \forall l$. Now

we just have a "stars and bars" problem with $k' = j$ bars and $n' = n - i - 2j$ stars. This reduces to:

$$\binom{n-i-j-1}{j-1},$$

orientations for the remaining bins with exactly j elements. Once these i bins with 1 element, and j bins with ≥ 2 elements are chosen, all the other bins are required to have 0 elements.

So adding these parts together, we have the expected amount data the server can expect to reconstruct perfectly becomes:

$$\frac{1}{\binom{k+n-1}{k-1}}\left[\sum_{i=1}^{n-2} i \cdot \binom{k}{i} \cdot \left(\sum_{j=1}^{\lfloor \frac{n-i}{2} \rfloor} \binom{k-i}{j}\binom{n-i-j-1}{j-1}\right)\right] + \overbrace{\frac{n}{\binom{k+n-1}{k-1}}\binom{k}{n} - \frac{n}{k}}^{r(n,k)},$$

We call the last two "residual" terms $r(n, k)$. The first of these terms corresponds to the term in the expectation where all elements of the batch end up in separate bins, and the second term is the expected number of elements that land in the tail of the CDF not covered in any bin. □

This can be thought of as a *lower* bound on privacy breaches since, often, identifiable information can be extracted from a mixture of two images. Note that increasing the expected number of perfectly reconstructed images requires increasing the number of imprint bins, which in turn requires increasing the number of channels of the matrix W_* and thus the number of parameters. Thus, to visualize the result above in terms of the server-side hyperparameter, k, we plot the expected proportion of data recovered as a

function of number of bins in fig. 3.1(a). Note that this inversion is analytic, and significantly more efficient and realistic than optimization based methods which often require tens of thousands of update steps.

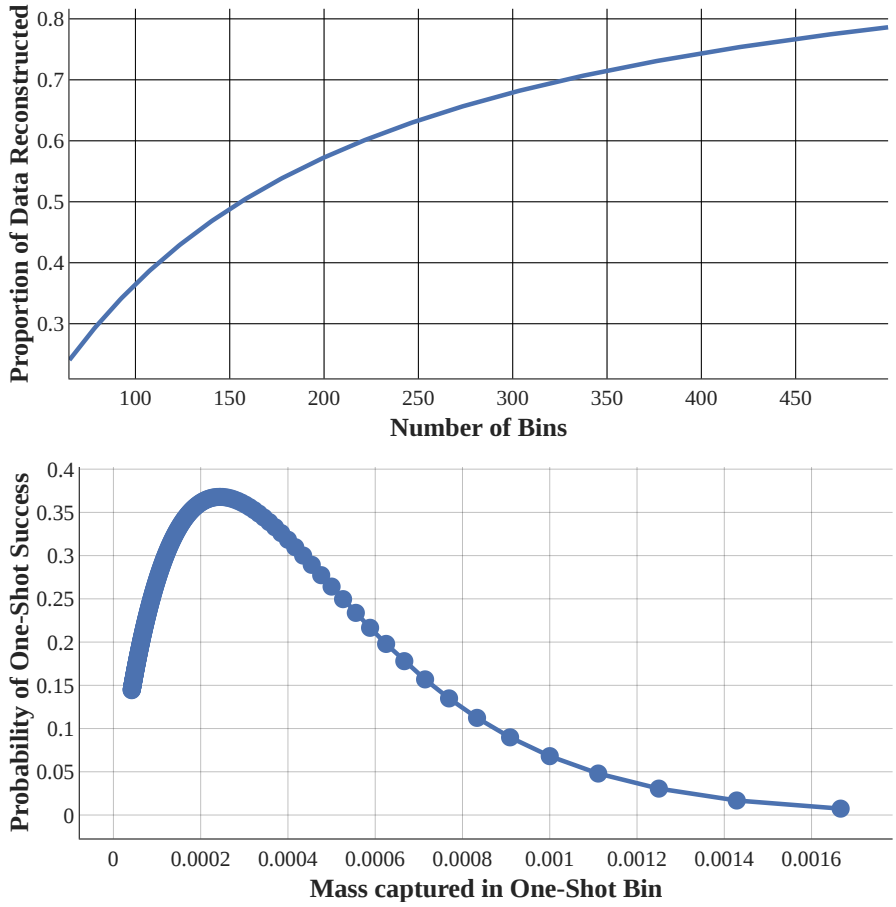


Figure 3.1: **Top (a)**: Expected proportion of a batch of 64 images *perfectly* recovered as a function of number of bins added via an imprint block in front of a ResNet-18 on ImageNet. With only 156 bins, an attacker can expect to recover over 50% of a batch of user images perfectly. **Bottom (b)**: Probability of a successful "one-shot" attack on a batch of 4096 images as a function of mass captured in the one-shot bin. An attacker can optimize their bin size given an expected batch size.

The imprint module as described can be inserted in any position in any neural network which receives a non-zero gradient signal. To recover the input feature dimension of the module, a second linear layer can be appended, or – if no additional parameters are of interest – the sum of the outputs of the imprint module can be added to the next

Figure 3.2: **Left:** Ground truth batch of 64 user images. **Right:** Analytic reconstruction for an imprint model with 128 bins in front of a ResNet-18. Gray reconstructions denote bins in which no data point falls.)

layer in the network. The binning behavior of the imprint module is independent of the structure of succeeding layers as long as any gradient signal is propagated. This flexibility allows for wide trade-offs between inconspicuousness and effectiveness of the imprint module. The module can be placed in later stages of model whereas an early linear layer might be suspicious to observers (now that they have seen this trick), but depending on the data modality, early linear layers can be a feature of an architecture anyway, in which case there is even no model modification necessary, only parameter changes. The parameter alterations necessary to trigger this vulnerability can furthermore be hidden from inspection until use. The layer can lay "dormant", functioning and training as a normal linear layer initialized with random activations, as long as the server desires. At any point, a party with access to the server can send out parameter updates containing W^*, b^* and trigger the attack.

3.4 Experiments

In the following section, we provide empirical examples of imprint modules. In all experiments we evaluate ImageNet examples to relate to previous work [169], but stress that the approach is entirely data agnostic. Given an aggregated gradient update, we always reconstruct as discussed in section 3.3. When, in the case of imprecision due to noise, more candidate data points are extracted than the expected batch size, only the candidates with highest gradient mean per row in W^* are selected and the rest discarded. For analysis, we then use ground-truth information to order all data points in their original order (as much as possible) and measure PSNR scores as well as Image Identifiability Precision (IIP) scores as described in Yin et al. [169]. For IIP we search for nearest-neighbors in pixel space to evaluate a model-independent distance - a more strict metric compared to IIP as used in Yin et al. [169]. All computations run in single floating point precision.

3.4.1 Full batch recovery

We begin with a straightforward and realistic case as a selling point for our method - user gradient updates aggregated over a batch of 64 ImageNet images. We modify a ResNet-18 to include an imprint module with 128 bins in front. This relatively vanilla setup presents is a major stumbling block for optimization based techniques. For example, for a much smaller batch size of 8, the prior art in optimization based batched reconstruction achieves only 12.93 average PSNR [169]. We do of course operate in different threat models (although Yin et al. [169] also uses non-obvious parameter modi-

fications). However, our imprint method is successfully able to recover almost perfect reconstructions of a majority of user data (see fig. 3.2), and achieves an average PSNR of 75.75. We further stress that a batch size of 64 is by no means a limitation of the method. If bins, and hence additional rows in W^* are added proportionally to the expected batch size, then recovery of significant proportions (see fig. 3.1) of batches of arbitrary size – albeit with the incurred cost in additional parameters for each row – is possible.

3.4.2 Privacy breaches in industrial-sized batches – One-shot Attacks

A breach in privacy can occur if even a *single* piece of user data is compromised and massively increasing the number of parameters might not be desirable for the server and could raise suspicion under inspection. However, there is a threatening modification of the imprint module in this case. If a server has access to enough users, then it becomes feasible for the server to start *fishing* for private data among all updates, and attempt to recover a single data point from each incoming batch of data. Attacks of this nature require only as many additional parameters as twice the size of a single piece of targeted user data, as only two bins are needed. For this, k bins are constructed initially, and then all bins are "fused" to create 2 final bins: a one-shot bin containing mass n/k, and the other containing the remaining mass $(n+1)/k$. For perspective, for a ResNet-18 on ImageNet, this would require only an additional 1% of parameters. And, based on proposition 3.3.1, it is always possible to select an optimal bin size, so that a data point is leaked on average once every four batches of incoming data, *no matter how large*. We demonstrate this statistical property by recovering a single image from an aggregated


Figure 3.3: **Left (a)**: A true user image from class "minibus". **Right (b)**: The reconstructed image from class "minibus" captured via our one-shot attack from averages aggregated over **16,384 datapoints**. The PSNR is 161.36, i.e. a verbatim copy at machine precision. This user could potentially be identified via their recovered license plate, which was blanked out (by us!) to preserve privacy.

batch of $2^{14} = 16,384$ ImageNet images (see fig. 3.3). Even though the gradient updates are averaged over a vast number of data points, there can be no perfect privacy, and one data point is leaked in its entirety by only a minor model modification.

3.4.3 Variants

Flexible placement The imprint module introduced in the previous section does not depend on its placement within a given model. No matter the position in a network, the incoming input features will be leaked to the server. Furthermore, the server can also change the parameters of preceding layers to represent (near)-identity mappings, allowing for the recovery of raw input data from inconspicuous positions deep in a network. For a convolutional network, we show examples of this strategy in fig. 3.14 for a ResNet-18. Here, the model parameters are manipulated to contain identity maps up to the location of the imprint module, while the downsampling operations remain. Even these later layers leak enough information that their inputs can be upsampled to the breach the privacy of the inputs. We remark that we show a simplified version of this attack here where the first three channels in each layer act as an identity, and all other channels are zero, but

the model can also be modified to provide off-set pixels in all other channels, effectively increasing the spatial resolution in any layer proportionally with the number of channels.

Multiple local updates: In several federated learning protocols, such as fedAVG, users take several local update steps on data before sending model updates to the server [85]. The imprint module is threatening when a large amount of user data is used for a *single* update step. In this case, the only variable that matters is amount of total data used in the update. That is to say, 10 users sending updates on 100 datapoints each is equivalent (recovery-wise) to a single user sending an update calculated on 1000 datapoints. However, when multiple steps are taken, inverting gradients from pairwise differences (as in eq. (3.4)) becomes more difficult, as entries in W^* shift with local updates. However, an imprint variant that produces sparse gradients per data point is a threat to such federated averaging. Defining a forward pass in this new variant as $M'(x)$ as $M'(x) = g(W_* x + b_*)$ where the non-linearity g is a thresholding function:

$$g(t) = \begin{cases} 0 & t \leq 0 \\ t & 0 \leq t \leq 1 \\ 1 & 1 \leq t \end{cases}$$

Note this non-linearity can be simply constructed with a combination of two ReLUs, or with an implementation of a Hardtanh. We now define W_*^i as: $\langle W_*^i, x \rangle = \frac{h(x)}{\delta_i}$ where h is the a linear function of the data as described before, and the biases are defined as:

$$b_*^i = -\frac{c_i}{\delta_i} \quad \text{where} \quad \delta_i = \Phi^{-1}(\frac{i+1}{k}) - \Phi^{-1}(\frac{i}{k}).$$

This setup creates sparse bins where inversion is directly possible (without taking pairwise differences) in gradient entries, at the cost of an additional activation layer. Analyzing a local update step with $W_*^{i,j}$ as the i^{th} row of W_* at update step j, reveals that

$$W_*^{i,j} = W_*^{i,j-1} - \alpha \frac{\partial \mathcal{L}}{\partial a^{i,j}} x_j$$

where x_j denotes the data from the previous batch that activated bin i, and $a^{i,j}$ denotes the i^{th} activation at step j, and α is the local learning rate. Note that if either a linear layer, or a convolutional layer follows this imprint module, then $\frac{\partial \mathcal{L}}{\partial a^{i,j}}$ does not depend on the scale of $W_*^{i,j}$. Therefore, a simple way to increase the effectiveness of the new imprint module, M' in the fedAVG case is to scale the linear function associated to the rows of W_* - i.e. $h'(x) = c_0 \cdot h(x)$. This "flattens" the distribution of values, and increases the relative size of $b_*^{i,j}$ compared to the gradient update $\frac{\partial \mathcal{L}}{\partial a^{i,j}}$, which prevents the bins from shifting too significantly during local updates. As bin shift goes to 0, we recover the situation where the only variable that matters is the total data used in an update.

With this modification, reconstruction quality remains similar to the fedSGD setting. For example, splitting the batch of 64 ImageNet images up into 8 local updates with learning rate $\tau = 1e-4$ yields an IIP score of 70.31%, due to minor image duplications where images hit multiple shifted bins, which we visualize in Appendix fig. 3.19.

Other data modalities: Yet another advantage to our imprint module over existing optimization based gradient inversion techniques is the flexibility in data domain. Other techniques have demonstrated some success in the image domain by leveraging strong regularizers including total variation (TV), image registration, matching batch norm stat-

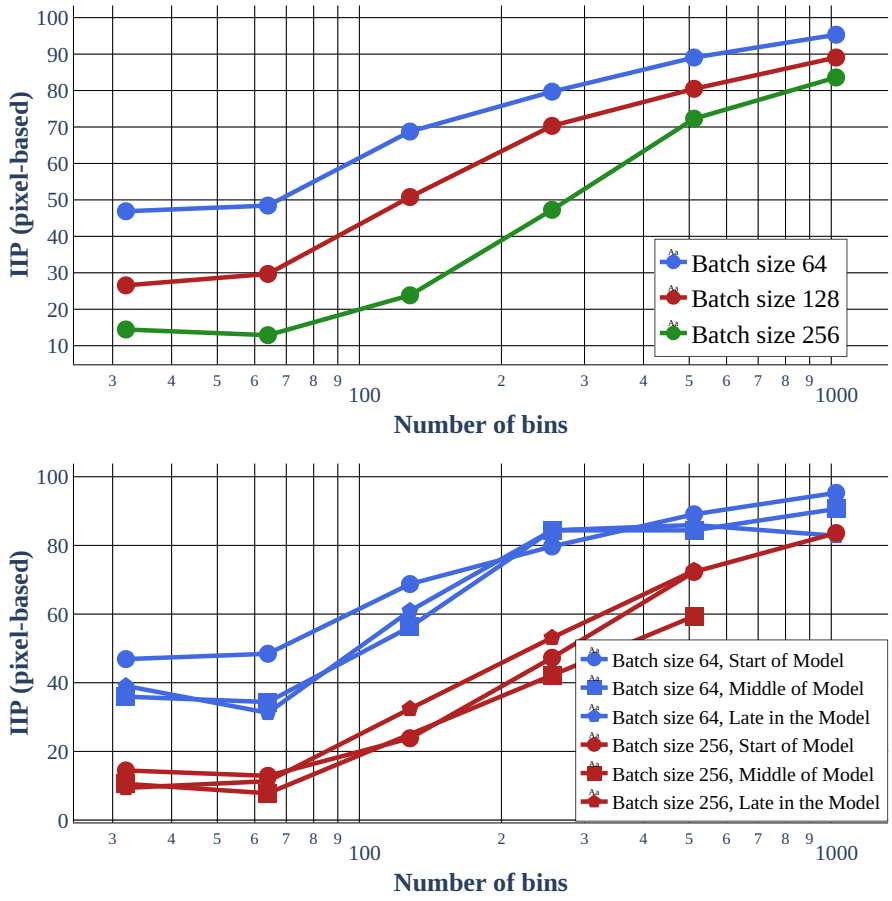


Figure 3.4: **Top:** Identification Success vs bin size. **Bottom:** Identification Success (via IIP score) vs. bin size and position in a ResNet-18 model. We find that the attack is stable over a range of batch sizes and positions in a model.

istics, and DeepInversion priors [46, 169]. Such strong regularizers do not always exist in other domains of interest, such as text or tabular data. The discussed imprint module, however, is data-agnostic, and while we focus our experiments on the image domain, nowhere do we use any assumptions unique to vision. In fact, linear layers often appear in language models, and tabular data models - cases in which the attacker only needs to modify parameters of an existing model to breach user privacy [147, 162] without architecture modifications.

3.4.4 Other Choices of Linear Functions and Distributions

In previous experiments we have restricted our investigations to the linear function $h : \mathbb{R}^m \to \mathbb{R}$ that measures average brightness, i.e. $h(x) = \frac{1}{m}\sum_{i=1}^{m} x_i$ which we approximate to be normally distributed. Given that ImageNet (and this also applies to most image datasets) is pre-processed by normalization by color in each channel, and that the number of pixels is large and they are not perfectly correlated, this is a reasonable approximation based on the central limit theorem that could similarly apply to other data modalities as well. For analysis, we visualize the closeness of this approximation based on an evaluation over the full ImageNet validation set in fig. 3.6.

