Machine Learning Models that Remember Too Much

Congzheng Song
Cornell University
cs2296@cornell.edu

Thomas Ristenpart
Cornell Tech
ristenpart@cornell.edu

Vitaly Shmatikov
Cornell Tech
shmat@cs.cornell.edu

ABSTRACT

Machine learning (ML) is becoming a commodity. Numerous ML frameworks and services are available to data holders who are not ML experts but want to train predictive models on their data. It is important that ML models trained on sensitive inputs (e.g., personal images or documents) not leak too much information about the training data.

We consider a malicious ML provider who supplies model-training code to the data holder, does *not* observe the training, but then obtains white- or black-box access to the resulting model. In this setting, we design and implement practical algorithms, some of them very similar to standard ML techniques such as regularization and data augmentation, that "memorize" information about the training dataset in the model—yet the model is as accurate and predictive as a conventionally trained model. We then explain how the adversary can extract memorized information from the model.

We evaluate our techniques on standard ML tasks for image classification (CIFAR10), face recognition (LFW and FaceScrub), and text analysis (20 Newsgroups and IMDB). In all cases, we show how our algorithms create models that have high predictive power yet allow accurate extraction of subsets of their training data.

CCS CONCEPTS

• **Security and privacy → Software and application security**;

KEYWORDS

privacy, machine learning

1 INTRODUCTION

Machine learning (ML) has been successfully applied to many data analysis tasks, from recognizing images to predicting retail purchases. Numerous ML libraries and online services are available (see Section 2.2) and new ones appear every year.

Data holders who seek to apply ML techniques to their datasets, many of which include sensitive data, may not be ML experts. They use third-party ML code "as is," without understanding what this code is doing. As long as the resulting models have high predictive power for the specified tasks, the data holder may not even ask "what *else* did the model capture about my training data?"

Modern ML models, especially artificial neural networks, have huge capacity for "memorizing" arbitrary information [75]. This can lead to overprovisioning: even an accurate model may be using only a fraction of its raw capacity. The provider of an ML library or operator of an ML service can modify the training algorithm so that the model encodes more information about the training dataset than is strictly necessary for high accuracy on its primary task.

Our contributions. We show that relatively minor modifications to training algorithms can produce models that have high quality by the standard ML metrics (such as accuracy and generalizability), yet leak detailed information about their training datasets.

We assume that a malicious ML provider supplies the training algorithm to the data holder but does not observe its execution. After the model has been created, the provider either obtains the entire model (white box) or gains input-output access to it (black box). The provider then aims to extract information about the training dataset from the model. This scenario can arise when the data holder uses a malicious ML library and also in algorithm marketplaces [2, 27, 54] that let data holders pay to use third-party training algorithms in an environment secured by the marketplace operator.

In the white-box case, we evaluate several techniques: (1) encoding sensitive information about the training dataset directly in the least significant bits of the model parameters, (2) forcing the parameters to be highly correlated with the sensitive information, and (3) encoding the sensitive information in the signs of the parameters. The latter two techniques involve adding a malicious "regularization" term to the loss function and, from the viewpoint of the data holder, could appear as yet another regularization technique.

In the black-box case, we use a technique that resembles data augmentation (extending the training dataset with additional synthetic data) without any modifications to the training algorithm. The resulting model is thus, in effect, trained on two tasks. The first, primary task is the main classification task specified by the data holder. The secondary, malicious task is as follows: given a particular synthetic input, "predict" one or more secret bits about the actual training dataset.

Because the labels associated with our synthetic augmented inputs encode secrets about the training data, they do not correspond to any structure in these inputs. Therefore, our secondary task asks the model to "learn" what is essentially random labeling. Nevertheless, we empirically demonstrate that models become overfitted to the synthetic inputs—without any significant impact on their accuracy and generalizability on the primary tasks. This enables black-box information extraction: the adversary provides a synthetic input, and the model outputs the label, i.e., the secret bits about the actual training dataset that it memorized during training.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '17, October 30-November 3, 2017, Dallas, TX, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery.
ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134077

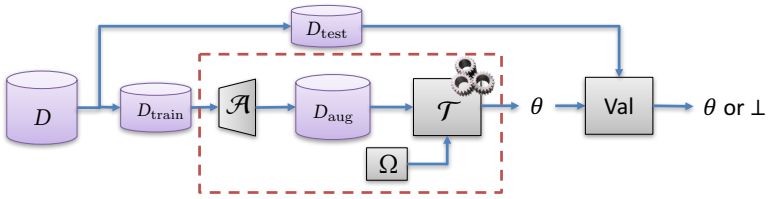

Figure 1: A typical ML training pipeline. Data D is split into training set D_{train} and test set D_{test}. Training data may be augmented using an algorithm $\mathcal{A}$, and then parameters are computed using a training algorithm $\mathcal{T}$ that uses a regularizer Ω. The resulting parameters are validated using the test set and either accepted or rejected (an error $\perp$ is output). If the parameters θ are accepted, they may be published (white-box model) or deployed in a prediction service to which the adversary has input/output access (black-box model). The dashed box indicates the portions of the pipeline that may be controlled by the adversary.

We evaluate white- and black-box malicious training techniques on several benchmark ML datasets and tasks: CIFAR10 (image classification), Labeled Faces in the Wild (face recognition), FaceScrub (gender classification and face recognition), 20 Newsgroups (text classification), and IMDB (binary sentiment classification). In all cases, accuracy and generalizability of the maliciously trained models are virtually identical to the conventional models.

We demonstrate how the adversary can extract subsets of the training data from maliciously trained models and measure how the choices of different parameters influence the amount and accuracy of extraction. For example, with a white-box attack that encodes training data directly in the model parameters, we create a text classifier that leaks 70% of its 10,000-document training corpus without any negative impact on the model's accuracy. With a black-box attack, we create a binary gender classifier that allows accurate reconstruction of 17 complete face images from its training dataset, even though the model leaks only one bit of information per query.

For the black-box attacks, we also evaluate how success of the attack depends on the adversary's auxiliary knowledge about the training dataset. For models trained on images, the adversary needs no auxiliary information and can simply use random images as synthetic augmented inputs. For models trained on text, we compare the accuracy of the attack when the adversary knows the exact vocabulary of the training texts and when the adversary uses a vocabulary compiled from a publicly available corpus.

In summary, using third-party code to train ML models on sensitive data is risky even if the code provider does not observe the training. We demonstrate how the vast memorization capacity of modern ML models can be abused to leak information even if the model is only released as a "black box," without significant impact on model-quality metrics such as accuracy and generalizability.

2 BACKGROUND

2.1 Machine Learning Pipelines

We focus for simplicity on the supervised learning setting, but our techniques can potentially be applied to unsupervised learning, too. A machine learning model is a function $f_\theta : \mathcal{X} \mapsto \mathcal{Y}$ parameterized by a bit string θ of *parameters*. We will sometimes abuse the notation and use f_θ and θ interchangeably. The input, or feature, space is $\mathcal{X}$, the output space is $\mathcal{Y}$. We focus on classification problems, where $\mathcal{X}$ is a d-dimensional vector space and $\mathcal{Y}$ is a discrete set of classes.