We verify that the actual ground truth distribution can be approximated by a normal distribution, but we also see that the approximation is imperfect. The attack works well in fig. 3.2 even with this discrepancy, however it could be further improved if the attacker has more accurate about the CDF. Image brightness is better described by a Laplacian distribution [66, 136]. Replacing the normal distribution by a Laplacian distribution with

Linear Function	Assumed Distribution	MSE	PSNR	IIP-Pixel
Mean	Normal	0.0183	75.75	65.62%
Mean	Laplacian	0.0174	79.63	71.88%
Cosine	Laplacian	0.0167	99.82	79.69%
Random	Normal	0.0203	91.29	75.00%

Table 3.1: Ablation study linear functions and distributions.

scale $1/\sqrt{2}$ does improve the accuracy slightly. This distribution can be further stabilized by considering higher frequencies compared to the mean, e.g. via DCT coefficients [66, 89]. We accordingly also visualize this distribution for e.g. the 32nd DCT coefficient in fig. 3.6 and use this cosine wave for the imprint module (with scaling factor $\frac{4}{m}f$. This leads to the strongest attack against image data, but of course utilizes attacker knowledge that the users train on natural images.

On the flip side, the estimation can also be improved by replacing the linear function h with a Gaussian random vector of independent draws from $\mathcal{N}(0, \frac{1}{\sqrt{m}})$. The resulting distribution (4th figure in fig. 3.6) approximates a normal distribution much better. While not as optimal as the Laplacian distribution for higher frequencies, this variant is applicable for other data modalities if the data has bounded variance. Visualizations of the reconstruction with other linear functions can be found in fig. 3.7.

Finally, even with a small amount of data, the server could estimate the density of the quantity of interest. Visually, we plot the the estimated density as for several amounts of ImageNet data used to estimate the brightness distribution. We find that even with 0.1% of the data used, the server could obtain a close approximation to the distribution of interest (see fig. 3.5).

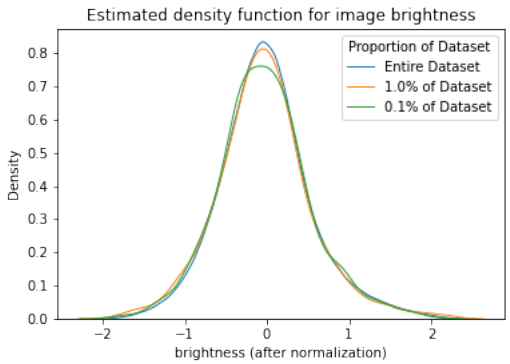
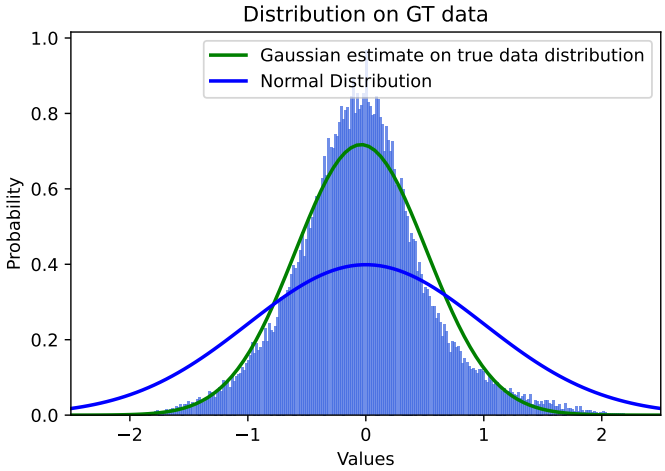


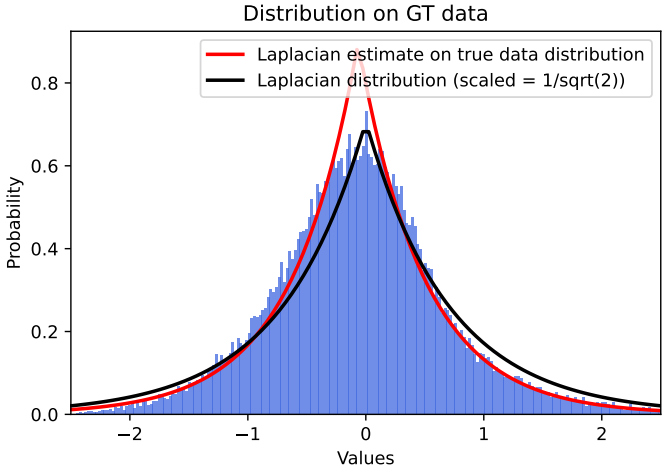
Figure 3.5: Density of image brightness estimated from access to different amounts of data from the ImageNet dataset.

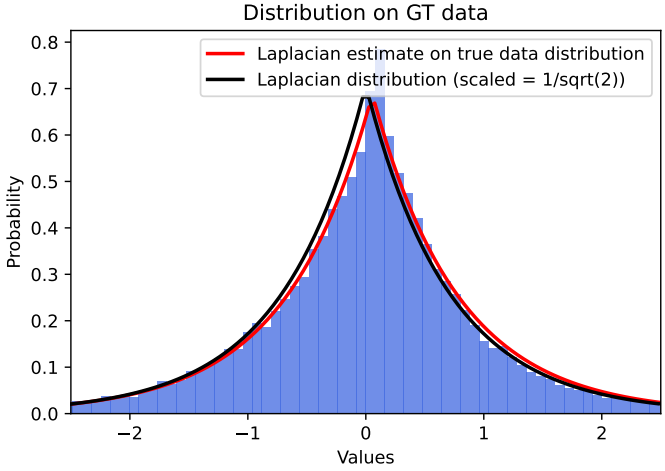
3.4.5 Comparison to Honest Servers and Optimization-based Attacks

We argue that the proposed attack operating in our threat model is significantly more threatening than existing optimization-based attacks in the honest-but-curious server model. To illustrate this point and show by example that the change is threat model which amounts to only a minor architectural change in the neural network leads to a massive difference in reconstruction, we run the attack of [46] in the scenario of fig. 3.2. The results can be found in fig. 3.8. The attack leads to an IIP score of 6.25% when measuring in pixel space, 6.25% when measuring in LPIPS [175] and, and 20.31% when measuring the cosine distances in feature space of this model (the metric of [169]).

As an additional ablation, we also investigate whether the optimization-based attacks can find the optimal solution in the malicious server threat model that we consider. However, the right side of fig. 3.8 shows that at least conventional optimization-based methods have trouble finding the vulnerability introduced by the imprint module. The vulnerability that the attack solves analytically might be hard to exploit by first-order optimization or require specifically tuned optimization schemes to succeed. This also shows


Distribution on GT data
Gaussian estimate on true data distribution
Normal Distribution
Probability
Values


Distribution on GT data
Laplacian estimate on true data distribution
Laplacian distribution (scaled = 1/sqrt(2))
Probability
Values


Distribution on GT data
Laplacian estimate on true data distribution
Laplacian distribution (scaled = 1/sqrt(2))
Probability
Values

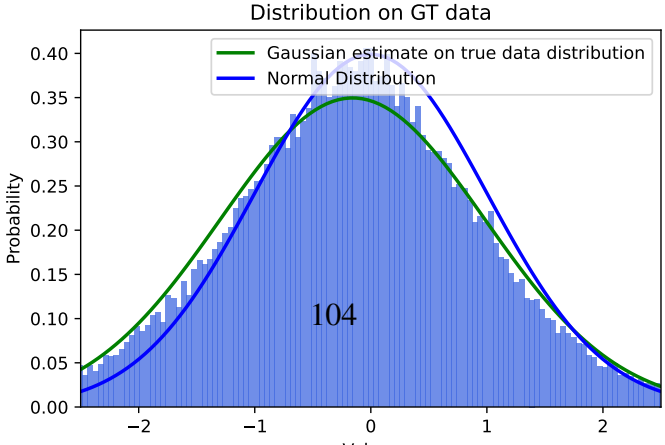

Distribution on GT data
Gaussian estimate on true data distribution
Normal Distribution
Probability

Figure 3.7: Analytic reconstruction for an imprint model with 128 bins in front of a ResNet-18. **Left:** The linear function is the 32nd DCT coefficient and bins are based on a Laplacian distribution. **Right:** Linear function is a Gaussian random vector and bins are based on a normal distribution. Gray reconstructions denote bins in which no data point falls.)

that defenses that attempt to detect privacy breaches by evaluating a range of optimization-based attacks would not have triggered an alarm for this attack.

3.4.6 Technical Details

All experiments were implemented in PyTorch [122] and were run on several laptop and machine CPUs, as the reconstruction itself requires only a few tensor operations. Especially, compared to optimization-based reconstruction techniques, this makes the approach significantly faster and significantly more portable. For visualization purposes and to measure accurate PSNR and IIP scores all images (which are recovered in the order given by the chosen function h) are matched with possible correspondences in the ground truth batch. The matching is found based on LPIPS feature similarities scores [175] which are matched using a linear sum assignment solver. No labels are recovered using the pro-

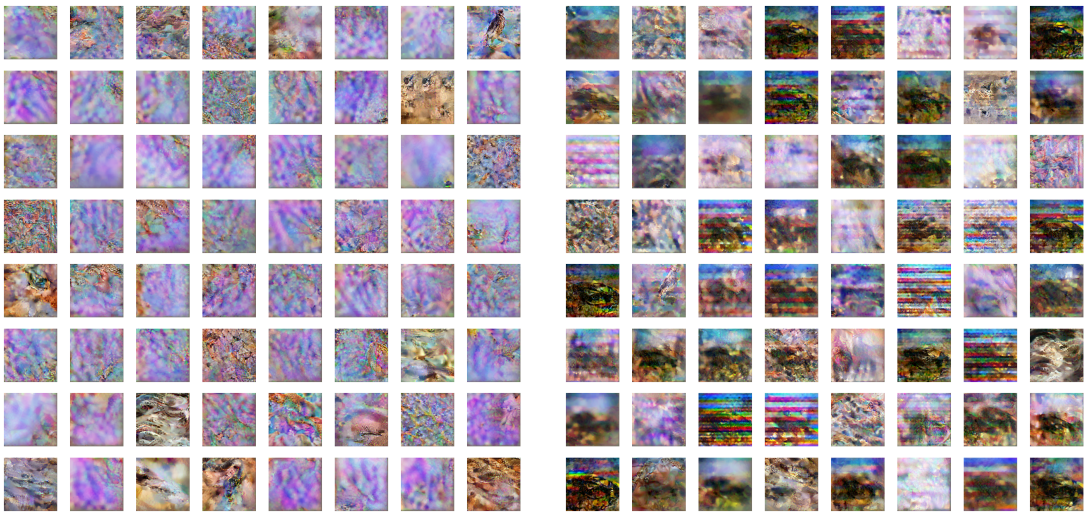


Figure 3.8: Optimization-based attack of Geiping et al. [46] for a ResNet-18 and a batch size of 64, the setting of fig. 3.2. **Left:** An honest server model. **Right:** Gradient inversion attack applied to a module that contains the imprint module.

posed approach which is entirely label-agnostic, but labels could be assigned a-posteriori using model predictions of the reconstructed data if required. For experiments where the imprint module is placed deeper into a network, the network (which is here a ResNet) is linearized by resetting batch normalization parameters and buffers to the identity map, setting all residual paths to zero and initializing the first convolution and the shortcut convolutions to identity maps. The nonlinearities can be bypassed by bias shifting as in [51].

PSNR scores are computed as average PSNR where we first compute PSNR scores per image and then average. This procedure is standard in computer vision, but does bias the score toward successful reconstructions, as the minimal PSNR score is bounded at 0, but its potential upside unbounded. For the image identifiability precision (IIP) score of Yin et al. [169] we implement the score as proposed therein, but measure nearest neighbors not in the model feature space (which we consider biased, given that the model para-

meters are already used for reconstruction), but directly in image pixel space, where we check whether the given reconstruction is indeed closer to its true counterpart in euclidean distance than any other image from this class in the validation set.

3.4.7 Additional Images

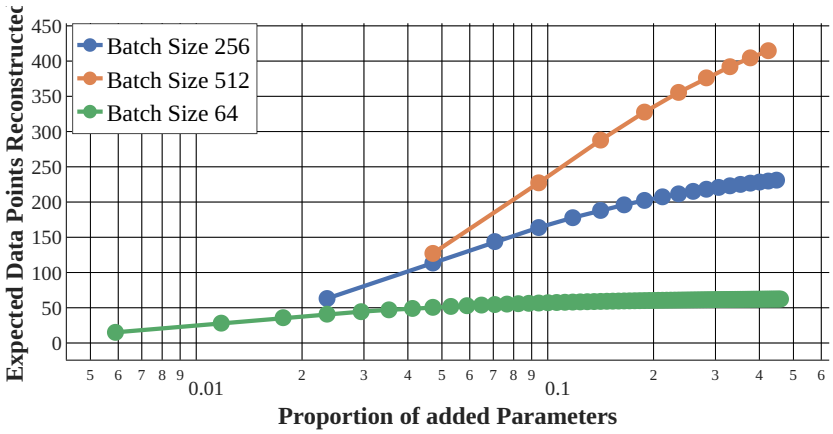


Figure 3.9: Expected number of data points recovered for several batch sizes and increased bins and corresponding parameter increase. Model: ResNet50 with ImageNet, targeting the input to the 3rd residual block.

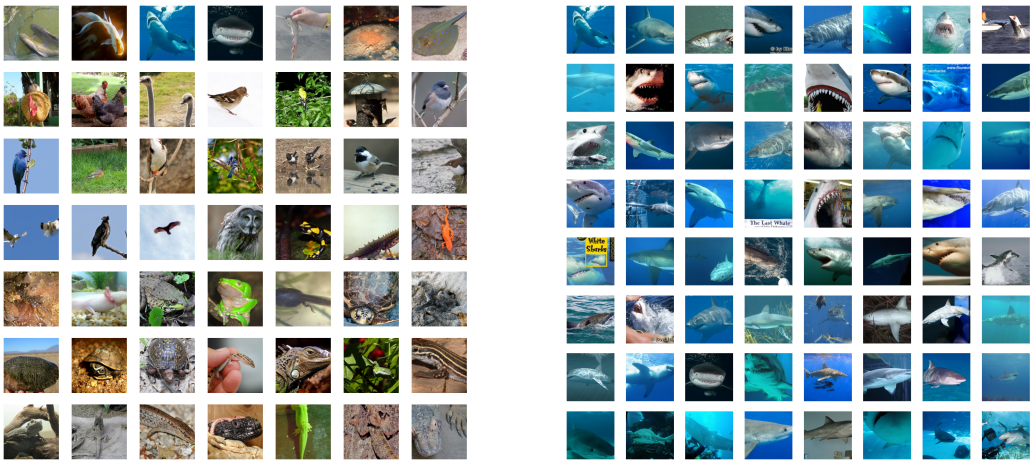


Figure 3.10: **Left:** Raw data for the 64 ImageNet images with separate classes. **Right:** Raw data for the 64 images from the white shark class.

This section contains additional image examples, such as using CIFAR-10 in fig. 3.12

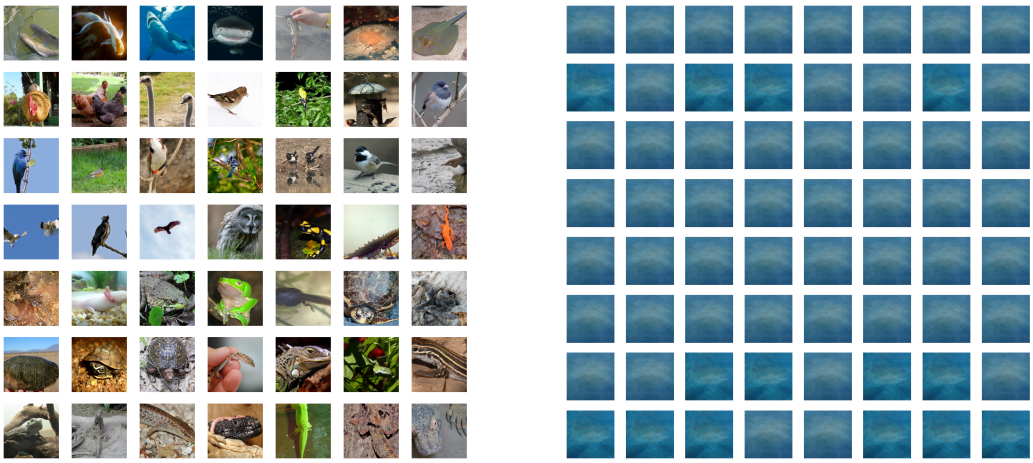


Figure 3.11: **Left:** Analytic reconstruction for a linear model of 64 ImageNet images with separate classes (PSNR: 36.45 versus true user data). **Right:** Same recovery algorithm but for 64 images from the same class (white shark), (PSNR: 13.84 versus true user data.)

Figure 3.12: **Left (a)**: A batch of 64 CIFAR10 images. **Right (b)**: The same batch of images reconstructed naively using eq. (3.2).

and fig. 3.13, as well as ImageNet examples from the same class (Figures 3.10, 3.11) where an attack with the imprint module on this dataset shows that almost all of the user data is perfectly recovered. Furthermore, example panels of imprint modules inserted in later ResNet layers are visualized as well as the results of the sparse variant that is used to attack a federated averaging scheme.

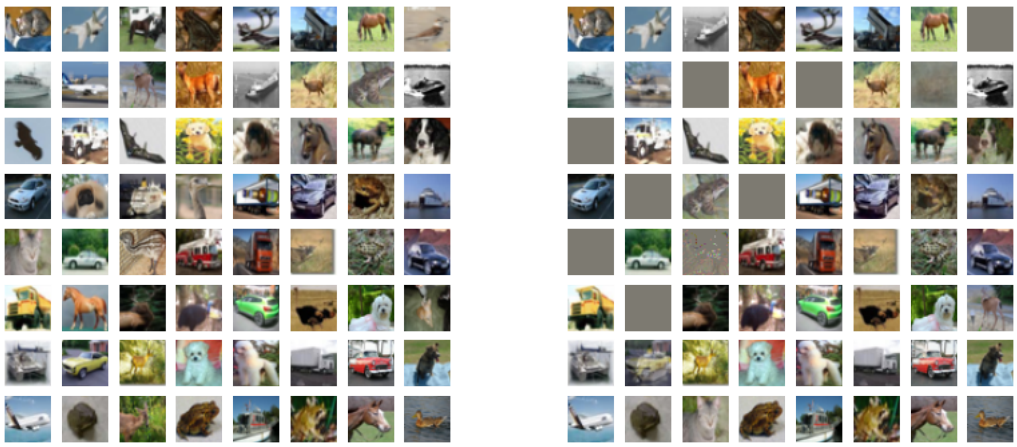


Figure 3.13: **Left (a)**: A batch of 64 CIFAR10 images. **Right (b)**: The same batch of images reconstructed using the imprint module with 300 bins. Gray images can result from collisions within a given bin.

Figure 3.14: Different placements of the imprint module in a ResNet-18. From left to right: Before the first block ($56x56$), before the third block ($28x28$), before the fourth block ($14x14$) and before the last average pooling ($7x7$). Compare to a placement before the first convolution ($224x224$) and raw input data in fig. 3.2. Enlarged versions of each panel can be seen in Figs. 3.15 - 3.18

3.4.8 Defense Discussion

An algorithmic defense against the proposed attack would be to validate the incoming model parameters on the user side. There, the attack with multiple bins requires repeated computations of the same quantity. At first, this pattern could be detected by rank analysis of all linear layers (which would return a rank of 1 for the linear layer of the imprint module described above). Yet, the attacker can easily randomize a few entries of the linear layer to increase its rank without significantly weakening the attack, or in-

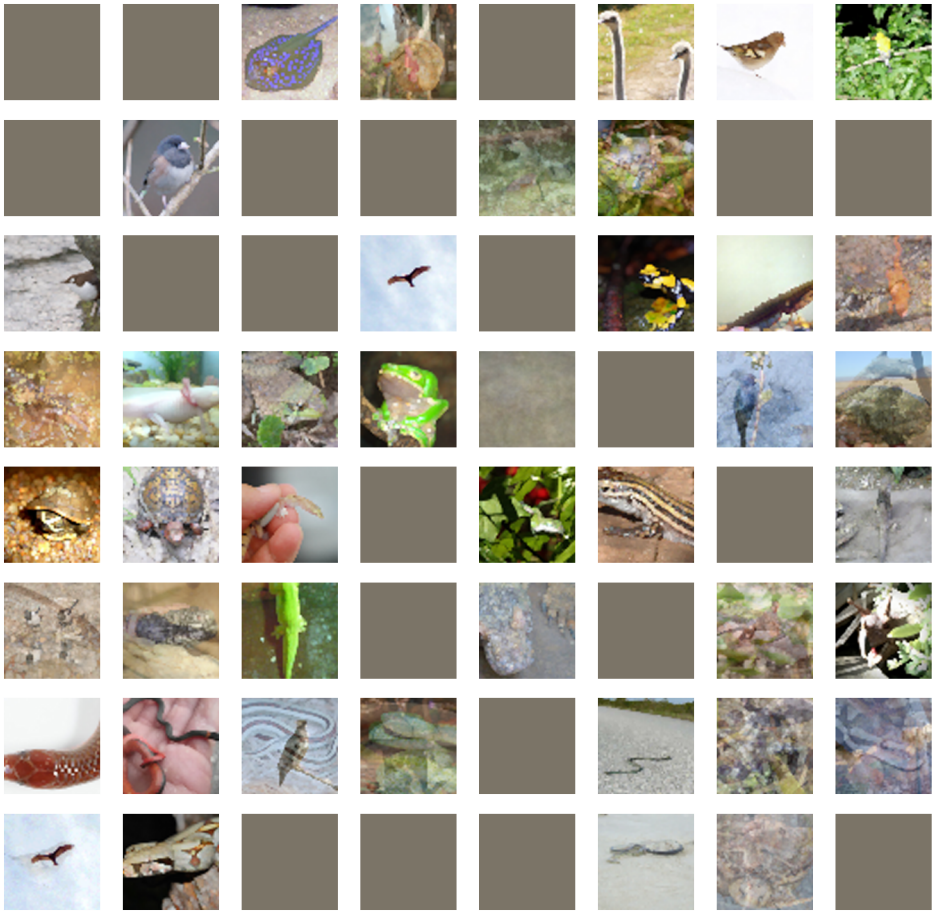


Figure 3.15: Different placements of the imprint module in a ResNet-18. Before the first block (56×56). Only the first three channels at this position are utilized in this demonstration.

troduce additional rows that compute normal deep features, so that we do not believe a defender can win by model analysis under the given threat model.

3.5 Potential Defense and Mitigation Strategies

If aggregation is the only source of security in a FL system, then the proposed attack breaks it, uncovering samples of private data from arbitrarily large batches, especially via the One-shot mechanism. In light of this attack, the effectiveness of secure aggregation is reduced to only *secure shuffling* [75]: When private data is uncovered via the imprint

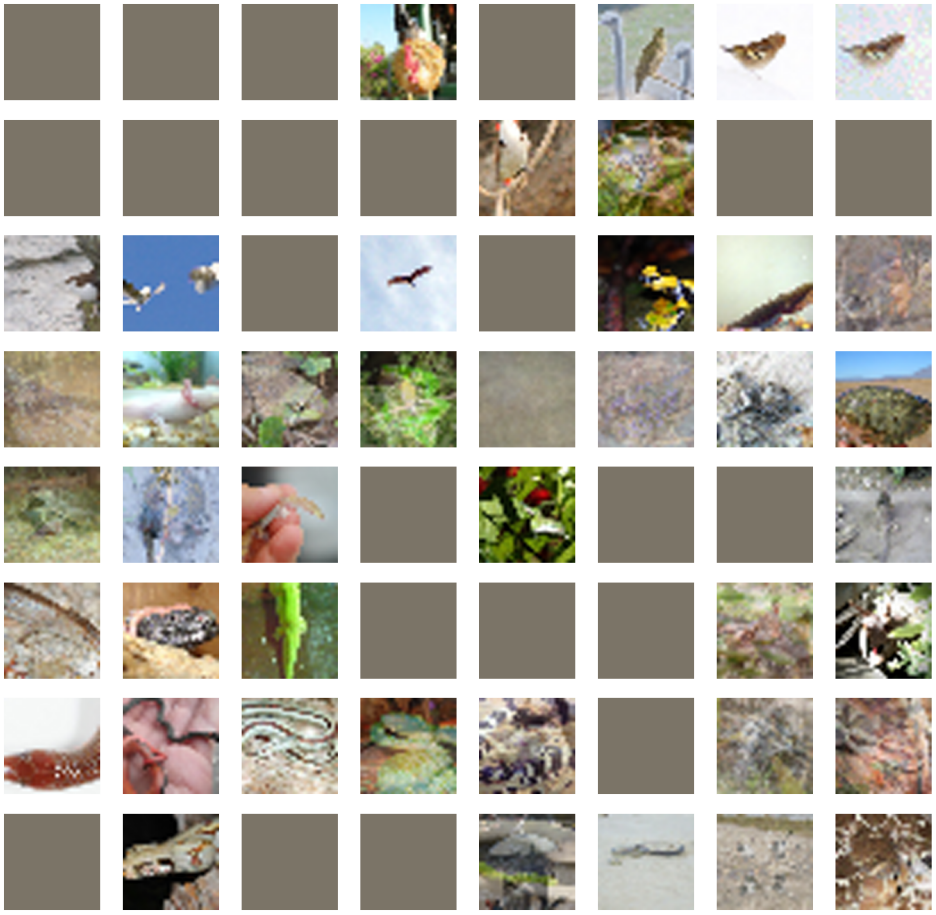


Figure 3.16: Different placements of the imprint module in a ResNet-18. Before the third block (28×28). Only the first three channels at this position are utilized in this demonstration.

module, based on data that has been securely aggregated, then the data is breached, but is not directly connected to any specific user (aside from possible revealing information in the data itself). A mitigation strategy for users that does not require coordination (or consent) of a central server is to employ local differential privacy [34]. Adding sufficient gradient noise can be a defense against this attack as the division in eq. (3.2) leads to potentially unbounded errors in the scale of the data. Yet, in practice, privacy is often still broken even if the correct scale cannot be determined, so that the amount of noise that has to be added is large. In fig. 3.20, even with $\sigma = 0.01$, private data is visibly leaked.

Figure 3.17: Different placements of the imprint module in a ResNet-18. Before the fourth block (14×14)

Additional discussion on defenses can be found in section 3.4.8

3.6 Conclusions

Federated learning offers a promising avenue for training models in a distributed fashion. However, the use of federated learning, even with large scale averaging, does not guarantee user privacy. Using common and inconspicuous machine learning modules, a malicious server can breach user privacy by sending minimally modified models and parameters in a federated setup. We hope that constructing these examples clarifies current limitations and informs discussions on upcoming applications, especially concerning

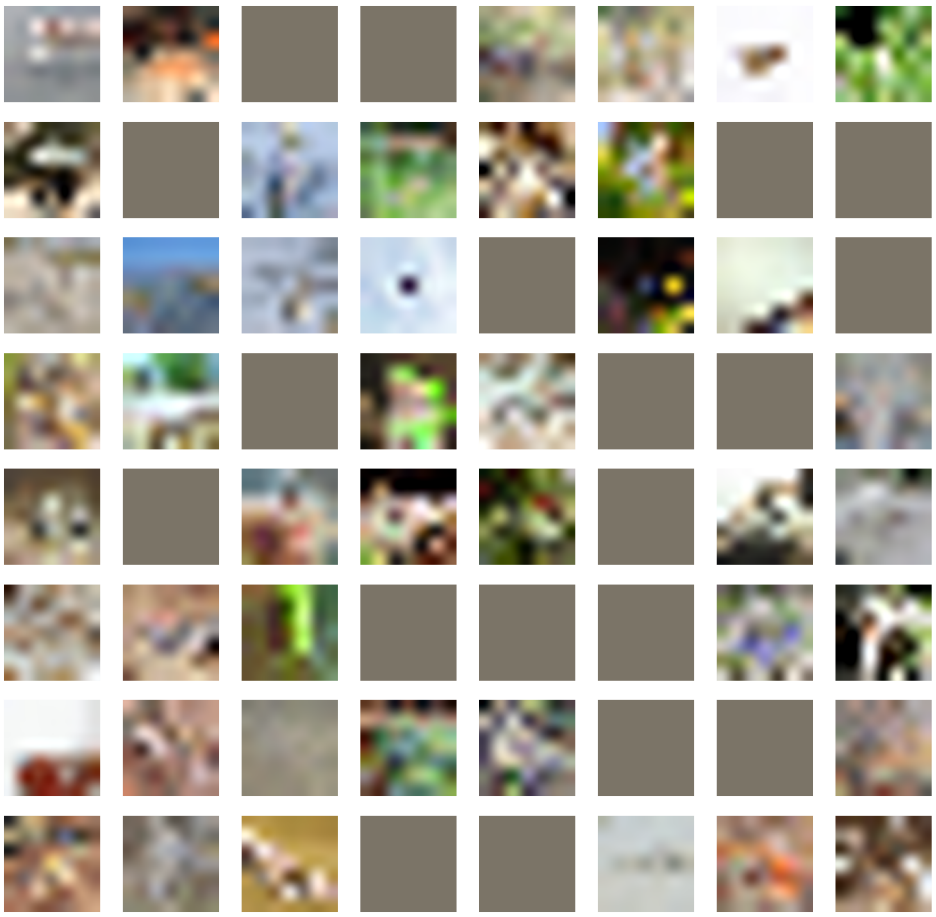


Figure 3.18: Different placements of the imprint module in a ResNet-18. Before the last average-pooling layer (7×7). Only the first three channels at this position are utilized in this demonstration.

API design.

Figure 3.19: Results for federated averaging for 8 steps with 8 images each, i.e. 64 unique data points for a single user, and 128 bins. PSNR: 32.65. IIP: 70.31%. Drift of bins during local updates leads to a few duplicated entries.

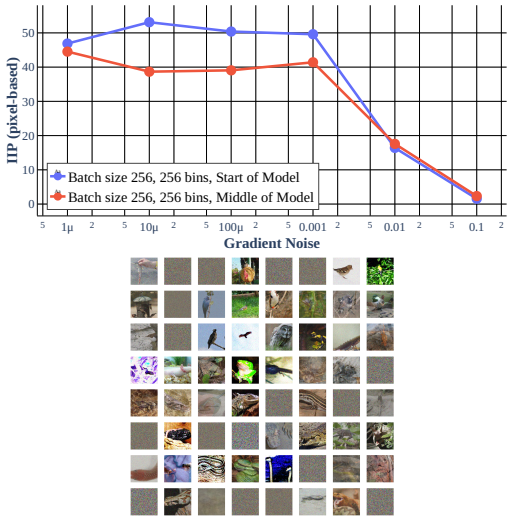


Figure 3.20: **Top:** IIP score vs Laplacian gradient noise. **Bottom:** Exemplary recovery for $\sigma = 0.01$. Recovery is stable for a large range of Laplacian gradient noise injections.

Chapter 4: Attacking Federated Learning for Text

A central tenet of Federated learning (FL), which trains models without centralizing user data, is privacy. However, previous work has shown that the gradient updates used in FL can leak user information. While the most industrial uses of FL are for text applications (e.g. keystroke prediction), nearly all attacks on FL privacy have focused on simple image classifiers. We propose a novel attack that reveals private user text by deploying malicious parameter vectors, and which succeeds even with mini-batches, multiple users, and long sequences. Unlike previous attacks on FL, the attack exploits characteristics of both the Transformer architecture and the token embedding, separately extracting tokens and positional embeddings to retrieve high-fidelity text. This work suggests that FL on text, which has historically been resistant to privacy attacks, is far more vulnerable than previously thought. This work was conducted with Jonas Geiping, Steven Reich, Yuxin Wen, Wojtek Czaja, Micah Goldblum, and Tom Goldstein. My contributions include jointly conceiving of the mechanism of position and token matching, as well as the attention mechanism manipulation. Additionally, I implemented the first version of the attack, and wrote a substantial portion of the manuscript.