For our purposes, a machine learning pipeline consists of several steps shown in Figure 1. The pipeline starts with a set of labeled data points $D = \{(x_i, y_i)\}_{i=1}^{n'}$ where $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ for $1 \le i \le n'$. This set is partitioned into two subsets, training data D_{train} of size n and test data D_{test}.

Data augmentation. A common strategy for improving generalizability of ML models (i.e., their predictive power on inputs outside their training datasets) is to use data augmentation as an optional preprocessing step before training the model. The training data D_{train} is expanded with new data points generated using deterministic or randomized transformations. For example, an augmentation algorithm for images may take each training image and flip it horizontally or inject noises and distortions. The resulting expanded dataset D_{aug} is then used for training. Many libraries and machine learning platforms provide this functionality, including Keras [36], MXNET [56], DeepDetect [19], and indico [34].

Training and regularization. The (possibly augmented) dataset D_{aug} is taken as input by a (usually randomized) training algorithm $\mathcal{T}$, which also takes as input a configuration string γ called the hyperparameters. The training algorithm $\mathcal{T}$ outputs a set of parameters θ, which defines a model $f_\theta : \mathcal{X} \mapsto \mathcal{Y}$.

In order to find the optimal set of parameters θ for f, the training algorithm $\mathcal{T}$ tries to minimize a loss function $\mathcal{L}$ which penalizes the mismatches between true labels y and predicted labels produced by $f_\theta(x)$. Empirical risk minimization is the general framework for doing so, and uses the following objective function over D_{train}:

$$\min_\theta \ \Omega(\theta) + \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f_\theta(x_i))$$

where $\Omega(\theta)$ is a regularization term that penalizes model complexity and thus helps prevent models from overfitting.

Popular choices for Ω are norm-based regularizers, including l_2-norm $\Omega(\theta) = \lambda \sum_i \theta_i^2$ which penalizes the parameters for being too large, and l_1-norm $\Omega(\theta) = \lambda \sum_i |\theta_i|$ which adds sparsity to the parameters. The coefficient λ controls how much the regularization term affects the training objective.

There are many methods to optimize the above objective function. Stochastic gradient descent (SGD) and its variants are commonly used to train artificial neural networks, but our methods apply to other numerical optimization methods as well. SGD is an iterative method where at each step the optimizer receives a

small batch of training data and updates the model parameters θ according to the direction of the negative gradient of the objective function with respect to θ. Training is finished when the model converges to a local minimum where the gradient is close to zero.

Validation. We define *accuracy* of a model f_θ relative to some dataset D using 0-1 loss:

$$\text{acc}(\theta, D) = \sum_{(x,y)\in D} \frac{\mathbb{I}(f_\theta(x) = y)}{|D|}$$

where $\mathbb{I}$ is the function that outputs 1 if $f_\theta(x) = y$ and outputs zero otherwise. A trained model is validated by measuring its test accuracy $\text{acc}(\theta, D_{\text{test}})$. If the test accuracy is too low, validation may reject the model, outputting some error that we represent with a distinguished symbol $\perp$.

A related metric is the train-test gap. It is defined as the difference in accuracy on the training and test datasets:

$$\text{acc}(\theta, D_{\text{train}}) - \text{acc}(\theta, D_{\text{test}}) \ .$$

This gap measures how overfitted the model is to its training dataset.

Linear models. Support Vector Machines (SVM) [17] and logistic regression (LR) are popular for classification tasks such as text categorization [35] and other natural language processing problems [8]. We assume feature space $\mathcal{X} = \mathbb{R}^d$ for some dimension d.

In an SVM for binary classification with $\mathcal{Y} = \{-1, 1\}$, $\theta \in \mathcal{X}$, the model is given by $f_\theta(x) = \text{sign}(\theta^\top x)$, where the function sign returns whether the input is positive or negative. Traditionally training uses hinge loss, i.e., $\mathcal{L}(y, f_\theta(x)) = \max\{0, 1 - y\theta^\top x\}$. A typical regularizer for an SVM is the l_2-norm.

With LR, the parameters again consist of a vector in $\mathcal{X}$ and define the model $f_\theta(x) = \sigma(\theta^\top x)$ where $\sigma(x) = (1 + e^{-x})^{-1}$. In binary classification where the classes are $\{0, 1\}$, the output gives a value in [0,1] representing the probability that the input is classified as 1; the predicted class is taken to be 1 if $f_\theta(x) \geq 0.5$ and 0 otherwise. A typical loss function used during training is cross-entropy: $\mathcal{L}(y, f_\theta(x)) = y \cdot \log(f_\theta(x)) + (1 - y)\log(1 - f_\theta(x))$. A regularizer is optional and typically chosen empirically.

Linear models are typically efficient to train and the number of parameters is linear in the number of input dimensions. For tasks like text classification where inputs have millions of dimensions, models can thus become very large.

Deep learning models. Deep learning has become very popular for many ML tasks, especially related to computer vision and image recognition (e.g., [41, 46]). In deep learning models, f is composed of layers of non-linear transformations that map inputs to a sequence of intermediate states and then to the output. The parameters θ describe the weights used within each transformation. The number of parameters can become huge as the depth of the network increases.

Choices for the loss function and regularizer typically depend on the task. In classification tasks, if there are c classes in $\mathcal{Y}$, the last layer of the deep learning model is usually a probability vector with dimension c representing the likelihood that the input belongs to each class. The model outputs $\text{argmax} f_\theta(x)$ as the predicted class label. A common loss function for classification is negative log likelihood: $\mathcal{L}(y, f_\theta(x)) = -\sum_{i=1}^{c} t \cdot \log(f_\theta(x)_i)$, where t is 1 if the class label $y = i$ and 0 otherwise. Here $f_\theta(x)_i$ denotes the i^{th} component of the c-dimensional vector $f_\theta(x)$.

2.2 ML Platforms and Algorithm Providers

The popularity of machine learning (ML) has led to an explosion in the number of ML libraries, frameworks, and services. A data holder might use in-house infrastructure with a third-party ML library, or, increasingly, outsource model creation to a cloud service such as Google's Prediction API [27], Amazon ML [3], Microsoft's Azure ML [54], or a bevy of startups [10, 30, 55, 58]. These services automate much of the modern ML pipeline. Users can upload datasets, perform training, and make the resulting models available for use—all without understanding the details of model creation.

An ML algorithm provider (or simply *ML provider*) is the entity that provides ML training code to data holders. Many cloud services are ML providers, but some also operate marketplaces for training algorithms where clients pay for access to algorithms uploaded by third-party developers. In the marketplace scenario, the ML provider is the algorithm developer, not the platform operator.

Algorithmia [2] is a mature example of an ML marketplace. Developers can upload and list arbitrary programs (in particular, programs for ML training). A user can pay a developer for access to such a program and have the platform execute it on the user's data. Programs need not be open source, allowing the use of proprietary algorithms. The platform may restrict marketplace programs from accessing the Internet, and Algorithmia explicitly warns users that they should use only Internet-restricted programs if they are worried about leakage of their sensitive data.