4.1 Introduction

Federated learning (FL) has recently emerged as a central paradigm for decentralized training. Where previously, training data had to be collected and accumulated on a central server, the data can now be kept locally and only model updates, such as parameter gradients, are shared and aggregated by a central party. The central tenet of federated learning is that these protocols enable privacy for users [56, 102]. This is appealing to industrial interests, as user data could be leveraged to train machine learning models without user concerns for privacy, app permissions or privacy regulations, such as GDPR [159, 163]. However, in reality, these federated learning protocols walk a tightrope between actual privacy and the appearance of privacy. Attacks that invert model updates sent by users can recover private information in several scenarios [126, 165]. Optimization-based inversion attacks have demonstrated the vulnerability of image data when only a few datapoints are used to calculate updates [46, 169, 180]. To stymie these attacks, user data can be aggregated securely before being sent to the server as in Bonawitz et al. [12], but this incurs additional communication overhead, and as such requires an estimation of the threat posed by inversion attacks against specific levels of aggregation, model architecture, and setting.

Most of the work on gradient inversion attacks so far has focused on image classification problems. Conversely, the most successful industrial applications of federated learning have been in language tasks. There, federated learning is not just a promising idea, it has been deployed to consumers in production, for example to improve keystroke prediction [58, 132] and settings search on the Google Pixel [13]. However, attacks against

this area have so far succeeded only on limited examples of single sequences with few (< 25) tokens [30, 180], even for massive models such as BERT (with worse recovery for smaller models) [32], leaving the impression that these models are hard to invert, and limited aggregation is already sufficient to protect user privacy.

In this work, we systematically re-investigate the privacy of these applications. We focus on the realistic threat model of an untrusted server update sent to users, and show that such a malicious update sent by the server can completely corrupt the behavior of user-side models, coercing them to spill significant amounts of user data. The server can then collect the original words and sentences used by the user with straightforward statistical evaluations and assignment problems.

Overall, we find that Transformer models can be reprogrammed to reveal astonishing amounts of private user information, and we show for the first time that recovery of all tokens and most of their absolute positions is feasible even on the order of *several thousand* tokens and even when applied to small models only 10% the size of BERT. Furthermore, we reveal the first attack which succeeds in the setting of updates averaged over multiple users, a setting where previous attacks fall woefully short.

4.2 Motivation and Threat Model

At first glance, updates from Transformer architectures might not appear to leak significant amounts of user data. Both the attention mechanisms and the linear components learn operations that act individually on tokens, so that their gradients are naturally averaged over the entire length of the sequence which is usually significant (e.g. 512

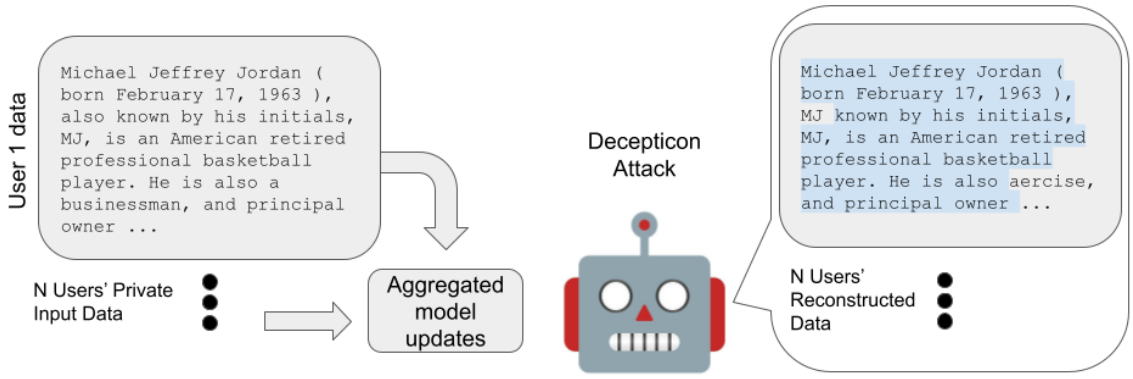


Figure 4.1: An example reconstruction from GPT-2 model using our proposed Decepticon. Using the Decepticon attack, an attacker can recover tens-of-thousands of tokens and positions from private user data.

tokens). Despite most architectures featuring large linear layers, the mixing of information reduces the utility of their content to an attacker. In fact, the only operation that "sees" the entire sequence, the scaled dot product attention, is non-learned and does not leak separate gradients for each entry in the sequence. If one were to draw intuition from vision-based attacks, gradients whose components are averaged over 512 images are impossible to invert even for state-of-the-art attacks [169].

On the other hand, the task of language modelling appears much more constrained than recovery in vision settings. The attacker knows from the beginning that only tokens that exist in the vocabulary are possible solutions and it is only necessary to find their location from a limited list of known positions and identify such tokens to reconstruct the input sequence perfectly.

Threat Model With this intuition, we consider the threat model of an untrusted server that is interested in recovering private user data. Both user and server are bound by secure implementations to follow the federated learning protocol, and the model architecture is compiled into a fixed state and verified by the same implementation to be a standard

Transformer architecture. The user downloads the server update and returns their model updates according to protocol.

We believe that this threat model is the most natural setting from a user perspective. Stronger threat models would, for example, allow the server to execute arbitrary code on user devices, but such threats are solvable by software solutions such as secure sandboxing [44]. Still other work investigates malicious model architectures [11, 43], but such malicious modifications have to be present at the conception of the machine learning application, before the architecture is fixed and compiled for production [13], making this attack most feasible for actors in the "analyst" role [75]. Ideally, user and server communicate through an open-source protocol implementation that only allows pre-defined and vetted model architectures. However, none of this stops malicious server updates. Such attacks are naturally ephemeral - the server can send benign server updates nearly all of the time to all users, then switch to a malicious update for single round and group of users (or single secure aggregator), collect user information, and return to benign updates immediately after. The malicious update is quickly overwritten and not detectable after the fact. Such an attack can be launched by any actor with temporary access to the server update, including, aside from the server owner, temporary breaches of server infrastructure, MITM attacks, single disgruntled employees or a sell-off or breach of the server's app which users initially trusted with FL rights.

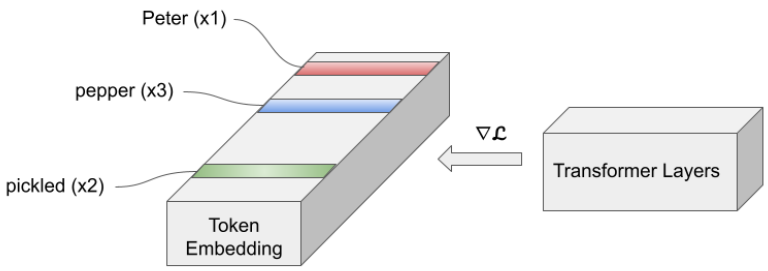


Figure 4.2: A high-level schematic of token leaking. The token embedding layer can leak tokens and frequency solely through its gradient entries.

4.3 Method - How to Program a Corrupted Transformer

The attack strategy we investigate in this work is comprised of three primary parts. We describe how the parameters of common Transformer architectures can be reprogrammed to deceive the user and encode their private information into gradient updates. Each component of the attack builds off the previous mechanism and allows the reconstruction of an increasingly compromising volume of user data. We first show how to recover tokens, then positions, then sequences from a corrupted transformer model.

4.3.1 Getting Tokens

Even without any malicious parameter modifications, an attacker can immediately retrieve the bag-of-words (the unordered list of all tokens and their assorted frequencies) of the user data from the gradient of the token embedding. Melis et al. [107] previously identified that unique tokens can be recovered due to the sparsity of rows in the token embedding gradient (c.f. Figure 4.2). But, perhaps surprisingly, even the frequency of all words can be extracted. Depending on the model architecture, this frequency estimation can either be triggered by analyzing the bias gradient of the decoder (last linear) layer, or

the norm of the embedding matrix. In both cases, the update magnitude is approximately proportional to the frequency of word usage, allowing for a greedy estimation by adapting the strategy of Wainakh et al. [164] which was originally proposed to extract label information in classification tasks.

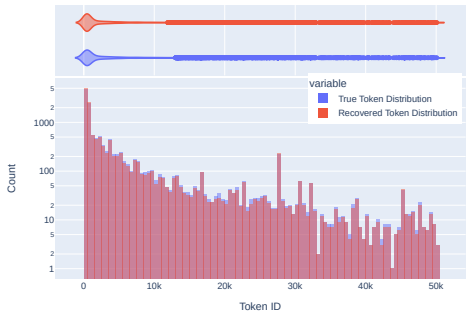


Figure 4.3: Distribution and histogram of Bag-of-words accuracy for the small Transformer with decoder bias. Out of 13824 tokens from *wikitext*, the estimated tokens leaked from the embedding layer are 96.7% accurate, unique tokens are 99.8% accurate.

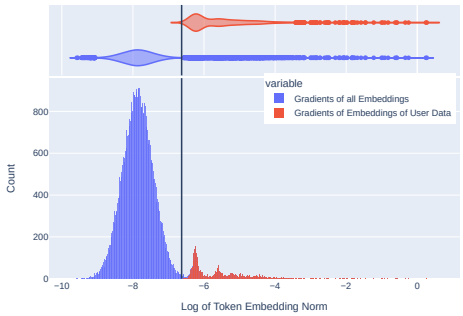


Figure 4.4: Distribution and Histogram of log of norms of all token embedding gradients for GPT-2 for 13824 tokens. Note how all embedding row gradients are non-zero, due to the tied encoder-decoder structure, but that the embeddings of true user data (red) are clearly separable from the mass of all embeddings (blue) by a simple cutoff at 1.5 standard deviations (marked in black).

For models which contain decoder bias, this estimation leads to a $> 99\%$ accurate estimate of word frequencies as shown in Figure 4.3. For models without decoder bias, such as GPT-2 [131], the same strategy can be employed based on the embedding norms. Additionally, we can accurately predict token frequency even for models which tie their embedding and decoding weights, and as such do not feature naturally sparse embedding gradients. We accomplish this by estimating the distribution of row norms in log-space, and thresholding unused token rows. As visualized in Figure 4.4, the estimation is inexact due to the cut-off, but still reaches 93.1% accuracy for GPT-2 (c.f. Figure 4.16).

This first insight already appears to have been overlooked in previous attacks [30] and does not require any malicious modifications of the shared model. However, for a model update averaged over multiple users and thousands of tokens, this recovery alone is less of a threat to privacy. Naively, the ways to order recovered tokens scale as $n!$, and any given ordering of a large amount of tokens may not reveal sensitive information about any constituent user involved in the model update.

4.3.2 Getting Positions

As stated previously, even for a single sequence, the number of possible orderings of leaked tokens means that for an attacker, breaching user privacy with information only about leaked tokens becomes intractable. However, once token embeddings are fed into a Transformer-based model, information is added about the relative position of each token - either through a learned positional encoding, or a fixed one as in the original Transformer architecture.

This motivates a possible attack wherein such information is used to rearrange the leaked tokens into a high-fidelity reconstruction. However, this positional information does not appear in the rows of the gradient entry for the token embedding. Thus, any positional information about the user sequence must be extracted from another part of the model.

To this end, we leverage the numerous large linear layers found in the feed-forward blocks of a Transformer-based model, combined with recent strategies for separating gradient signals. We denote a forward pass on the first layer (of the feed-forward com-

ponent) of the i^{th} Transformer block by $\phi_1^i = W_{1,i}x + b_{1,i}$. It is straightforward to derive that for a single embedding vector x passed through ϕ_1^i, for the j^{th} row of the linear layer, $W_{1,i}^j$, as long as $\frac{\partial \mathcal{L}}{\partial b_{1,i}^j} \neq 0$, then $\nabla_{W_{1,i}^j} \mathcal{L} / \frac{\partial \mathcal{L}}{\partial b_{1,i}^j} = x$. Simply put, linear layers can easily encode embedding vectors in their gradient information.

Then, modifying the strategy found in Fowl et al. [43], and Chapter 3, we sample a Gaussian vector $m \sim \mathcal{N}(\vec{0}, \mathbb{I}_d)$ where $d = d_{model}$, and identically set each row j of the weights of $\phi_{1,j}^i = m$, and the biases of this layer we set to $b_1^i = [c_{i \cdot k}, \ldots, c_{(i+1) \cdot k}]$. Here, k is the dimension of the codomain of ϕ_1^i, and $c_j = -\Phi^{-1}(\frac{j}{M})$ for Φ^{-1}, the inverse of the standard Gaussian CDF, and M is the sum over all Transformer blocks of the dimension of the codomain for ϕ_1^i. Note that we can very well estimate the mean and variance of this quantity (needed to normalize it) from known text datasets (see Figure 4.11). Then, when $\langle x, m \rangle$ uniquely falls between two values $[c_{j-1}, c_j]$, the gradient of the corresponding linear layer with those biases contains unique information that can be inverted to recover the embedding x.

However, this is only part of the battle (for the attacker), as the recovered embeddings themselves do not naturally reveal any private information. To achieve this goal, the attacker must associate both a token and position to each recovered embedding. We show that an attacker can accomplish this by first solving a linear sum assignment problem on the recovered embeddings and known positional embeddings. As seen in Figure 4.5, the attacker can calculate the correlation between the recovered embeddings, and "dummy" positional vectors, given that the sequence length of the model is known. This alone can give the attacker information about the most likely position for a recovered embedding. Then, the attacker can simply subtract the positional encoding from recovered embedding

to (almost) recover the token embedding that comprises the recovered embedding. This process is muddied by layer normalization, but the "faux" token embeddings that are recovered in this manner are enough to perform *another* linear sum assignment problem to associate each recovered embedding to a token recovered from the token embedding layer's gradient as in Figure 4.3.1. In a similar fashion, the attacker calculates correlations between the token embeddings and the "faux" token embeddings (the recovered embeddings minus the dummy positional encodings). After these two assignment problems, the attacker has retrieved a set of tokens, and their corresponding positions. Note that the attacker can set the second linear layer in the feed-forward portion of a Transformer block to collapse the outputs of the first linear layer into a small quantity to be added with the residual connection to a single entry for the original embeddings, thus allowing gradients to flow to each linear layer, while also not perturbing the embeddings too much.

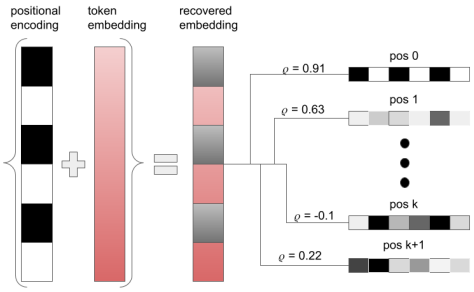


Figure 4.5: A high-level schematic for our position assignment. We find correlations ρ between known positions and recovered embeddings, and solve a linear sum assignment problem to determine which position is most likely for the recovered embedding.

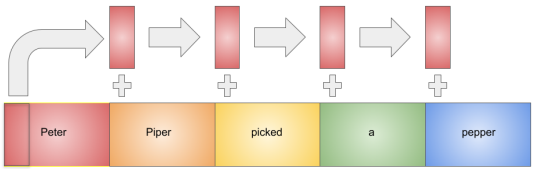


Figure 4.6: A high-level schematic of our MHA manipulation. The MHA block attends to the first word identically for every input sequence, encoding a part of the first token for each embedding in the entire sequence.

4.3.3 Getting Sequences

Recovering multiple sequences - either from user data comprising multiple separate sentences, or aggregates of multiple users - presents the most difficult task for the attacker. Naively using the strategy from Section 4.3.2 can only recover a partially ordered set of tokens. For example, if a model update consists of five user sequences, then the attacker recovers five first words, five second words, and so on. However, grouping these words into salient sentences quickly becomes intractable as the number of possible groupings grows as n^l for n users sending updates on sequences of length l. Further complicating matters for the attacker is that no learned parameters (and thus no parameters returning gradients) operate on the entire length of a sequence in the Transformer model. In fact, the only interaction between embeddings from the same sequence comes in the scaled dot product attention mechanism. Thus, if any malicious modification can be made to encode sequence information into embeddings, it must utilize the attention mechanism of Transformers.

With this as motivation, we uncover a straightforward mechanism by which an attacker could recover user sequences, even when model updates are aggregated over a large number of separate sequences. Let W_Q, W_K, W_V represent the query, key, and value weight matrices respectively, and let b_Q, b_K, b_V represent their biases. For simplicity of presentation, we explain this attack on a single head, but it is easily adapted to multi-head attention. We first set the W_K matrix to the identity ($\mathbb{I}_{d_{model}}$), and $b_K = \vec{0}$. This leaves incoming embeddings unaltered. Then, we set $W_Q = \vec{0}$, and $b_Q = \gamma \cdot \vec{p}_0$ where $\vec{p}_0$ is the first positional encoding. Here we choose the first position vector for simplicity, but there

are many potential choices for this bias vector. This query matrix then transforms each embedding identically to be a scaled version of the first positional encoding. We then set $W_V = \mathbb{I}_{d'}$ to be a "partial" identity matrix (identity in the first d' entries where $d' \leq d$ is a hyperparameter that the server controls). Finally, we set $b_v = \vec{0}$.

Now, we investigate how these changes transform an incoming sequence of embeddings. Let $\{x_i\}_{i=0}^{l-1}$ be embeddings for a sequence of length l that enters the attention mechanism. x_0 is the embedding corresponding to the first token in the sequence, so x_0 is made up of the token embedding for the first token in the sequence, and the first positional encoding. W_K produces keys K that exactly correspond to the incoming embeddings, however, W_Q, b_Q collapses the embeddings to produce Q, consisting of l identical copies of a single vector, the first positonal encoding. Then, when the attention weights are calculated as:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

the attacker finds that the first embedding dominates the attention weights for all the embeddings, as the query vectors all correlate with the first embedding the most. In fact, the γ parameter can effectively turn the attention weights to a delta function on the first position. Finally, when the attention weights are used to combine the values V with the embeddings, by construction, a part of the embedding for the first word in the sequence is identically added to each other word in that sequence. So the embeddings are transformed as $\{x_i\}_{i=0}^{l-1} \rightarrow \{x_i + x_{0,d'}\}_{i=0}^{l-1}$ where $x_{0,d'}$ is a vector where the first d' entries are the first d' entries of x_0, and the other $d_{model} - d'$ entries are identically 0.

If the attacker chooses to perform this modification on the first Transformer block,

this means that embeddings that the attacker recovers from each of the linear layers (as detailed in Section 4.3.2) now contain unique information about the sequence from which they came. The attacker can then calculate correlations between the first part of each recovered embedding and dynamically threshold these correlations in order to group each embedding into a sequence with other embeddings.

4.3.4 Putting It All Together

Now that we have introduced the three main mechanisms for our proposed attack, we will describe the procedure from start to finish for the attacker and introduce additional mechanisms that are useful for the attack. The attack begins after a user or aggregate group of users has computed their model update based on the corrupted parameters, at which point the server retrieves their update and begins the inversion procedure.

The server first computes normalized un-ordered embeddings $E_{\text{unordered}}$ based on the bag-of-words from Section 4.3.1, and normalized positional embeddings $E_{\text{positions}}$ based on the known sequence length (normalizing by the first layer norm, as the linear layers appear after the layer norm operation). The server then computes the actual breached embeddings from the bins inserted into the linear layers. All gradient rows in these linear layers record the sum over all embeddings whose inner product with the measurement vector is smaller than the bias of the row c_i. Conversely, by subtracting subsequent rows, the server recovers in each row the embedding with inner product within the range $[c_i, c_{i+1}]$. All rows with non-zero bias now correspond to such bins which contain 1 or more embeddings. Dividing these embeddings by their bias recovers a number of breached em-

Algorithm 2 Decepticon Data Readout

Input: Transformer model T, malicious server update θ_0, user update θ_1, number of expected sequences n.
$E_{\text{unordered}}$ = Embeddings of estimated bag-of-words of leaked tokens
E_{breached} = Breached Embeddings $\frac{\nabla_{W_i^j} - \nabla_{W_{i+1}^j}}{\nabla_{b_i^j} - \nabla_{b_{i+1}^j}}$ for linear layers $j = 1, ..L$ and rows $i = 1, ..., r$.
L_{batch} = Batch label for each $E_{\text{breached}}[: v]$ from clustering into n clusters via dynamic thresholding.
$E_{\text{positions}} \leftarrow$ Known positional embeddings
for b in $0...B$ **do**
    E^b_{ordered} = Match($E_{\text{positions}}[v :]$, $E_{\text{breached}}[L_{\text{batch}} = b, v :]$)
end for
E_{ordered} = concatenate $\{E^b_{\text{ordered}}\}_{b=1}^{B}$
while empty rows in E_{ordered} **do**
    $L_{\text{free positions}}$ = indices of empty rows in E_{ordered}
    E_{ordered} =Match($E_{\text{positions}}[L_{\text{free positions}}]$, $E_{\text{breached}}[: v]$
end while
$E_{\text{ordered, no pos.}}$ = E_{ordered} - $E_{\text{positions}}$
$T_{\text{final tokens}}$ = Indices of Match($E_{\text{ordered, no pos.}}$, $E_{\text{unordered}}$)

beddings E_{breached}, which correspond directly to the input of the linear layers (although some recoveries may be mixtures of multiple embeddings).

The first d' entries of each embedding encode the sentence as described in Section 4.3.3. The server can hence label each of the embeddings E_{breached} uniquely and assign them to their sentence by clustering. We describe a simple algorithm to cluster these embeddings into n groups and find their label in the Section 2.3.3 (based on dynamic thresholding and linear assignments), but other strategies such as constrained K-means are also possible [16].

We can now proceed for each sequence separately and match all found embeddings for the sentence to the known positional embeddings $E_{\text{positions}}$, thereby ordering the embeddings E_{breached} for each sentence. Due to mixtures, some positions will not be filled after this initial matching step. We thus iteratively fill up free positions with the

best-matching embeddings from E_{breached}, even if they have been used before in other positions. After this procedure all entries in E_{ordered} will be filled with entries from E_{breached}. We can then compute $E_{\text{ordered, no pos.}}$ by subtracting the positional encodings from these ordered embeddings, which should be an estimate of the token embedding at the given position. Finally we match $E_{\text{ordered, no pos.}}$ with the embeddings of tokens recovered in the beginning $E_{\text{unordered}}$ to recover the most likely identity of each token in the sequence. The attack is summarized in Section 2.

In general, this process works very well to recover even several batches of long sequences. In the limit of more and more tokens the quality eventually degrades as more and more breached embeddings are mixtures of multiple inputs, making the identification of their position and token id less certain. This is partially compensated for by the iterative filling of missing positions. When in the regime of a massive number of tokens, only a subset of positions are accurately recovered, and the remaining tokens are inaccurate, appearing in wrong positions in the input. This process is semi-random, so that at even for a massive number of tokens there will still exist some batches which can be recovered very well.

4.4 Empirical Evaluation of the Attack

In this section, we evaluate the threat of the attack discussed in Section 4.3 and evaluate its efficacy on a range of Transformer architectures. Our main focus is the application to next-word prediction as emphasized in practical use-cases [58, 132]. We consider three model architectures of differing sizes: first, the small 3-layer Transformer architecture

	Batch Size = 1	Batch Size = 8	Batch Size = 16
Length 32	Ancient Egyptian deities Egypt the gods and goddesses worshipped. ancient gods are The beliefs of rituals surrounding these in	Ancient Egyptian deities are the gods and goddesses worshipped in ancient Egypt ph The beliefs and rituals surrounding these gods	Ancient for deities are the gods and goddesses worshipped in ancient Egypt. The beliefs and rituals surrounding these gods
Length 128	Ancient Egyptian deities are the gods and goddesses worshipped Egypt ancient constitu. The beliefs and rituals myths these gods	Ancient Egyptian deities are the gods and goddesses worshipped in ancient Egypt. The beliefs view rituals surrounding these gods	Ancient Egyptian deities are the gods and goddesses worshipped in ancient Egypt. The beliefs view rituals surrounding these continue
Length 512	Ancient Egyptian well are the gods and goddesses worshipped in ancient Egypt  The beliefs whereas ritualsies these gods formed	Ancient Egyptian deities are the gods and goddesses worshipped in ancient vague. " beliefs and. tried these gods	Ancient Egyptian deities are the gods and goddess hours thoughts in ancient final conception divine beliefs and rituals and these

Figure 4.7: The first 20 tokens reconstructed from a GPT-2 model update for different combinations of sequence length and batch size. Highlighted text represents *exact* matches for both position and token to the original text (a randomly selected user).

discussed as a template for Transformers in federated learning scenarios in Kairouz et al. [75] (11 million parameters), second, the BERT-Base model [32] also attacked in previous work [30, 180] (110 million parameters), and finally, the smallest GPT-2 variation [131] (124 million parameters). We train the small Transformer and GPT-2 as causal language models and BERT as masked language model.

We assemble data from *wikitext* [108], which we partition into separate users by article and tokenize using the GPT-2 (BPE) tokenizer for the small Transformer and GPT-2, and the original BERT (WordPiece) tokenizer for BERT. We then evaluate the attack over a range of sequence lengths and batch sizes. For quantitative evaluations, we always report average scores over the first 100 users (i.e articles), which are long enough to fit batch size $\times$ sequence length tokens. We focus on fedSGD, i.e. single gradient updates

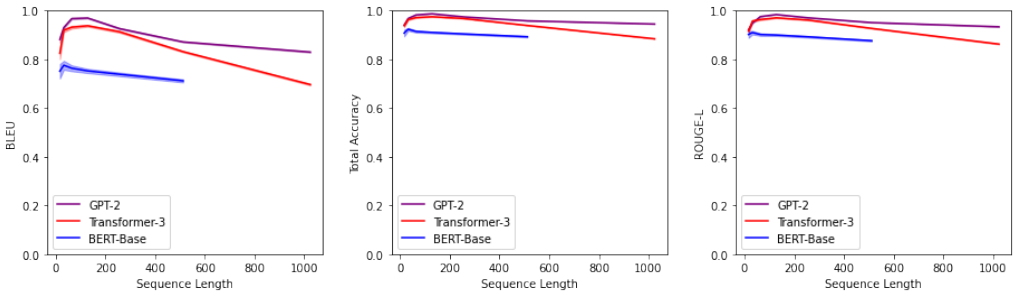


Figure 4.8: Baseline results for our method for different popular architectures and metrics for variable length sequence input (batch size 1). Note BERT's positional encoding caps out at sequence length 512, so we end our experiments for BERT there.

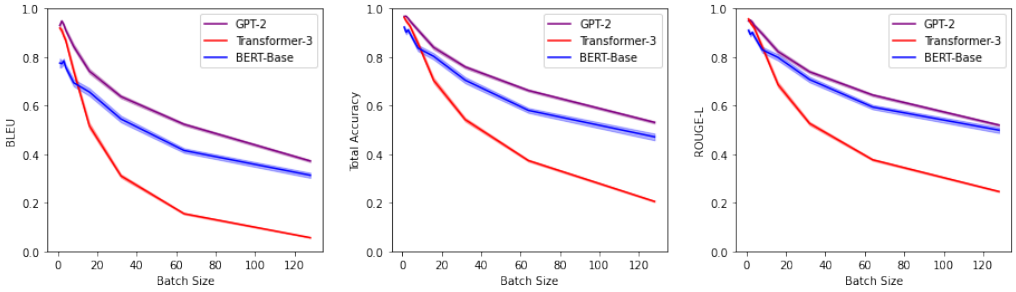


Figure 4.9: Results for our method for different popular architectures and metrics for variable length batch size (sequence length 32).

from all users in this work, but note that related attacks in Fowl et al. [43] are also applicable in fedAVG scenarios - although, arguably, the server controls client learning rate and could thus turn all updates into gradient updates by choosing it maliciously.

We evaluate all attacks using well-known metrics including BLEU score [121], *total* accuracy, and ROUGE-L [94]. Note that our total accuracy score is much stronger than the token accuracy as described in previous works, such as Deng et al. [30], as we are concerned with accuracy of the token *and* the position recovered, and if we were just concerned with token accuracy, we would trivially achieve very high scores via our embedding-layer strategy in Section 4.3.1.

We first present qualitative results for our method. Figure 4.7 shows partial reconstructions for a randomly selected, challenging sequence as batch size and sequence

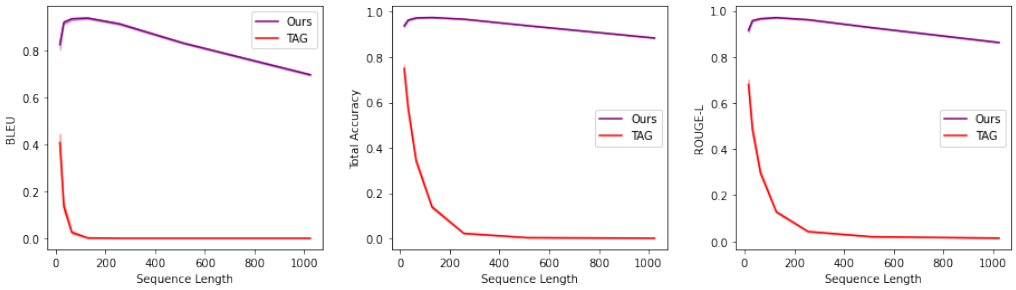


Figure 4.10: Comparison between our method and TAG (honest-but-curious SOTA) for Transformer-3 architecture, variable sequence length (batch size 1).

length increase. We find that even for more difficult scenarios with more tokens, a vast majority of tokens are recoverd, and a majority are recovered in their *exact* correct position.

Single Sequence. As a baseline, we begin experimenting with single sequence recovery - a problem that has proven difficult for previous attacks on text using even moderately sized sequences. This corresponds to the scenario wherein a single user's update on a single sequence is received. In Figure 4.8, we find that even for sequences with > 1000 tokens, for large models like GPT-2, we recover $> 90\%$ of the ground-truth tokens in their *exact* position.

Multi Sequence. In addition to reconstructing single sequences, as discussed in Section 4.3.4, our multi-head attention strategy allows an attacker to reconstruct multiple sequences. This applies to an update from a single user with more than one sequence, and also multiple users aggregating model updates. To the best of our knowledge, we are the first attack to evaluate and explicitly address the problem of multi-user updates. We find that for a sequence length of 32, the proposed attack can recover almost all tokens used in the user updates, and $> 50\%$ of tokens at their *exact* position for > 100 users. Further experimentation for longer sequences can be found in Section 4.11.

Comparison to other threat models. While attacks on FL for text have been few and far between, some previous works have approached the topic of reconstructing user data from FL language model updates. Many of these works focus on the "honest-but-curious" threat model where benign parameters are sent to the user, and the FL protocol is performed normally. We believe that our proposed threat model is just as realistic as the "honest-but-curious" threat model, as anytime a server becomes "curious" about reconstructing user data, they could send a single malicious parameter vector to users to retrieve private data via the proposed attack. Moreover, we find that the severity of the threat posed by our attack is orders of magnitude greater than the current state-of-the-art in the "honest-but-curious" threat model.

We compare by fixing an architecture - in this case the Transformer-3 model described earlier. We then consider the setting with batch size 1 and evaluate our proposed method against the previous art for text attacks - the TAG attack proposed by Deng et al. [30]. We experiment for varying sequence lengths in Figure 4.10. The performance of TAG very quickly degrades for sequences of even moderate length. For a single sequence, our proposed attack maintains high levels of total accuracy for sequences of length exceeding 1000 tokens, whereas the total accuracy for TAG drops below 20% almost immediately, and soon after approaches 0%. Further comparison can be found in Section 4.11.

4.5 Mitigations

Previously, aggregation was thought to preserve privacy in FL on text. However, the proposed attack involving maliciously modified Transformers shows that aggregation alone might not be enough in several use cases. For example, the setting deployed to production in Hard et al. [58] runs FL on users with 400 sentences with 4 words per update on average, well within the range of the investigated attack. We thus briefly discuss other mitigation strategies for attacks like this. We roughly group them into two categories: parameter inspection and differential privacy based defenses.

The former strategy - parameter inspection - might seem obvious on face. For example, in the proposed attack, we duplicate the same measurement in the fully connected part of each Transformer block. Inspecting for such duplication or the rank of certain layers would detect this modification. Furthermore, the discussed attack scales layers to suppress or magnify outputs, so one could look for weights of abnormally large magnitude. However, we believe that these defenses are *not* a reliable or safe way to proceed. This type of strategy inherently puts the user/defender at a disadvantage as they are always reactive to an attack, and not proactive. This means such defenses will always have to "catch up" to ever-adapting attacks. For example, an attacker could easily circumvent some of the above parameter inspections by adding small amounts of noise to the measurement vector, or instead of magnifying/suppressing weights in the multi-head attention block, the attacker could aim for mixed weights (instead of one-hot weights). Overall, the design space for such attacks within the parameters of a Transformer model seems too large to be defended by inspecting parameters for known attacks.

The latter strategy - differential privacy (DP) based defenses - continues to be the more promising route. Local differential privacy, controlled directly by the user and added directly on-device (instead of at the server level [104]) allows protection for users without the need to trust the update sent by the server [8]. However, local DP comes at a noticeable cost in utility, observed in theoretical [75, 78, 161] as well as practical investigations [71]. DP currently also comes at a cost in another dimension: fairness, as model utility is reduced most for underrepresented groups [3]. Overall, any attack on privacy will break down when faced with sufficient differential privacy, but the benefits of federated learning are diminished by it.