These controls show that existing platform operators already focus on building trustworthy ML marketplaces. Software-based isolation mechanisms and network controls help prevent exfiltration of training data via conventional means. Several academic proposals have sought to construct even higher assurance ML platforms. For example, Zhai et al. [74] propose a cloud service with isolated environments in which one user supplies sensitive data, another supplies a secret training algorithm, and the cloud ensures that the algorithm cannot communicate with the outside world except by outputting a trained model. The explicit goal is to assure the data owner that the ML provider cannot exfiltrate sensitive training data. Advances in data analytics frameworks based on trusted hardware such as SGX [7, 61, 66] and cryptographic protocols based on secure multi-party computation (see Section 8) may also serve as the basis for secure ML platforms.

Even if the ML platform is secure (whether operated in-house or in a cloud), the algorithms supplied by the ML provider may not be trustworthy. Non-expert users may not audit open-source implementations or not understand what the code is doing. Audit may not be feasible for closed-source and proprietary implementations. Furthermore, libraries can be subverted, e.g., by compromising a code repository [37, 71] or a VM image [6, 14, 73]. In this paper, we investigate potential consequences of using untrusted training algorithms on a trusted platform.

3 THREAT MODEL

As explained in subsection 2.2, data holders often use other people's training algorithms to create models from their data. We thus focus on the scenario where a data holder (*client*) applies ML code

provided by an adversary (*ML provider*) to the client's data. We investigate ***if an adversarial ML provider can exfiltrate sensitive training data, even when his code runs on a secure platform?***

Client. The client has a dataset D sampled from the feature space $\mathcal{X}$ and wants to train a classification model f_θ on D, as described in subsection 2.1. We assume that the client wishes to keep D private, as would be the case when D is proprietary documents, sensitive medical images, etc.

The client applies a machine learning pipeline (see Figure 1) provided by the adversary to D_{train}, the training subset of D. This pipeline outputs a model, defined by its parameters θ. The client validates the model by measuring its accuracy on the test subset D_{test} and the test-train gap, *accepts* the model if it passes validation, and then publishes it by releasing θ or making an API interface to f_θ available for prediction queries. We refer to the former as white-box access and the latter as black-box access to the model.

Adversary. We assume that the ML pipeline shown in Figure 1 is controlled by the adversary. In general, the adversary controls the core training algorithm $\mathcal{T}$, but in this paper we assume that $\mathcal{T}$ is a conventional, benign algorithm and focus on smaller modifications to the pipeline. For example, the adversary may provide a malicious data augmentation algorithm $\mathcal{A}$, or else a malicious regularizer Ω, while keeping $\mathcal{T}$ intact. The adversary may also modify the parameters θ after they have been computed by $\mathcal{T}$.

The adversarially controlled pipeline can execute entirely on the client side—for example, if the client runs the adversary's ML library locally on his data. It can also execute on a third-party platform, such as Algorithmia. We assume that the environment running the algorithms is secured using software [2, 74] or hardware [61, 66] isolation or cryptographic techniques. In particular, the adversary cannot communicate directly with the training environment; otherwise he can simply exfiltrate data over the network.

Adversary's objectives. The adversary's main objective is to infer as much as of the client's private training dataset D as possible.

Some existing models already reveal parts of the training data. For example, nearest neighbors classifiers and SVMs explicitly store some training data points in θ. Deep neural networks and classic logistic regression are not known to leak any specific training information (see Section 8 for more discussion about privacy of the existing training algorithms). Even with SVMs, the adversary may want to exfilitrate more, or different, training data than revealed by θ in the default setting. For black-box attacks, in which the adversary does not have direct access to θ, there is no known way to extract the sensitive data stored in θ by SVMs and nearest neighbor models.

Other, more limited, objectives may include inferring the presence of a known input in the dataset D (this problem is known as membership inference), partial information about D (e.g., the presence of a particular face in some image in D), or metadata associated with the elements of D (e.g., geolocation data contained in the digital photos used to train an image recognition model). While we do not explore these in the current paper, our techniques can be used directly to achieve these goals. Furthermore, they require extracting much less information than is needed to reconstruct entire training inputs, therefore we expect our techniques will be even more effective.

Assumptions about the training environment. The adversary's pipeline has unrestricted access to the training data D_{train} and the model θ being trained. As mentioned above, we focus on the scenarios where the adversary does *not* modify the training algorithm $\mathcal{T}$ but instead (a) modifies the parameters θ of the resulting model, or (b) uses $\mathcal{A}$ to augment D_{train} with additional training data, or (c) applies his own regularizer Ω while $\mathcal{T}$ is executing.

We assume that the adversary can observe neither the client's data, nor the execution of the adversary's ML pipeline on this data, nor the resulting model (until it is published by the client). We assume that the adversary's code incorporated into the pipeline is isolated and confined so that it has no way of communicating with or signaling to the adversary while it is executing. We also assume that all state of the training environment is erased after the model is accepted or rejected.

Therefore, the only way the pipeline can leak information about the dataset D_{train} to the adversary is by (1) forcing the model θ to somehow "memorize" this information and (2) ensuring that θ passes validation.

Access to the model. With *white-box* access, the adversary receives the model directly. He can directly inspect all parameters in θ, but not any temporary information used during the training. This scenario arises, for example, if the client publishes θ.

With *black-box* access, the adversary has input-output access to θ: given any input x, he can obtain the model's output $f_\theta(x)$. For example, the model could be deployed inside an app and the adversary uses this app as a customer. Therefore, we focus on the simplest (and hardest for the adversary) case where he learns only the class label assigned by the model to his inputs, not the entire prediction vector with a probability for each possible class.

4 WHITE-BOX ATTACKS

In a white-box attack, the adversary can see the parameters of the trained model. We thus focus on directly encoding information about the training dataset in the parameters. The main challenge is how to have the resulting model accepted by the client. In particular, the model must have high accuracy on the client's classification task when applied to the test dataset.

4.1 LSB Encoding

Many studies have shown that high-precision parameters are not required to achieve high performance in machine learning models [29, 48, 64]. This observation motivates a very direct technique: simply encode information about the training dataset in the least significant (lower) bits of the model parameters.

Encoding. Algorithm 1 describes the encoding method. First, train a benign model using a conventional training algorithm $\mathcal{T}$, then post-process the model parameters θ by setting the lower b bits of each parameter to a bit string s extracted from the training data, producing modified parameters θ'.

Extraction. The secret string s can be either compressed raw data from D_{train}, or any information about D_{train} that the adversary wishes to capture. The length of s is limited to ℓb, where ℓ is the number of parameters in the model.

Algorithm 1 LSB encoding attack

Input: Training dataset D_{train}, a benign ML training algorithm $\mathcal{T}$, number of bits b to encode per parameter.
Output: ML model parameters θ' with secrets encoded in the lower b bits.
$\theta \leftarrow \mathcal{T}(D_{\text{train}})$
$\ell \leftarrow$ number of parameters in θ
$s \leftarrow$ **ExtractSecretBitString**$(D_{\text{train}}, \ell b)$
$\theta' \leftarrow$ set the lower b bits in each parameter of θ to a substring of s of length b.