4.6 Technical Details

We implement all attacks in an extensible `PyTorch` framework [122] for this type of attack which allows for a full reproduction of the attack and which we attach to this submission. We utilize `huggingface` models and data extensions for the text data [166].The attack is separated into two parts. The first part is the malicious modification of the server parameters, which is triggered immediately before the server sends out their model update payload to users.

We implement the malicious parameter modifications as described in Section 4.3. We initialize from a random model initialization and reserve the first 6 entries of the embedding layer for the sentence encoding for the Transformer-3 model and the first 32 entries for the larger BERT and GPT-2 models. The corresponding entries in the positional and token embeddings are reset to 0. In the MHA modification, we choose a

softmax skewing of $1e8$, although this value can be significantly lower in practice as well. We reserve the last entry of the embedding layer for gradient flow to each linear layer, so that the attack is able to utilize all layers as described. This entry is scaled by $\varepsilon = 1e-6$ for the smaller Transformer-3 model and $\varepsilon = 1e-8$ for the larger models, so that the layer normalization is not skewed by large values arising from the last embedding entry.

For the attack part, we retrieve the gradient update on the server and run all recovery computations in single float precision. We first run a token recovery as discussed in Section 4.3.1, which we additionally summarize in Algorithm 3 and then proceed with the attack following Algorithm 2. For the sentence labeling we choose a dynamic thresholding and assignment algorithm which we describe in Algorithm 5 For all assignment problems we utilize the linear sum assignment solver proposed in Crouse [27] which is a modification of the shortest augmenting path strategy originally proposed in Jonker and Volgenant [74].

During the quantitative evaluation, we trial 100 users each with a request for updates of size sequence length $\times$ batches. We skip all users which do *not* own enough data and do not pad user data with [PAD] tokens, which we think would skew results as it would include a large number of easy tokens to recover. All measurements are hence done on non-trivial sentences, which are concatenated to reach the desired sequence length. Each user's data is completely separated, representing different wikipedia articles as described in the main body. Overall, the attack is highly successful over a range of users even with very different article content.

To compute metrics, we first resort the recovered batches toward the ground truth order by assigning to ground truth batches based on total token accuracy per sentence.

This leads to a potential underestimation of the true accuracy and ROUGE metrics of the recovery as sentence are potentially mismatched. We compute Rouge [94] scores based on the sorted batches and BLEU scores [121] (with the default `huggingface` implementation) by giving all batches as references. We measure total accuracy as the number of tokens that are perfectly identified and in the correct position, assigning no partial credit for either token identity or position. This is notably in contrast to the naming of metrics in Deng et al. [30] where accuracy refers to only overall token accuracy. We refer to that metric as "token accuracy", measuring only the overlap between reference and reconstruction sentence bag-of-words, but report only the total accuracy, given that token accuracy is already near-perfect after the attack on the bag-of-words described in Section 4.3.1.

4.7 Algorithm Details

We detail sub-algorithms as additional material in this section. Algorithm 3 and Algorithm 4 detail the token recovery for transformers with decoder bias and for transformer with a tied embedding. These roughly follow the principles of greedy label recovery strategy proposed in Wainakh et al. [164] and we reproduce them here for completeness, incorporating additional considerations necessary for token retrieval.

We then detail the dynamic thresholding and assignment algorithm for sentence labeling in Algorithm 5. This strategy clusters embeddings into sentences by first greedily matching all embeddings into n groups from a dynamically chosen threshold, and then computing the mean of each group and finally solving an assignment problem to match

each embedding to the group it correlates most to, while obeying the constraints given by sequence length s and number of sentences n. We find this strategy to be more successful than e.g. K-Means clustering as in Bradley et al. [16].

Algorithm 3 Token Recovery (Decoder Bias)

Input: Decoder bias gradient g_b, embedding gradient g_e, sequence length s, number of sequences n.
$v_{\text{tokens}} \leftarrow$ all indices where $g_b < 0$
$v_e \leftarrow$ all indices where $g_e < 0$
v_{tokens} append $v_{\text{tokens}} \setminus v_e$
$m_{\text{impact}} = \frac{1}{sn} \sum_{i \in v_{\text{tokens}}} g_{bi}$
$g_b[v_{\text{tokens}}] \leftarrow g_b[v_{\text{tokens}}] - m_{\text{impact}}$
while Length of $v_{\text{tokens}} < sn$ **do**
    $j = \arg\min_i g_{bi}$
    $g_{bj} \leftarrow g_{bj} - m_{\text{impact}}$
    v_{tokens} append j
end while

Algorithm 4 Token Recovery (Tied encoder Embedding)

Input: Embedding weight gradient g_e, sequence length s, number of expected sequences n, cutoff factor f
$n_{ej} = ||g_{ej}||_2$ for all $j = 1, ...N$ embedding rows
$\mu, \sigma = \text{Mean}(\log n_e), \text{Std}(\log n_e)$
$c = \mu + f\sigma$
$v_{\text{tokens}} \leftarrow$ all indices where $n_e > c$
$m_{\text{impact}} = \frac{1}{sn} \sum_{i \in v_{\text{tokens}}} n_{ei}$
$n_e[v_{\text{tokens}}] \leftarrow n_e[v_{\text{tokens}}] - m_{\text{impact}}$
while Length of $v_{\text{tokens}} < sn$ **do**
    $j = \arg\min_{i \in v_{\text{tokens}}} n_{ei}$
    $n_{ej} \leftarrow n_{ej} - m_{\text{impact}}$
    v_{tokens} append j
end while

4.8 Measurement Transferability

In order to retrieve positionally encoded features from the first linear layer in the feed-forward part of each Transformer block, we create "bins" which partition the possible outcomes of taking the scalar product of an embedding with a random Gaussian vector. To do this, we estimate the mean and variance of such a measurement, and create bins according to partitions of equal mass for a Gaussian with this mean and variance.

Algorithm 5 Sentence Labeling

Input: First v entries of each breached embedding b, sequence length s, number of expected sequences n.
Compute correlation matrix $C = \text{Corr}(b)$
Find minimal threshold $t = \arg\min_t \max(\sum_j C_{ij} > t) > s$
$t_{\text{groups}} = 0$
Set of already assigned indices $A = \emptyset$
Define list of initial labels L
for i where $\sum_j C_{ij} > t$ **do**
  if $i \notin A$ **then**
    Find matches $m = \{j \text{where} C_{i,j} > t\}$
    Find unassigned matches $m \leftarrow m \setminus A$
    Set all entries m in L to i
    A append m
    $t_{\text{groups}} \leftarrow t_{\text{groups}} + 1$
    if t_{groups} equal n **then**
      BREAK
    end if
  end if
end for
Compute n candidates as averages over labelled embeddings $s_j = \frac{1}{|\{i=j, i \in L|\}} \sum_{i=j, i \in L}) b_i$
Compute new correlations $C_s = \text{Corr}(s, b)$
Find final labels L_{final} by solving the assignment problem with costs C_s.

A natural question arises: how well can we estimate this quantity? If the server does not have much data from the user distribution, will this present a problem for recovering user data? To this end, we demonstrate that the server can estimate these measurement quantities well using a surrogate corpus of data. In Figure 4.11, we see that the Gaussian fit using the Wikitext dataset is strikingly similar to the one the server would have found, even on a very different dataset (Shakespeare).

We further note that an ambitious server could also directly use model updates retrieved from users in a first round of federated learning, to directly glean the distribution of these linear layers, given that the lowest bin records the average of all activations of this layer. However, given the ubiquity of public text data, this step appears almost un-

necessary - but possibly relevant in specialized text domains or when using Transformers for other data modalities such as tabular data [147].

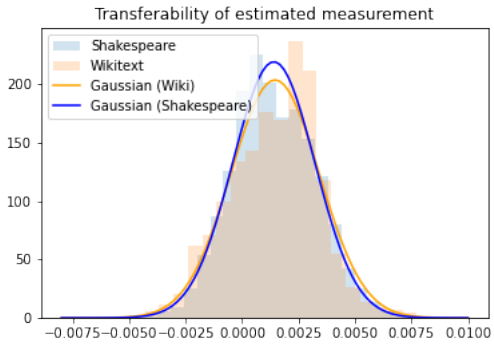


Figure 4.11: Comparison of Gaussian fit to the measurements of Shakespeare dataset, and a Gaussian fit to the measurements (against the same vector) for the Wikitext dataset.

4.9 Variants and Details

Here we describe variants and issues that may arise from different architectures, architectural features, and tasks. All these details could potentially affect the potency of the attack.

4.9.1 Masked Language Modelling

For problems with masked language modelling, the loss is sparse over the sequence length, as only masked tokens compute loss. This impacts the strategy proposed in Section 4.3.2, as with disabled attention mechanisms in all but the first layer, no gradient information flows to unmasked entries in the sequence. However, this can be quickly solved by reactivating the last attention layer, so that it equally attends to all elements in the sequence with a minor weight increment which we set to 10. This way, gradient flow

is re-enabled and all computations can proceed as designed. For BERT, we further disable the default `huggingface` initialiazation of the token embedding, which we reset to a random normal initialization.

Masked language modelling further inhibits the decoder bias strategy discussed in Section 4.3.1, as only masked tokens lead to a non-positive decoder bias gradient. However, we can proceed for masked language models by recovering tokens from the embedding layer as discussed for models without decoder bias. The mask tokens extracted from the bias can later be used to fill masked positions in the input.

4.9.2 GeLU

A minor stumbling for the attacker occurs if the pre-defined model uses GeLU [60] activation instead of ReLU. This is because GeLU does not threshold activations in the same was as ReLU, and transmits gradient signal when the activation < 0. However, a simple workaround for the attacker is to increase the size of the measurement vector and, by doing so, push activations away from 0, and thus more toward standard ReLU behavior. For the main-body experiments, we use ReLU activation for simplicity of presentation, but using the workaround described above, we find in Figure 4.12, Figure 4.13 that the performance of the attack against a GeLU network is comparable to its level against a ReLU network.

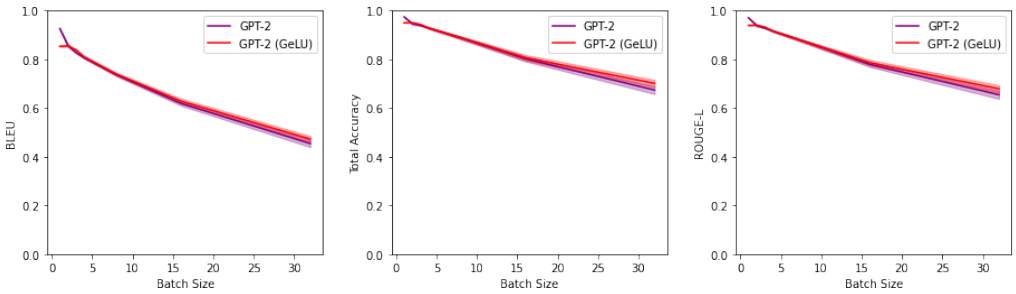


Figure 4.12: Comparison of GeLU and ReLU activation with magnified measurement vector (sequence length 256).

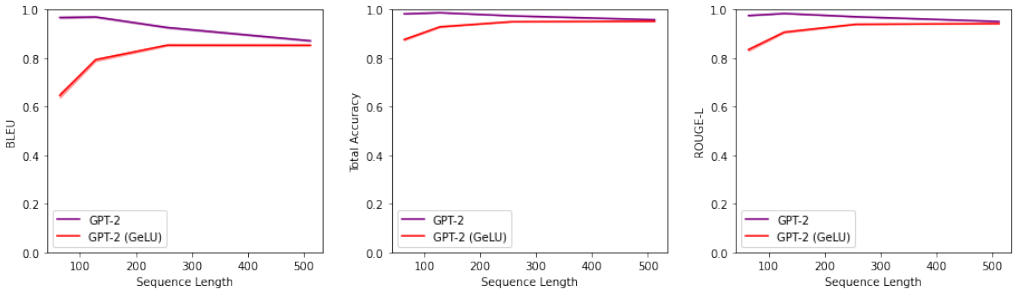


Figure 4.13: Comparison of GeLU and ReLU activation with magnified measurement vector (batch size 1).

4.9.3 Dropout

All experiments in the main body, including comparisons, have assumed that dropout has been turned off for all models under consideration. In standard applications, dropout can be modified from the server-side, even for compiled and otherwise fixed models [117]. In our threat model, dropout hence falls into the category of a server-side parameter that a malicious update could turn off. We also investigated the performance of the attack if the attacker cannot alter standard dropout parameters, finding that dropout decreased total accuracy of the attack by about 10%, e.g. from 93.36% for a sequence of length 512 on GPT-2 to 81.25% with dropout.

4.10 Additional Background Material

The attack TAG in Deng et al. [30] approaches gradient inversion attacks against transformer models from the direct optimization-based angle. This was first suggested in Zhu et al. [180], who provide some preliminary experiments on recovery of short sequences from BERT. Basically, the attack works to recover user data (x^*, y^*) from the measurement of the gradient on this data g by solving the optimization problem of

$$\min_{x,y} ||\mathcal{L}(n(x, \theta), y) - g||^2, \tag{4.1}$$

where x and y are the inputs and labels to be recovered, $\mathcal{L}$ is the loss and $n(\cdot, \theta)$ is the language model with parameters θ. This optimization approach does succeed for short sequences, but the optimization becomes quickly stuck in local minima and cannot recover the original input data. Zhu et al. [180] originally propose the usage of an L-BFGS solver to optimize this problem, but this optimizer can often get stuck in local minima. Deng et al. [30] instead utilize the "BERT" Adam optimizer with hyperparameters as in BERT training [32]. They further improve the objective by adding an additional term that especially penalizes gradients of the first layers, which we also implement when running their attack. A major problem for the attacks of Zhu et al. [180] and Deng et al. [30], however, is the large label space for next-word prediction. In comparison to binary classification tasks as mainly investigated in Deng et al. [31], the large label space leads to significant uncertainty in the optimization of labels y, which leads their attack to reduce in performance as vocabulary size increases.

We further note that Boenisch et al. [11] also propose malicious model modifications (as opposed to our more realistic malicious parameter modifications) to breach privacy in FL for text, however the proposed model in their work is a toy two-layer fully connected model that is not a Transformer-based model. In fact, the strategy employed in their attack cannot be deployed against modern language models that do not construct a linear layer of the length of the sequence, aside from handling of facets of modern language models like positional encodings, or attention mechanisms.

4.11 Further Results

Here we include further results for our attack including large batch size comparisons in Figure 4.14, comparisons to existing attacks for multiple sequences in Figure 4.15, and token recovery with different cutoffs in Figure 4.16.

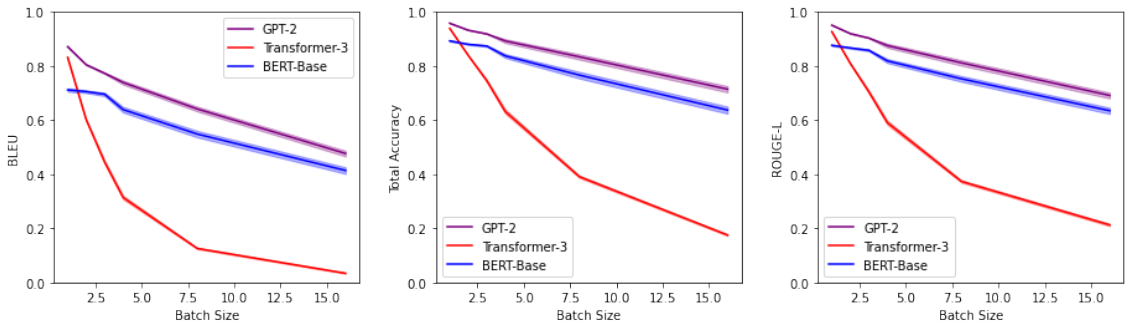


Figure 4.14: Baseline results for our method for different popular architectures and metrics for variable batch size (sequence length 512).