Algorithm 2 SGD with correlation value encoding

Input: Training dataset $D_{\text{train}} = \{(x_j, y_j)\}_{i=1}^{n}$, a benign loss function $\mathcal{L}$, a model f, number of epochs T, learning rate η, attack coefficient λ_c, size of mini-batch q.
Output: ML model parameters θ correlated to secrets.
$\theta \leftarrow$ **Initialize**(f)
$\ell \leftarrow$ number of parameters in θ
$s \leftarrow$ **ExtractSecretValues**(D, ℓ)
for $t = 1$ to T **do**
  for each mini-batch $\{(x_j, y_j)\}_{j=1}^{q} \subset D_{\text{train}}$ **do**
    $g_t \leftarrow \nabla_\theta \frac{1}{m} \sum_{j=1}^{q} \mathcal{L}(y_j, f(x_j, \theta)) + \nabla_\theta C(\theta, s)$
    $\theta \leftarrow$ **UpdateParameters**(η, θ, g_t)
  end for
end for

Decoding. Simply read the lower bits of the parameters θ' and interpret them as bits of the secret.

4.2 Correlated Value Encoding

Another approach is to gradually encode information while training model parameters. The adversary can add a malicious term to the loss function $\mathcal{L}$ (see Section 2.1) that maximizes the correlation between the parameters and the secret s that he wants to encode.

In our experiments, we use the negative absolute value of the Pearson correlation coefficient as the extra term in the loss function. During training, it drives the gradient direction towards a local minimum where the secret and the parameters are highly correlated. Algorithm 2 shows the template of the SGD training algorithm with the malicious regularization term in the loss function.

Encoding. First extract the vector of secret values $s \in \mathbb{R}^\ell$ from the training data, where ℓ is the number of parameters. Then, add a malicious correlation term C to the loss function where

$$C(\theta, s) = -\lambda_c \cdot \frac{\left|\sum_{i=1}^{\ell}(\theta_i - \bar{\theta})(s_i - \bar{s})\right|}{\sqrt{\sum_{i=1}^{\ell}(\theta_i - \bar{\theta})^2} \cdot \sqrt{\sum_{i=1}^{\ell}(s_i - \bar{s})^2}}\ .$$

In the above expression, λ_c controls the level of correlation and $\bar{\theta}, \bar{s}$ are the mean values of θ and s, respectively. The larger C, the more correlated θ and s. During optimization, the gradient of C with respect to θ is used for parameter update.

Observe that the C term resembles a conventional *regularizer* (see Section 2.1), commonly used in machine learning frameworks. The difference from the norm-based regularizers discussed previously is that we assign a weight to each parameter in C that depends on the secrets that we want the model to memorize. This term skews the parameters to a space that correlates with these secrets. The parameters found with the malicious regularizer will not necessarily be the same as with a conventional regularizer, but the malicious regularizer has the same effect of confining the parameter space to a less complex subspace [72].

Extraction. The method for extracting sensitive data s from the training data D_{train} depends on the nature of the data. If the features in the raw data are all numerical, then raw data can be directly used as the secret. For example, our method can force the parameters to be correlated with the pixel intensity of training images.

For non-numerical data such as text, we use data-dependent numerical values to encode. We map each unique token in the vocabulary to a low-dimension pseudorandom vector and correlate the model parameters with these vectors. Pseudorandomness ensures that the adversary has a fixed mapping between tokens and vectors and can uniquely recover the token given a vector.

Decoding. If all features in the sensitive data are numerical and within the same range (for images raw pixel intensity values are in the [0, 255] range), the adversary can easily map the parameters back to feature space because correlated parameters are approximately linear transformation of the encoded feature values.

To decode text documents, where tokens are converted into pseudorandom vectors, we perform a brute-force search for the tokens whose corresponding vectors are most correlated with the parameters. More sophisticated approaches (e.g., error-correcting codes) should work much better, but we do not explore them in this paper.

We provide more details about these decoding procedures for specific datasets in Section 6.

4.3 Sign Encoding

Another way to encode information in the model parameters is to interpret their signs as a bit string, e.g., a positive parameter represents 1 and a negative parameter represents 0. Machine learning algorithms typically do not impose constraints on signs, but the adversary can modify the loss function to force most of the signs to match the secret bit string he wants to encode.

Encoding. Extract a secret binary vector $s \in \{-1, 1\}^\ell$ from the training data, where ℓ is the number of parameters in θ, and constrain the sign of θ_i to match s_i. This encoding method is equivalent to solving the following constrained optimization problem:

$$\min_{\theta}\ \Omega(\theta) + \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y_i, f(x_i, \theta))$$
$$\text{such that}\quad \theta_i s_i > 0 \text{ for } i = 1, 2, \ldots, \ell$$

Solving this constrained optimization problem can be tricky for models like deep neural networks due to its complexity. Instead, we can relax it to an unconstrained optimization problem using the penalty function method [60]. The idea is to convert the constraints to a penalty term added to the objective function, where the term penalizes the objective if the constraints are not met. In our case,

we define the penalty term P as follows:

$$P(\theta, s) = \frac{\lambda_s}{\ell} \sum_{i=1}^{\ell} |\max(0, -\theta_i s_i)| \ .$$

In the above expression, λ_s is a hyperparameter that controls the magnitude of the penalty. Zero penalty is added when θ_i and s_i have the same sign, $|\theta_i s_i|$ is the penalty otherwise.

The attack algorithm is mostly identical to Algorithm 2 with two lines changed. Line 5 becomes $s \leftarrow$ **ExtractSecretSigns**(D, ℓ), where s is a binary vector of length ℓ instead of a vector of real numbers. In line 9, P replaces the correlation term C. Similar to the correlation term, P changes the direction of the gradient to drive the parameters towards the subspace in $\mathbb{R}^\ell$ where all sign constraints are met. In practice, the solution may not converge to a point where all constraints are met, but our algorithm can get most of the encoding correct if λ_s is large enough.

Observe that P is very similar to l_1-norm regularization. When all signs of the parameters do not match, the term P is exactly the l_1-norm because $-\theta_i s_i$ is always positive. Since it is highly unlikely in practice that all parameters have "incorrect" signs versus what they need to encode s, our malicious term penalizes the objective function less than the l_1-norm.

Extraction. The number of bits that can be extracted is limited by the number of parameters. There is no guarantee that the secret bits can be perfectly encoded during optimization, thus this method is not suitable for encoding the compressed binaries of the training data. Instead, it can be used to encode the bit representation of the raw data. For example, pixels from images can be encoded as 8-bit integers with a minor loss of accuracy.

Decoding. Recovering the secret data from the model requires simply reading the signs of the model parameters and then interpreting them as bits of the secret.

5 BLACK-BOX ATTACKS

Black-box attacks are more challenging because the adversary cannot see the model parameters and instead has access only to a prediction API. We focus on the (harder) setting in which the API, in response to an adversarially chosen feature vector x, applies $f_\theta(x)$ and outputs the corresponding classification label (but not the associated confidence values). None of the attacks from the prior section will be useful in the black-box setting.

5.1 Abusing Model Capacity

We exploit the fact that modern machine learning models have vast capacity for memorizing arbitrarily labeled data [75].

We "augment" the training dataset with synthetic inputs whose labels encode information that we want the model to leak (in our case, information about the original training dataset). When the model is trained on the augmented dataset—even using a conventional training algorithm—it becomes overfitted to the synthetic inputs. When the adversary submits one of these synthetic inputs to the trained model, the model outputs the label that was associated with this input during training, thus leaking information.