4.12 Conclusions

In this work, we unearth a threatening attack on federated learning for Transformer-based language models. This attack operates in a natural and realistic threat-model centered

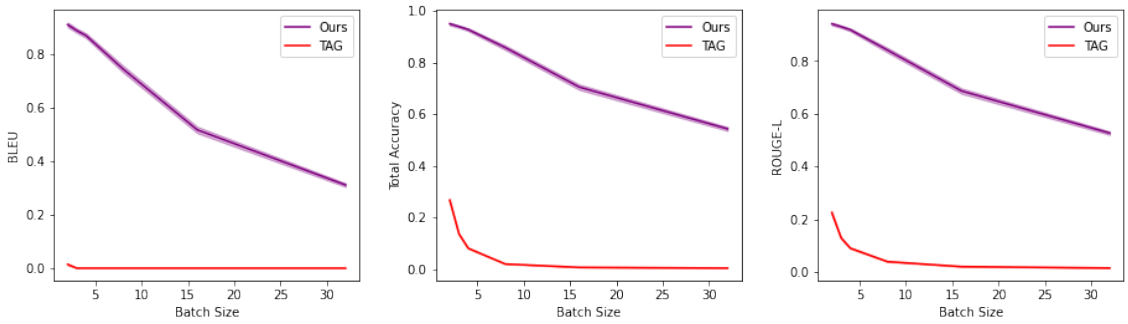


Figure 4.15: Comparison to TAG attack for variable batch size (sequence length 32).

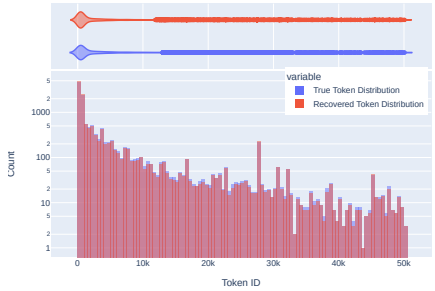


Figure 4.16: Bag-of-words Accuracy for GPT-2 from embedding gradients. Out of 13824 tokens from *wikitext*, the estimated tokens leaked from the embedding layer are 93.1% accurate, unique tokens are 97.1% accurate due to the cutoff at $\frac{3\sigma}{2}$.

on untrusted server updates and recovers a large amount of private user information. This attack operates by reprogramming parameters in a Transformer, and then recovering user data based on a series of simple statistical procedures and assignment problems that can be computed quickly and cheaply, unlike more complicated or learned attacks. We believe that attacks under this threat model are under-explored and caution that such attacks could be made even stronger. The implications for federated learning on text continue to be that federated learning by itself, even with aggregation of users, is not sufficient to guarantee meaningful privacy.

Bibliography

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 308–318, Vienna, Austria, October 2016. Association for Computing Machinery. ISBN 978-1-4503-4139-4. doi: 10.1145/2976749.2978318.

[2] Hojjat Aghakhani, Dongyu Meng, Yu-Xiang Wang, Christopher Kruegel, and Giovanni Vigna. Bullseye Polytope: A Scalable Clean-Label Poisoning Attack with Improved Transferability. *arXiv:2005.00191 [cs, stat]*, April 2020. URL `http://arxiv.org/abs/2005.00191`.

[3] Eugene Bagdasaryan, Omid Poursaeed, and Vitaly Shmatikov. Differential Privacy Has Disparate Impact on Model Accuracy. In *Advances in Neural Information Processing Systems 32*, pages 15479–15488. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/9681-differential-privacy-has-disparate-impact-on-model-accuracy.pdf`.

[4] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[5] Jonathan F. Bard and James E. Falk. An explicit solution to the multi-level programming problem. *Computers & Operations Research*, 9(1):77–100, January 1982. ISSN 0305-0548. doi: 10.1016/0305-0548(82)90007-7.

[6] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Language*, 81(2):121–148, November 2010. ISSN 0885-6125. doi: 10.1007/s10994-010-5188-5.

[7] Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile. *arXiv preprint arXiv:2006.14651*, 2020.

[8] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. Protection Against Reconstruction and Its Applications in Private Federated Learning. *arXiv:1812.00984 [cs, stat]*, June 2019. URL `http://arxiv.org/abs/1812.00984`.

[9] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[10] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning Attacks against Support Vector Machines. *arXiv:1206.6389 [cs, stat]*, June 2012. URL `http://arxiv.org/abs/1206.6389`.

[11] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. When the Curious Abandon Honesty: Federated Learning Is Not Private. *arXiv:2112.02918 [cs]*, December 2021. URL `http://arxiv.org/abs/2112.02918`.

[12] Kallista Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical Secure Aggregation for Privacy Preserving Machine Learning. Technical Report 281, 2017. URL `http://eprint.iacr.org/2017/281`.

[13] Kallista Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards Federated Learning at Scale: System Design. *arXiv:1902.01046 [cs, stat]*, March 2019. URL `http://arxiv.org/abs/1902.01046`.

[14] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[15] Eitan Borgnia, Jonas Geiping, Valeriia Cherepanova, Liam Fowl, Arjun Gupta, Amin Ghiasi, Furong Huang, Micah Goldblum, and Tom Goldstein. Dp-instahide: Provably defusing poisoning and backdoor attacks with differentially private data augmentations. *arXiv preprint arXiv:2103.02079*, 2021.

[16] Paul S. Bradley, Kristin P. Bennett, and Ayhan Demiriz. Constrained k-means clustering. *Microsoft Research, Redmond*, 20(0):0, 2000.

[17] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[18] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[19] Nicholas Carlini and David Wagner. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. In *Proceedings of the 10th ACM Workshop*

on Artificial Intelligence and Security, AISec '17, pages 3–14, Dallas, Texas, USA, November 2017. Association for Computing Machinery. ISBN 978-1-4503-5202-4. doi: 10.1145/3128572.3140444.

[20] Patrick Cason. *Announcing 4 New Libraries for Federated Learning on Web and Mobile Devices*, 2020. https://blog.openmined.org/announcing-new-libraries-for-fl-on-web-and-mobile/.

[21] Adrien Chan-Hon-Tong. An algorithm for generating invisible data poisoning using adversarial noise that breaks image classification deep learning. *Machine Learning and Knowledge Extraction*, 1(1):192–204, 2019.

[22] Guillaume Charpiat, Nicolas Girard, Loris Felardos, and Yuliya Tarabalka. Input Similarity from the Neural Network Perspective. In *Advances in Neural Information Processing Systems 32*, pages 5342–5351. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/8775-input-similarity-from-the-neural-network-perspective.pdf.

[23] Xiaoyi Chen, Ahmed Salem, Michael Backes, Shiqing Ma, and Yang Zhang. Badnl: Backdoor attacks against nlp models. *arXiv preprint arXiv:2006.01043*, 2020.

[24] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved Baselines with Momentum Contrastive Learning. *arXiv:2003.04297 [cs]*, March 2020. URL http://arxiv.org/abs/2003.04297.

[25] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[26] Valeriia Cherepanova, Micah Goldblum, Harrison Foley, Shiyuan Duan, John Dickerson, Gavin Taylor, and Tom Goldstein. Lowkey: leveraging adversarial attacks to protect social media users from facial recognition. *arXiv preprint arXiv:2101.07922*, 2021.

[27] David F. Crouse. On implementing 2D rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, August 2016. ISSN 1557-9603. doi: 10.1109/TAES.2016.140952.

[28] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. AutoAugment: Learning Augmentation Policies from Data. *arXiv:1805.09501 [cs, stat]*, April 2019. URL http://arxiv.org/abs/1805.09501.

[29] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attacks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 321–338, 2019. ISBN 978-1-939133-06-9. URL https://www.usenix.org/conference/usenixsecurity19/presentation/demontis.

[30] Jieren Deng, Yijue Wang, Ji Li, Chao Shang, Hang Liu, Sanguthevar Rajasekaran, and Caiwen Ding. TAG: Gradient Attack on Transformer-based Language Models. *arXiv:2103.06819 [cs]*, September 2021. URL http://arxiv.org/abs/2103.06819.

[31] Yuntian Deng, Anton Bakhtin, Myle Ott, Arthur Szlam, and Marc'Aurelio Ranzato. Residual Energy-Based Models for Text Generation. *arXiv:2004.11714 [cs]*, April 2020. URL http://arxiv.org/abs/2004.11714.

[32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. URL http://arxiv.org/abs/1810.04805.

[33] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs]*, June 2021. URL http://arxiv.org/abs/2010.11929.

[34] Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3-4):211–407, 2013. ISSN 1551-305X, 1551-3068. doi: 10.1561/0400000042.

[35] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations. *arXiv:1712.02779 [cs, stat]*, December 2017. URL http://arxiv.org/abs/1712.02779.

[36] Lixin Fan, Kam Woh Ng, Ce Ju, Tianyu Zhang, Chang Liu, Chee Seng Chan, and Qiang Yang. Rethinking Privacy Preserving Deep Learning: How to Evaluate and Thwart Privacy Attacks. In *Federated Learning: Privacy and Incentive*, Lecture Notes in Computer Science, pages 32–50. Springer International Publishing, Cham, 2020. ISBN 978-3-030-63076-8. doi: 10.1007/978-3-030-63076-8_3.

[37] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. Poisoning Attacks to Graph-Based Recommender Systems. In *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, pages 381–392, San Juan, PR, USA, December 2018. Association for Computing Machinery. ISBN 978-1-4503-6569-7. doi: 10.1145/3274694.3274706.

[38] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. Influence function based data poisoning attacks to top-n recommender systems. In *Proceedings of The Web Conference 2020*, pages 3019–3025, 2020.

[39] Ji Feng, Qi-Zhi Cai, and Zhi-Hua Zhou. Learning to Confuse: Generating Training Time Adversarial Data with Auto-Encoder. *arXiv:1905.09027 [cs, stat]*, May 2019. URL http://arxiv.org/abs/1905.09027.

[40] Ji Feng, Qi-Zhi Cai, and Zhi-Hua Zhou. Learning to Confuse: Generating Training Time Adversarial Data with Auto-Encoder. In *Advances in Neural Information Processing Systems 32*, pages 11994–12004. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9368-learning-to-confuse-generating-training-time-adversarial-data-with-auto-encoder.pdf.

[41] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv:1703.03400 [cs]*, March 2017. URL http://arxiv.org/abs/1703.03400.

[42] Liam Fowl, Ping-yeh Chiang, Micah Goldblum, Jonas Geiping, Arpit Bansal, Wojtek Czaja, and Tom Goldstein. Preventing unauthorized use of proprietary data: Poisoning for secure dataset release. *arXiv preprint arXiv:2103.02683*, 2021.

[43] Liam Fowl, Jonas Geiping, Wojtek Czaja, Micah Goldblum, and Tom Goldstein. Robbing the Fed: Directly Obtaining Private Data in Federated Learning with Modified Models. *arXiv:2110.13057 [cs]*, October 2021. URL http://arxiv.org/abs/2110.13057.

[44] Suzanne Frey. Introducing Android's Private Compute Services, September 2021. URL https://security.googleblog.com/2021/09/introducing-androids-private-compute.html.

[45] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting Gradients – How easy is it to break privacy in federated learning? *arXiv:2003.14053 [cs]*, March 2020. URL https://arxiv.org/abs/2003.14053v1.

[46] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting Gradients - How easy is it to break privacy in federated learning? In *Advances in Neural Information Processing Systems*, volume 33, December 2020. URL https://proceedings.neurips.cc//paper_files/paper/2020/hash/c4ede56bbd98819ae6112b20ac6bf145-Abstract.html.

[47] Jonas Geiping, Liam Fowl, W Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. Witches' brew: Industrial scale data poisoning via gradient matching. *arXiv preprint arXiv:2009.02276*, 2020.

[48] Jonas Geiping, Liam Fowl, Gowthami Somepalli, Micah Goldblum, Michael Moeller, and Tom Goldstein. What doesn't kill you makes you robust (er): Adversarial training against poisons and backdoors. *arXiv preprint arXiv:2102.13624*, 2021.

[49] Gaurav Ghati. Head pruning in transformer models!, May 2020. URL https://towardsdatascience.com/head-pruning-in-transformer-models-ec222ca9ece7.

[50] Micah Goldblum, Liam Fowl, Soheil Feizi, and Tom Goldstein. Adversarially robust distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3996–4003, 2020.

[51] Micah Goldblum, Jonas Geiping, Avi Schwarzschild, Michael Moeller, and Tom Goldstein. Truth or backpropaganda? An empirical investigation of deep learning theory. In *Eighth International Conference on Learning Representations (ICLR 2020, Oral Presentation)*, April 2020. URL https://iclr.cc/virtual_2020/poster_HyxyIgHFvr.html.

[52] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *arXiv preprint arXiv:2012.10544*, 2020.

[53] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, Cambridge, 1st edition edition, September 2009. ISBN 978-0-521-11991-7.

[54] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[55] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[56] Team Google Research. Federated Learning, May 2019. URL https://federated.withgoogle.com.

[57] Sven Gowal, Jonathan Uesato, Chongli Qin, Po-Sen Huang, Timothy Mann, and Pushmeet Kohli. An alternative surrogate loss for pgd-based adversarial testing. *arXiv preprint arXiv:1910.09338*, 2019.

[58] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated Learning for Mobile Keyboard Prediction. *arXiv:1811.03604 [cs]*, February 2019. URL http://arxiv.org/abs/1811.03604.

[59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL http://arxiv.org/abs/1512.03385.

[60] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[61] Wen Heng, Shuchang Zhou, and Tingting Jiang. Harmonic Adversarial Attack Method. *arXiv:1807.10590 [cs]*, July 2018. URL http://arxiv.org/abs/1807.10590.

[62] Sanghyun Hong, Varun Chandrasekaran, Yiğitcan Kaya, Tudor Dumitraş, and Nicolas Papernot. On the Effectiveness of Mitigating Data Poisoning Attacks with Gradient Shaping. *arXiv:2002.11497 [cs]*, February 2020. URL http://arxiv.org/abs/2002.11497.

[63] Rui Hu, Yuanxiong Guo, Miao Pan, and Yanmin Gong. Targeted Poisoning Attacks on Social Recommender Systems. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, December 2019. doi: 10.1109/GLOBECOM38437.2019.9013539.

[64] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. August 2016. URL https://arxiv.org/abs/1608.06993.

[65] Hanxun Huang, Xingjun Ma, Sarah Monazam Erfani, James Bailey, and Yisen Wang. Unlearnable examples: Making personal data unexploitable. *arXiv preprint arXiv:2101.04898*, 2021.

[66] Jinggang Huang and David Mumford. Statistics of natural images and models. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, volume 1, pages 541–547 Vol. 1, June 1999. doi: 10.1109/CVPR.1999.786990.

[67] W Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. Metapoison: Practical general-purpose clean-label data poisoning. *arXiv preprint arXiv:2004.00225*, 2020.

[68] W. Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. MetaPoison: Practical General-purpose Clean-label Data Poisoning. In *Advances in Neural Information Processing Systems*, volume 33, Vancouver, Canada, December 2020. URL https://proceedings.neurips.cc//paper_files/paper/2020/hash/8ce6fc704072e351679ac97d4a985574-Abstract.html.

[69] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *arXiv preprint arXiv:1905.02175*, 2019.

[70] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial Transformer Networks. In *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5854-spatial-transformer-networks.pdf.

[71] Bargav Jayaraman and David Evans. Evaluating Differentially Private Machine Learning in Practice. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1895–1912, 2019. ISBN 978-1-939133-06-9. URL

https://www.usenix.org/conference/usenixsecurity19/presentation/jayaraman.

[72] Jinwoo Jeon, Jaechang Kim, Kangwook Lee, Sewoong Oh, and Jungseul Ok. Gradient Inversion with Generative Image Prior. In *International Workshop on Federated Learning for User Privacy and Data Confidentiality*, page 12, 2021.

[73] Arthur Jochems, Timo M. Deist, Johan van Soest, Michael Eble, Paul Bulens, Philippe Coucke, Wim Dries, Philippe Lambin, and Andre Dekker. Distributed learning: Developing a predictive model based on data from multiple hospitals without data leaving the hospital – A real life proof of concept. *Radiotherapy and Oncology*, 121(3):459–467, December 2016. ISSN 0167-8140, 1879-0887. doi: 10.1016/j.radonc.2016.10.002.

[74] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, December 1987. ISSN 1436-5057. doi: 10.1007/BF02278710.

[75] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and Open Problems in Federated Learning. *arXiv:1912.04977 [cs, stat]*, March 2021. URL http://arxiv.org/abs/1912.04977.

[76] Danny Karmon, Daniel Zoran, and Yoav Goldberg. LaVAN: Localized and Visible Adversarial Noise. *arXiv:1801.02608 [cs]*, January 2018. URL http://arxiv.org/abs/1801.02608.

[77] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks, 2015. *URL http://karpathy. github. io/2015/05/21/rnn-effectiveness*, 2016.

[78] Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What Can We Learn Privately? *SIAM Journal on Computing*, 40(3):793–826, January 2011. ISSN 0097-5397. doi: 10.1137/090756090.

[79] Marc Khoury and Dylan Hadfield-Menell. On the geometry of adversarial examples. *arXiv preprint arXiv:1811.00525*, 2018.