Algorithm 3 Capacity-abuse attack

1: **Input:** Training dataset D_{train}, a benign ML training algorithm $\mathcal{T}$, number of inputs m to be synthesized.
2: **Output:** ML model parameters θ that memorize the malicious synthetic inputs and their labels.
3: $D_{\text{mal}} \leftarrow$ **SynthesizeMaliciousData**(D_{train}, m)
4: $\theta \leftarrow \mathcal{T}(D_{\text{train}} \cup D_{\text{mal}})$

Algorithm 3 outlines the attack. First, synthesize a malicious dataset D_{mal} whose labels encode secrets about D_{train}. Then train the model on the union of D_{train} and D_{mal}.

Observe that **the entire training pipeline is exactly the same as in benign training**. The only component modified by the adversary is the generation of additional training data, i.e., the augmentation algorithm $\mathcal{A}$. Data augmentation is a very common practice for boosting the performance of machine learning models [41, 69].

5.2 Synthesizing Malicious Augmented Data

Ideally, each synthetic data point can encode $\lfloor \log_2(c) \rfloor$ bits of information where c is the number of classes in the output space of the model. Algorithm 4 outlines our synthesis method. Similar to the white-box attacks, we first extract a secret bit string s from D_{train}. We then deterministically synthesize one data point for each substring of length $\lfloor \log_2(c) \rfloor$ in s.

Algorithm 4 Synthesizing malicious data

1: **Input:** A training dataset D_{train}, number of inputs to be synthesized m, auxiliary knowledge K.
2: **Output:** Synthesized malicious data D_{mal}
3: $D_{\text{mal}} \leftarrow \emptyset$
4: $s \leftarrow$ **ExtractSecretBitString**(D_{train}, m)
5: $c \leftarrow$ number of classes in D_{train}
6: **for** each $\lfloor \log_2(c) \rfloor$ bits s' in s **do**
7: $\quad x_{\text{mal}} \leftarrow$ **GenData**(K)
8: $\quad y_{\text{mal}} \leftarrow$ **BitsToLabel**(s')
9: $\quad D_{\text{mal}} \leftarrow D_{\text{mal}} \cup \{(x_{\text{mal}}, y_{\text{mal}})\}$
10: **end for**

Different types of data require different synthesis methods.

Synthesizing images. We assume no auxiliary knowledge for synthesizing images. The adversary can use any suitable **GenData** method: for example, generate pseudorandom images using the adversary's choice of pseudorandom function (PRF) (e.g., HMAC [39]) or else create sparse images where only one pixel is filled with a (similarly generated) pseudorandom value.

We found the latter technique to be very effective in practice. **GenData** enumerates all pixels in an image and, for each pixel, creates a synthetic image where the corresponding pixel is set to the pseudorandom value while other pixels are set to zero. The same technique can be used with multiple pixels in each synthetic image.

Synthesizing text. We consider two scenarios for synthesizing text documents.

If the adversary knows the exact vocabulary of the training dataset, he can use this vocabulary as the auxiliary knowledge

in **GenData**. A simple deterministic implementation of **GenData** enumerates the tokens in the auxiliary vocabulary in a certain order. For example, **GenData** can enumerate all singleton tokens in lexicographic order, then all pairs of tokens in lexicographic order, and so on until the list is as long as the number of synthetic documents needed. Each list entry is then set to be a text in the augmented training dataset.

If the adversary does not know the exact vocabulary, he can collect frequently used words from some public corpus as the auxiliary vocabulary for generating synthetic documents. In this case, a deterministic implementation of **GenData** pseudorandomly (with a seed known to the adversary) samples words from the vocabulary until generating the desired number of documents.

To generate a document in this case, our simple synthesis algorithm samples a constant number of words (50, in our experiments) from the public vocabulary and joins them as a single document. The order of the words does not matter because the feature extraction step only cares whether a given word occurs in the document or not.

This synthesis algorithm may occasionally generate documents consisting only of words that do not occur in the model's actual vocabulary. Such words will typically be ignored in the feature extraction phase, thus the resulting documents will have empty features. If the attacker does not know the model's vocabulary, he cannot know if a particular synthetic document consists only of out-of-vocabulary words. This can potentially degrade both the test accuracy and decoding accuracy of the model.

In Section 6.7, we empirically measure the accuracy of the capacity-abuse attack with a public vocabulary.

Decoding memorized information. Because our synthesis methods for augmented data are deterministic, the adversary can replicate the synthesis process and query the trained model with the same synthetic inputs as were used during training. If the model is overfitted to these inputs, the labels returned by the model will be exactly the same labels that were associated with these inputs during training, i.e., the encoded secret bits.

If a model has sufficient capacity to achieve good accuracy and generalizability on its original training data *and* to memorize malicious training data, then $\text{acc}(\theta, D_{\text{mal}})$ will be near perfect, leading to low error when extracting the sensitive data.

5.3 Why Capacity Abuse Works

Deep learning models have such a vast memorization capacity that they can essentially express any function to fit the data [75]. In our case, the model is fitted not just to the original training dataset but also to the synthetic data which is (in essence) randomly labeled. If the test accuracy on the original data is high, the model is accepted. If the training accuracy on the synthetic data is high, the adversary can extract information from the labels assigned to these inputs.

Critically, these two goals are not in conflict. Training on maliciously augmented datasets thus produces models that have high quality on their original training inputs yet leak information on the augmented inputs.

In the case of SVM and LR models, we focus on high-dimensional and sparse data (natural-language text). Our synthesis method also produces very sparse inputs. Empirically, the likelihood that a synthetic input lies on the wrong side of the hyperplane (classifier) becomes very small in this high-dimensional space.

Dataset	Data size			f	Num params	Test acc
	n	d	bits			
CIFAR10	50K	3072	1228M	RES	460K	92.89
LFW	10K	8742	692M	CNN	880K	87.83
FaceScrub (G)	57K	7500	3444M	RES	460K	97.44
FaceScrub (F)					500K	90.08
News	11K	130K	176M	SVM	2.6M	80.58
				LR		80.51
IMDB	25K	300K	265M	SVM	300K	90.13
				LR		90.48

Table 1: Summary of datasets and models. n is the size of the training dataset, d is the number of input dimensions. RES stands for Residual Network, CNN for Convolutional Neural Network. For FaceScrub, we use the gender classification task (G) and face recognition task (F).

6 EXPERIMENTS

We evaluate our attack methods on benchmark image and text datasets, using, respectively, gray-scale training images and ordered tokens as the secret to be memorized in the model.

For each dataset and task, we first train a benign model using a conventional training algorithm. We then train and evaluate a malicious model for each attack method. We assume that the malicious training algorithm has a hard-coded secret that can be used as the key for a pseudorandom function or encryption.

All ML models and attacks were implemented in Python 2.7 with Theano [70] and Lasagne [20]. The experiments were conducted on a machine with two 8-core Intel i7-5960X CPUs, 64 GB RAM, and three Nvidia TITAN X (Pascal) GPUs with 12 GB VRAM each.

6.1 Datasets and Tasks

Table 1 summarizes the datasets, models, and classification tasks we used in our experiments. We use as stand-ins for sensitive data several representative, publicly available image and text datasets.

CIFAR10 is an object classification dataset with 50,000 training images (10 categories, 5,000 images per category) and 10,000 test images [40]. Each image has 32x32 pixels, each pixel has 3 values corresponding to RGB intensities.

Labeled Faces in the Wild (LFW) contains 13,233 images for 5,749 individuals [33, 45]. We use 75% for training, 25% for testing. For the gender classification task, we use additional attribute labels [42]. Each image is rescaled to 67x42 RGB pixels from its original size, so that all images have the same size.

FaceScrub is a dataset of URLs for 100K images [59]. The tasks are face recognition and gender classification. Some URLs have expired, but we were able to download 76,541 images for 530 individuals. We use 75% for training, 25% for testing. Each image is rescaled to 50x50 RGB pixels from its original size.

20 Newsgroups is a corpus of 20,000 documents classified into 20 categories [44]. We use 75% for training, 25% for testing.

IMDB Movie Reviews is a dataset of 50,000 reviews labeled with positive or negative sentiment [52]. The task is (binary) sentiment analysis. We use 50% for training, 50% for testing.

6.2 ML Models

Convolutional Neural Networks. Convolutional Neural Networks (CNN) [47] are composed of a series of convolution operations as building blocks which can extract spatial-invariant features. The filters in these convolution operations are the parameters to be learned. We use a 5-layer CNN for gender classification on the LFW dataset. The first three layers are convolution layers (32 filters in the first layer, 64 in the second, 128 in the third) followed by a max-pooling operation which reduces the size of convolved features by half. Each filter in the convolution layer is 3x3. The convolution output is connected to a fully-connected layer with 256 units. The latter layer connects to the output layer which predicts gender.

For the hyperparameters, we set the mini-batch size to be 128, learning rate to be 0.1, and use SGD with Nesterov Momentum for optimizing the loss function. We also use the l_2-norm as the regularizer with λ set to 10^{-5}. We set the number of epochs for training to 100. In epochs 40 and 60, we decrease the learning rate by a factor of 0.1 for better convergence. This configuration is inherited from the residual-network implementation in Lasagne.[1]

Residual Networks. Residual networks (RES) [31] overcome the gradient vanishing problem when optimizing very deep CNNs by adding identity mappings from lower layers to high layers. These networks achieved state-of-the-art performance on many benchmark vision datasets in 2016.

We use a 34-layer residual network for CIFAR10 and FaceScrub. Although the network has fewer parameters than CNN, it is much deeper and can learn better representations of the input data. The hyperparameters are the same as for the CNN.

Bag-of-Words and Linear Models. For text datasets, we use a popular pipeline that extracts features using Bag-of-Words (BOW) and trains linear models.

BOW maps each text document into a vector in $\mathbb{R}^{|V|}$ where V is the vocabulary of tokens that appear in the corpus. Each dimension represents the count of that token in the document. The vectors are extremely sparse because only a few tokens from V appear in any given document.

We then feed the BOW vectors into an SVM or LR model. For 20 Newsgroups, there are 20 categories and we apply the One-vs-All method to train 20 binary classifiers to predict whether a data point belongs to the corresponding class or not. We train linear models using AdaGrad [23], a variant of SGD with adaptive adjustment to the learning rate of each parameter. We set the mini-batch size to 128, learning rate to 0.1, and the number of epochs for training to 50 as AdaGrad converges very fast on these linear models.

6.3 Evaluation Metrics

Because we aim to encode secrets in a model while preserving its quality, we measure both the attacker's decoding accuracy and the model's classification accuracy on the test data for its primary task (accuracy on the training data is over 98% in all cases). Our attacks introduce minor stochasticity into training, thus accuracy of maliciously trained models occasionally exceeds that of conventionally trained models.

Metrics for decoding images. For images, we use mean absolute pixel error (MAPE). Given a decoded image x' and the original image x with k pixels, MAPE is $\frac{1}{k}\sum_{i=1}^{k}|x_i - x'_i|$. Its range is [0, 255], where 0 means the two images are identical and 255 means every pair of corresponding pixels has maximum mismatch.

Metrics for decoding text. For text, we use precision (percentage of tokens from the decoded document that appear in the original document) and recall (percentage of tokens from the original document that appear in the decoded document). To evaluate similarity between the decoded and original documents, we also measure their cosine similarity based on their feature vectors constructed from the BOW model with the training vocabulary.

[1] https://github.com/Lasagne/Recipes/blob/master/modelzoo/resnet50.py

Dataset	f	b	Encoded bits	Test acc $\pm\delta$
CIFAR10	RES	18	8.3M	92.75 −0.14
LFW	CNN	22	17.6M	87.69 −0.14
FaceScrub (G)	RES	20	9.2M	97.33 −0.11
FaceScrub (F)		18	8.3M	89.95 −0.13
News	SVM	22	57.2M	80.60 +0.02
	LR			80.40 −0.11
IMDB	SVM	22	6.6M	90.12 −0.01
	LR			90.31 −0.17

Table 2: Results of the LSB encoding attack. Here f is the model used, b is the maximum number of lower bits used beyond which accuracy drops significantly, δ is the difference with the baseline test accuracy.

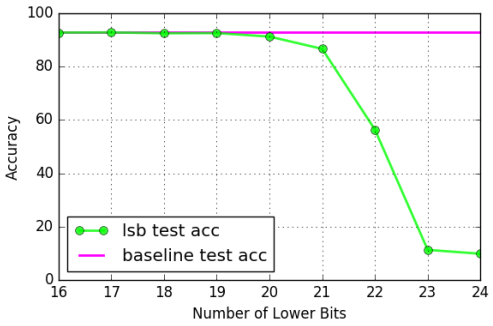

Figure 2: Test accuracy of the CIFAR10 model with different amounts of lower bits used for the LSB attack.

6.4 LSB Encoding Attack

Table 2 summarizes the results for the LSB encoding attack.

Encoding. For each task, we compressed a subset of the training data, encrypted it with AES in CBC mode, and wrote the ciphertext bits into the lower bits of the parameters of a benignly trained

Dataset	f	λ_c	Test acc $\pm\delta$	Decode MAPE
CIFAR10	RES	0.1	92.90 +0.01	52.2
		1.0	91.09 −1.80	29.9
LFW	CNN	0.1	87.94 +0.11	35.8
		1.0	87.91 −0.08	16.6
FaceScrub (G)	RES	0.1	97.32 −0.11	24.5
		1.0	97.27 −0.16	15.0
FaceScrub (F)		0.1	90.33 +0.25	52.9
		1.0	88.64 −1.44	38.6

Dataset	f	λ_c	Test acc $\pm\delta$	Decode			
				τ	Pre	Rec	Sim
News	SVM	0.1	80.42 −0.16	0.85	0.85	0.70	0.84
				0.95	1.00	0.56	0.78
	LR	1.0	80.35 −0.16	0.85	0.90	0.80	0.88
				0.95	1.00	0.65	0.83
IMDB	SVM	0.5	89.47 −0.66	0.85	0.90	0.73	0.88
				0.95	1.00	0.16	0.51
	LR	1.0	89.33 −1.15	0.85	0.98	0.94	0.97
				0.95	1.00	0.73	0.90

Table 3: Results of the correlated value encoding attack. Here λ_c is the coefficient for the correlation term in the objective function and δ is the difference with the baseline test accuracy. For image data, decode MAPE is the mean absolute pixel error. For text data, τ is the decoding threshold for the correlation value. Pre is precision, Rec is recall, and Sim is cosine similarity.