[80] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, San Diego, May 2015. URL http://arxiv.org/abs/1412.6980.

[81] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894. PMLR, 2017.

[82] Pang Wei Koh and Percy Liang. Understanding Black-box Predictions via Influence Functions. In *International Conference on Machine Learning*, pages 1885–1894, July 2017. URL http://proceedings.mlr.press/v70/koh17a.html.

[83] Pang Wei Koh, Jacob Steinhardt, and Percy Liang. Stronger data poisoning attacks break data sanitization defenses. *arXiv preprint arXiv:1811.00741*, 2018.

[84] Pang Wei Koh, Jacob Steinhardt, and Percy Liang. Stronger Data Poisoning Attacks Break Data Sanitization Defenses. *arXiv:1811.00741 [cs, stat]*, November 2018. URL http://arxiv.org/abs/1811.00741.

[85] Jakub Konečný, Brendan McMahan, and Daniel Ramage. Federated Optimization:Distributed Optimization Beyond the Datacenter. *arXiv:1511.03575 [cs, math]*, November 2015. URL http://arxiv.org/abs/1511.03575.

[86] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[87] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[88] R. S. Siva Kumar, M. Nyström, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia. Adversarial Machine Learning-Industry Perspectives. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 69–75, May 2020. doi: 10.1109/SPW50608.2020.00028.

[89] E.Y. Lam and J.W. Goodman. A mathematical analysis of the DCT coefficient distributions for images. *IEEE Transactions on Image Processing*, 9(10):1661–1666, October 2000. ISSN 10577149. doi: 10.1109/83.869177.

[90] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient BackProp. In *Neural Networks: Tricks of the Trade*, Lecture Notes in Computer Science, pages 9–50. Springer, Berlin, Heidelberg, 1998. ISBN 978-3-540-49430-0. doi: 10.1007/3-540-49430-8_2.

[91] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, May 2015. ISSN 1476-4687. doi: 10.1038/nature14539.

[92] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data Poisoning Attacks on Factorization-Based Collaborative Filtering. In *Advances in Neural Information Processing Systems 29*, pages 1885–1893. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6142-data-poisoning-attacks-on-factorization-based-collaborative-filtering.pdf.

[93] Weilin Li. *Topics in Harmonic Analysis, Sparse Representations, and Data Analysis*. PhD thesis, University of Maryland, College Park, 2018.

[94] Chin-Yew Lin. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013.

[95] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into Transferable Adversarial Examples and Black-box Attacks. *arXiv:1611.02770 [cs]*, February 2017. URL http://arxiv.org/abs/1611.02770.

[96] Giulio Lovisotto, Simon Eberz, and Ivan Martinovic. Biometric Backdoors: A Poisoning Attack Against Unsupervised Template Updating. *arXiv:1905.09162 [cs]*, May 2019. URL http://arxiv.org/abs/1905.09162.

[97] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep Neural Network Fingerprinting by Conferrable Adversarial Examples. *arXiv:1912.00888 [cs, stat]*, February 2020. URL http://arxiv.org/abs/1912.00888.

[98] Yuzhe Ma, Xiaojin Zhu, and Justin Hsu. Data Poisoning against Differentially-Private Learners: Attacks and Defenses. *arXiv:1903.09860 [cs]*, July 2019. URL http://arxiv.org/abs/1903.09860.

[99] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[100] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv:1706.06083 [cs, stat]*, June 2017. URL http://arxiv.org/abs/1706.06083.

[101] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[102] Brendan McMahan and Daniel Ramage. Federated Learning: Collaborative Machine Learning without Centralized Training Data, April 2017. URL http://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[103] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. *arXiv:1602.05629 [cs]*, February 2017. URL http://arxiv.org/abs/1602.05629.

[104] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning Differentially Private Recurrent Language Models. *arXiv:1710.06963 [cs]*, February 2018. URL http://arxiv.org/abs/1710.06963.

[105] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[106] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. *arXiv:1805.04049 [cs]*, November 2018. URL http://arxiv.org/abs/1805.04049.

[107] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting Unintended Feature Leakage in Collaborative Learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706, May 2019. doi: 10.1109/SP.2019.00029.

[108] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. November 2016. URL https://openreview.net/forum?id=Byj72udxe.

[109] Marius Mosbach, Maksym Andriushchenko, Thomas Trost, Matthias Hein, and Dietrich Klakow. Logit Pairing Methods Can Fool Gradient-Based Attacks. *arXiv:1810.12042 [cs, stat]*, March 2019. URL http://arxiv.org/abs/1810.12042.

[110] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards Poisoning of Deep Learning Algorithms with Back-gradient Optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, AISec '17, pages 27–38, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5202-4. doi: 10.1145/3128572.3140451.

[111] Luis Muñoz-González, Bjarne Pfitzner, Matteo Russo, Javier Carnerero-Cano, and Emil C. Lupu. Poisoning Attacks with Generative Adversarial Nets. *arXiv:1906.07773 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1906.07773.

[112] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 27–38, 2017.

[113] Preetum Nakkiran. A discussion of 'adversarial examples are not bugs, they are features': Adversarial examples are just bugs, too. *Distill*, 2019. doi: 10.23915/distill.00019.5. https://distill.pub/2019/advex-bugs-discussion/response-5.

[114] Gaurav Kumar Nayak, Konda Reddy Mopuri, Vaisakh Shaj, Venkatesh Babu Radhakrishnan, and Anirban Chakraborty. Zero-shot knowledge distillation in deep networks. In *International Conference on Machine Learning*, pages 4743–4751. PMLR, 2019.

[115] Franck Olivier Ndjakou Njeunje. *Computational methods in machine learning: transport model, Haar wavelet, DNA classification, and MRI*. PhD thesis, University of Maryland, College Park, 2018.

[116] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, New York, 2nd ed edition, 2006. ISBN 978-0-387-30303-1.

[117] Documentation ONNX. ONNX Operator Schemas. Open Neural Network Exchange, January 2022. URL `https://github.com/onnx/onnx/blob/50a1981dc3d50af13075cec33b08b4c87fb0e41f/docs/Operators.md`.

[118] Xudong Pan, Mi Zhang, Yifan Yan, Jiaming Zhu, and Min Yang. Theory-Oriented Deep Leakage from Gradients via Linear Equation Solver. *arXiv:2010.13356 [cs, stat]*, October 2020. URL `http://arxiv.org/abs/2010.13356`.

[119] Nicolas Papernot. A Marauder's Map of Security and Privacy in Machine Learning. *arXiv:1811.01134 [cs]*, November 2018. URL `http://arxiv.org/abs/1811.01134`.

[120] Nicolas Papernot and Patrick McDaniel. Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning. *arXiv:1803.04765 [cs, stat]*, March 2018. URL `http://arxiv.org/abs/1803.04765`.

[121] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135.

[122] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS 2017 Autodiff Workshop*, Long Beach, CA, 2017. URL `https://openreview.net/forum?id=BJJsrmfCZ`.

[123] Andrea Paudice, Luis Muñoz-González, Andras Gyorgy, and Emil C. Lupu. Detection of Adversarial Training Examples in Poisoning Attacks through Anomaly Detection. *arXiv:1802.03041 [cs, stat]*, February 2018. URL http://arxiv.org/abs/1802.03041.

[124] Andrea Paudice, Luis Muñoz-González, and Emil C. Lupu. Label Sanitization Against Label Flipping Poisoning Attacks. In *ECML PKDD 2018 Workshops*, Lecture Notes in Computer Science, pages 5–15, Cham, 2019. Springer International Publishing. ISBN 978-3-030-13453-2. doi: 10.1007/978-3-030-13453-2_1.

[125] Neehar Peri, Neal Gupta, W. Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P. Dickerson. Deep k-nn defense against clean-label data poisoning attacks, 2019.

[126] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. Technical Report 715, 2017. URL http://eprint.iacr.org/2017/715.

[127] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-Preserving Deep Learning: Revisited and Enhanced. In *Applications and Techniques in Information Security*, Communications in Computer and Information Science, pages 100–110, Singapore, 2017. Springer. ISBN 978-981-10-5421-1. doi: 10.1007/978-981-10-5421-1_9.

[128] Jia Qian and Lars Kai Hansen. What can we learn from gradients? September 2020. URL https://openreview.net/forum?id=gQn5xeVtz0I.

[129] Jia Qian, Hiba Nassar, and Lars Kai Hansen. Minimal conditions analysis of gradient-based reconstruction in Federated Learning. *arXiv:2010.15718 [cs, eess]*, March 2021. URL http://arxiv.org/abs/2010.15718.

[130] Chongli Qin, James Martens, Sven Gowal, Dilip Krishnan, Krishnamurthy Dvijotham, Alhussein Fawzi, Soham De, Robert Stanforth, and Pushmeet Kohli. Adversarial Robustness through Local Linearization. *arXiv:1907.02610 [cs, stat]*, October 2019. URL http://arxiv.org/abs/1907.02610.

[131] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. page 24, 2019.

[132] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated Learning for Emoji Prediction in a Mobile Keyboard. *arXiv:1906.04329 [cs]*, June 2019. URL http://arxiv.org/abs/1906.04329.

[133] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[134] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Lecture Notes in Computer Science, pages 234–241. Springer International Publishing, 2015. ISBN 978-3-319-24574-4.

[135] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[136] Daniel L. Ruderman. The statistics of natural images. 5(4):517–548, January 1994. ISSN 0954-898X. doi: 10.1088/0954-898X/5/4/006.

[137] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, December 2015. ISSN 1573-1405. doi: 10.1007/s11263-015-0816-y.

[138] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden Trigger Backdoor Attacks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):11957–11965, April 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i07.6871.

[139] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *arXiv:1801.04381 [cs]*, January 2018. URL http://arxiv.org/abs/1801.04381.

[140] Avi Schwarzschild, Micah Goldblum, Arjun Gupta, John P. Dickerson, and Tom Goldstein. Just How Toxic is Data Poisoning? A Unified Benchmark for Backdoor and Data Poisoning Attacks. *arXiv:2006.12557 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2006.12557.

[141] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. *arXiv:1804.00792 [cs, stat]*, April 2018. URL http://arxiv.org/abs/1804.00792.

[142] Shawn Shan, Emily Wenger, Jiayun Zhang, Huiying Li, Haitao Zheng, and Ben Y. Zhao. Fawkes: Protecting Personal Privacy against Unauthorized Deep Learning Models. *arXiv:2002.08327 [cs, stat]*, February 2020. URL http://arxiv.org/abs/2002.08327.

[143] J. Shen, X. Zhu, and D. Ma. TensorClog: An Imperceptible Poisoning Attack on Deep Neural Network Applications. *IEEE Access*, 7:41498–41506, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2905915.

[144] Reza Shokri and Vitaly Shmatikov. Privacy-Preserving Deep Learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*, pages 1310–1321, Denver, Colorado, USA, 2015. ACM Press. ISBN 978-1-4503-3832-5. doi: 10.1145/2810103.2813687.

[145] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, September 2014. URL http://arxiv.org/abs/1409.1556.

[146] David Solans, Battista Biggio, and Carlos Castillo. Poisoning Attacks on Algorithmic Fairness. *arXiv:2004.07401 [cs.LG]*, April 2020. URL https://arxiv.org/abs/2004.07401v1.

[147] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

[148] Hossein Souri, Micah Goldblum, Liam Fowl, Rama Chellappa, and Tom Goldstein. Sleeper agent: Scalable hidden trigger backdoors for neural networks trained from scratch. *arXiv preprint arXiv:2106.08970*, 2021.

[149] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified Defenses for Data Poisoning Attacks. In *Advances in Neural Information Processing Systems 30*, pages 3517–3529. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6943-certified-defenses-for-data-poisoning-attacks.pdf.

[150] David Stutz, Matthias Hein, and Bernt Schiele. Disentangling adversarial robustness and generalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6976–6987, 2019.

[151] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume Iii, and Tudor Dumitras. When Does Machine Learning {FAIL}? Generalized Transferability for Evasion and Poisoning Attacks. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1299–1316, 2018. ISBN 978-1-939133-04-5. URL https://www.usenix.org/conference/usenixsecurity18/presentation/suciu.

[152] Jingwei Sun, Ang Li, Binghui Wang, Huanrui Yang, Hai Li, and Yiran Chen. Soteria: Provable Defense Against Privacy Leakage in Federated Learning From Representation Perspective. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9311–9319, 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Sun_Soteria_Provable_Defense_Against_Privacy_Leakage_in_Federated_Lea

[153] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), pages 1–9, June 2015. doi: 10.1109/CVPR.2015.7298594.

[154] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[155] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *arXiv:1312.6199 [Cs]*, December 2013. URL http://arxiv.org/abs/1312.6199.

[156] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014. doi: 10.1109/CVPR.2014.220.

[157] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, Columbus, OH, USA, June 2014. IEEE. ISBN 978-1-4799-5118-5. doi: 10.1109/CVPR.2014.220.

[158] Lue Tao, Lei Feng, Jinfeng Yi, Sheng-Jun Huang, and Songcan Chen. Provable defense against delusive poisoning. *arXiv preprint arXiv:2102.04716*, 2021.

[159] Nguyen Truong, Kai Sun, Siyao Wang, Florian Guitton, and YiKe Guo. Privacy preservation in federated learning: An insightful survey from the GDPR perspective. *Computers & Security*, 110:102402, November 2021. ISSN 0167-4048. doi: 10.1016/j.cose.2021.102402.

[160] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Clean-Label Backdoor Attacks. *openreview*, September 2018. URL https://openreview.net/forum?id=HJg6e2CcK7.

[161] Jonathan Ullman. Tight Lower Bounds for Locally Differentially Private Selection. *arXiv:1802.02638 [cs]*, May 2021. URL http://arxiv.org/abs/1802.02638.

[162] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. URL http://arxiv.org/abs/1706.03762.

[163] Michael Veale, Reuben Binns, and Lilian Edwards. Algorithms that remember: Model inversion attacks and data protection law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2133): 20180083, November 2018. doi: 10.1098/rsta.2018.0083.

[164] Aidmar Wainakh, Fabrizio Ventola, Till Müßig, Jens Keim, Carlos Garcia Cordero, Ephraim Zimmer, Tim Grube, Kristian Kersting, and Max Mühlhäuser. User Label Leakage from Gradients in Federated Learning. *arXiv:2105.09369 [cs]*, June 2021. URL http://arxiv.org/abs/2105.09369.

[165] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. Beyond Inferring Class Representatives: User-Level Privacy Leakage From Federated Learning. *arXiv:1812.00535 [cs]*, December 2018. URL http://arxiv.org/abs/1812.00535.

[166] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *arXiv:1910.03771 [cs]*, July 2020. URL http://arxiv.org/abs/1910.03771.

[167] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pages 1689–1698. PMLR, 2015.

[168] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014.

[169] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M. Alvarez, Jan Kautz, and Pavlo 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.

[170] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6023–6032, 2019.

[171] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. *arXiv:1605.07146 [cs]*, May 2016. URL http://arxiv.org/abs/1605.07146.

[172] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations (ICLR) 2017*, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

[173] Hanwei Zhang, Yannis Avrithis, Teddy Furon, and Laurent Amsaleg. Smooth Adversarial Examples. *arXiv:1903.11862 [cs]*, March 2019. URL http://arxiv.org/abs/1903.11862.

[174] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. Mixup: Beyond Empirical Risk Minimization. *arXiv:1710.09412 [cs, stat]*, October 2017. URL http://arxiv.org/abs/1710.09412.

[175] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Zhang_The_Unreasonable_Effectiveness_CVPR_2018_paper.html.

[176] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. iDLG: Improved Deep Leakage from Gradients. *arXiv:2001.02610 [cs, stat]*, January 2020. URL http://arxiv.org/abs/2001.02610.

[177] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. *arXiv preprint arXiv:1710.11342*, 2017.

[178] Chen Zhu, W. Ronny Huang, Ali Shafahi, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable Clean-Label Poisoning Attacks on Deep Neural Nets. *arXiv:1905.05897 [cs, stat]*, May 2019. URL http://arxiv.org/abs/1905.05897.

[179] Junyi Zhu and Matthew Blaschko. R-GAP: Recursive Gradient Attack on Privacy. *arXiv:2010.07733 [cs]*, March 2021. URL http://arxiv.org/abs/2010.07733.

[180] Ligeng Zhu, Zhijian Liu, and Song Han. Deep Leakage from Gradients. In *Advances in Neural Information Processing Systems 32*, pages 14774–14784. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9617-deep-leakage-from-gradients.pdf.