Dataset	f	λ_s	Test acc $\pm\delta$	Decode MAPE
CIFAR10	RES	10.0	92.96 +0.07	36.00
		50.0	92.31 −0.58	3.52
LFW	CNN	10.0	88.00 +0.17	37.30
		50.0	87.63 −0.20	5.24
FaceScrub (G)	RES	10.0	97.31 −0.13	2.51
		50.0	97.45 +0.01	0.15
FaceScrub (F)		10.0	89.99 −0.09	39.85
		50.0	87.45 −2.63	7.46

Dataset	f	λ_s	Test acc $\pm\delta$	Decode		
				Pre	Rec	Sim
News	SVM	5.0	80.42 −0.16	0.56	0.66	0.69
		7.5	80.49 −0.09	0.71	0.80	0.82
	LR	5.0	80.45 −0.06	0.57	0.67	0.70
		7.5	80.20 −0.31	0.63	0.73	0.75
IMDB	SVM	5.0	89.32 −0.81	0.60	0.68	0.75
		7.5	89.08 −1.05	0.66	0.75	0.81
	LR	5.0	89.52 −0.92	0.67	0.76	0.81
		7.5	89.27 −1.21	0.76	0.83	0.88

Table 4: Results of the sign encoding attack. Here λ_s is the coefficient for the correlation term in the objective function.

model. The fourth column in Table 2 shows the number of bits we can use before test accuracy drops significantly.

Decoding. Decoding is always perfect because we use lossless compression and no errors are introduced during encoding. For the 20 Newsgroup model, the adversary can successfully extract about 57 Mb of compressed data, equivalent to 70% of the training dataset.

Test accuracy. In our implementation, each model parameter is a 32-bit floating-point number. Empirically, b under 20 does not decrease test accuracy on the primary task for most datasets. Binary classification on images (LFW, FaceScrub Gender) can endure more loss of precision. For multi-class tasks, test accuracy drops significantly when b exceeds 20 as shown for CIFAR10 in Figure 2.

6.5 Correlated Value Encoding Attack

Table 3 summarizes the results for this attack.

Image encoding and decoding. We correlate model parameters with the pixel intensity of gray-scale training images. The number of parameters limits the number of images that can be encoded in this way: 455 for CIFAR10, 200 for FaceScrub, 300 for LFW.

We decode images by mapping the correlated parameters back to pixel space (if correlation is perfect, the parameters are simply linearly transformed images). To do so given a sequence of parameters, we map the minimum parameter to 0, maximum to 255, and other parameters to the corresponding pixel value using min-max scaling. We obtain an approximate original image after transformation if the correlation is positive and an approximate inverted original image if the correlation is negative.

After the transformation, we measure the mean absolute pixel error (MAPE) for different choices of λ_c, which controls the level of correlation. We find that to recover reasonable images, λ_c needs to be over 1.0 for all tasks. For a fixed λ_c, errors are smaller for binary classification than for multi-class tasks. Examples of reconstructed images are shown in Figure 3 for the FaceScrub dataset.

Text encoding and decoding. To encode, we generate a pseudorandom, d'-dimensional vector of 32-bit floating point numbers for each token in the vocabulary of the training corpus. Then, given a training document, we use the pseudorandom vectors for the first 100 tokens in that document as the secret to correlate with the model parameters. We set d' to 20. Encoding one document thus requires up to 2000 parameters, allowing us to encode around 1300 documents for 20 Newsgroups and 150 for IMDB.

To decode, we first reproduce the pseudorandom vectors for each token used during training. For each consecutive part of the parameters that should match a token, we decode by searching for a token whose corresponding vector is best correlated with the parameters. We set a threshold value τ and if the correlation value is above τ, we accept this token and reject otherwise.

Table 3 shows the decoding results for different τ. As expected, larger τ increases precision and reduces recall. Empirically, $\tau = 0.85$ yields high-quality decoded documents (see examples in Table 5).

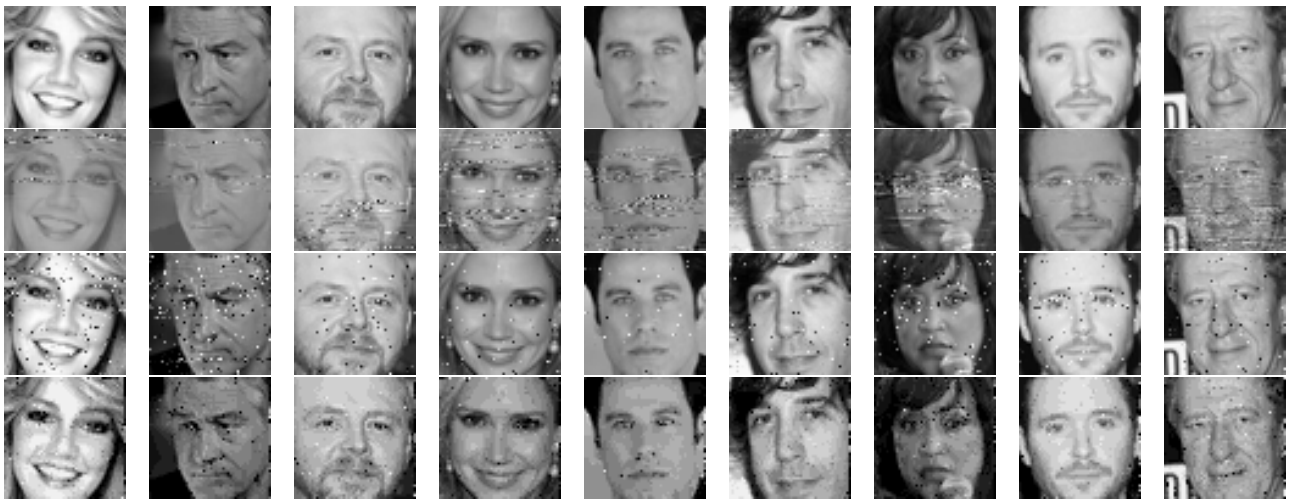

Figure 3: Decoded examples from all attacks applied to models trained on the FaceScrub gender classification task. First row is the ground truth. Second row is the correlated value encoding attack (λ_c=1.0, MAPE=15.0). Third row is the sign encoding attack (λ_s=10.0, MAPE=2.51). Fourth row is the capacity abuse attack (m=110K, MAPE=10.8).

Test accuracy. Models with a lower decoding error also have lower test accuracy. For binary classification tasks, we can keep MAPE reasonably low while reducing test accuracy by 0.1%. For CIFAR10 and FaceScrub face recognition, lower MAPE requires larger λ_c, which in turn reduces test accuracy by more than 1%.

For 20 Newsgroups, test accuracy drops only by 0.16%. For IMDB, the drop is more significant: 0.66% for SVM and 1.15% for LR.

6.6 Sign Encoding Attack

Table 4 summarizes the results of the sign encoding attack.

Image encoding and decoding. As mentioned in Section 4.3, the sign encoding attack may not encode all bits correctly. Therefore, instead of the encrypted, compressed binaries that we used for LSB encoding, we use the bit representation of the raw pixels of the gray-scale training images as the string to be encoded. Each pixel is an 8-bit unsigned integer. The encoding capacity is thus $\frac{1}{8}$ of the correlated value encoding attack. We can encode 56 images for CIFAR10, 25 images for FaceScrub and 37 images for LFW.

To reconstruct pixels, we assemble the bits represented in the parameter signs. With $\lambda_s = 50$, MAPE is small for all datasets. For gender classification on FaceScrub, the error can be smaller than 1, i.e., reconstruction is nearly perfect.

Text encoding and decoding. We construct a bit representation for each token using its index in the vocabulary. The number of bits per token is $\lceil \log_2(|V|) \rceil$, which is 17 for both 20 Newsgroups and IMDB. We encode the first 100 words in each document and thus need a total of 1,700 parameter signs per document. We encode 1530 documents for 20 Newsgroups and 180 for IMDB in this way.

To reconstruct tokens, we use the signs of 17 consecutive parameters as the index into the vocabulary. Setting $\lambda_s \geq 5$ yields good results for most tasks (see examples in Table 5). Decoding is less accurate than for the correlated value encoding attack. The reason is that signs need to be encoded almost perfectly to recover high-quality documents; even if 1 bit out of 17 is wrong, our decoding produces a completely different token. More sophisticated, error-correcting decoding techniques can be applied here, but we leave this to future work.

Test accuracy. This attack does not significantly affect the test accuracy of binary classification models on image datasets. For LFW and CIFAR10, test accuracy occasionally increases. For multi-class tasks, when λ_s is large, FaceScrub face recognition degrades by 2.6%, while the CIFAR10 model with $\lambda_s = 50$ still generalizes well.

For 20 Newsgroups, test accuracy changes by less than 0.5% for all values of λ_s. For IMDB, accuracy decreases by around 0.8% to 1.2% for both SVM and LR.

6.7 Capacity Abuse Attack

Table 6 summarizes the results.

Image encoding and decoding. We could use the same technique as in the sign encoding attack, but for a binary classifier this requires 8 synthetic inputs per each pixel. Instead, we encode an approximate pixel value in 4 bits. We map a pixel value $p \in \{0, \ldots, 255\}$ to $p' \in \{0, \ldots, 15\}$ (e.g., map 0-15 in p to 0 in p') and use 4 synthetic data points to encode p'. Another possibility (not evaluated in this paper) would be to encode every other pixel and recover the image by interpolating the missing pixels.

We evaluate two settings of m, the number of synthesized data points. For LFW, we can encode 3 images for m = 34K and 5 images for m = 58K. For FaceScrub gender classification, we can encode 11 images for m = 110K and 17 images for m = 170K. While these numbers may appear low, this attack works in a black-box setting against a binary classifier, where the adversary aims to recover information from a *single output bit*. Moreover, for many tasks (e.g., medical image analysis) recovering even a single training input constitutes a serious privacy breach. Finally, if the attacker's goal is to recover not the raw images but some other information about

Ground Truth	Correlation Encoding ($\lambda_c = 1.0$)	Sign Encoding ($\lambda_s = 7.5$)	Capacity Abuse (m = 24K)
has only been week since saw my first john waters film female trouble and wasn sure what to expect	it natch only been week since saw my first john waters film female trouble and wasn sure what to expect	it has peering been week saw mxyzptlk first john waters film bloch trouble and wasn sure what to extremism the	it has peering been week saw my first john waters film female trouble and wasn sure what to expect the
in brave new girl holly comes from small town in texas sings the yellow rose of texas at local competition	in chasing new girl holly comes from willed town in texas sings the yellow rose of texas at local competition	in brave newton girl hoists comes from small town impressible texas sings urban rosebud of texas at local obsess and	in brave newton girl holly comes from small town in texas sings the yellow rose of texas at local competition
maybe need to have my head examined but thought this was pretty good movie the cg is not too bad	maybe need to have my head examined but thought this was pretty good movie the cg pirouetting not too bad	maybe need to enjoyed my head hippo but tiburon wastage pretty good movie the cg is northwest too bad have	maybe need to have my head examined but thoughout tiburon was pretty good movie the cg is not too bad
was around when saw this movie first it wasn so special then but few years later saw it again and	was around when saw this movie martine it wasn so special then but few years later saw it again and	was around saw this movie first possession tributed so special zellweger but few years linette saw isoyc again and that	was around when saw this movie first it wasn soapbox special then but few years later saw it again and

Table 5: Decoded text examples from all attacks applied to LR models trained on the IMDB dataset.

Dataset	f	m	$\frac{m}{n}$	Test Acc $\pm\delta$	Decode MAPE
CIFAR10	RES	49K	0.98	92.21 −0.69	7.60
		98K	1.96	91.48 −1.41	8.05
LFW	CNN	34K	3.4	88.03 +0.20	18.6
		58K	5.8	88.17 +0.34	22.4
FaceScrub (G)	RES	110K	2.0	97.08 −0.36	10.8
		170K	3.0	96.94 −0.50	11.4
FaceScrub (F)		55K	1.0	87.46 −2.62	7.62
		110K	2.0	86.36 −3.72	8.11

Dataset	f	m	$\frac{m}{n}$	Test Acc $\pm\delta$	Decode Pre	Decode Rec	Decode Sim
News	SVM	11K	1.0	80.53 −0.07	1.0	1.0	1.0
		33K	3.0	79.77 −0.63	0.99	0.99	0.99
	LR	11K	1.0	80.06 −0.45	0.98	0.99	0.99
		33K	3.0	79.94 −0.57	0.95	0.97	0.97
IMDB	SVM	24K	0.95	89.82 −0.31	0.90	0.94	0.96
		75K	3.0	89.05 −1.08	0.89	0.93	0.95
	LR	24K	0.95	89.90 −0.58	0.87	0.92	0.95
		75K	3.0	89.26 −1.22	0.86	0.91	0.94

Table 6: Results of the capacity abuse attack. Here m is the number of synthesized inputs and $\frac{m}{n}$ is the ratio of synthesized data to training data.

the training dataset (e.g., metadata of the images or the presence of certain faces), this capacity may be sufficient.

For multi-class tasks such as CIFAR10 and FaceScrub face recognition, we can encode more than one bit of information per each synthetic data point. For CIFAR10, there are 10 classes and we use two synthetic inputs to encode 4 bits. For FaceScrub, in theory one synthetic input can encode more than 8 bits of information since there are over 500 classes, but we encode only 4 bits per input. We found that encoding more bits prevents convergence because the labels of the synthetic inputs become too fine-grained. We evaluate two settings of m. For CIFAR10, we can encode 25 images with m = 49K and 50 with m =98K. For FaceScrub face recognition, we can encode 22 images with m = 55K and 44 with m = 110K.

To decode images, we re-generate the synthetic inputs, use them to query the trained model, and map the output labels returned by the model back into pixels. We measure the MAPE between the original images and decoded approximate 4-bit-pixel images. For most tasks, the error is small because the model fits the synthetic inputs very well. Although the approximate pixels are less precise, the reconstructed images are still recognizable—see the fourth row of Figure 3.

Text encoding and decoding. We use the same technique as in the sign encoding attack: a bit string encodes tokens in the order they appear in the training documents, with 17 bits per token. Each document thus needs 1,700 synthetic inputs to encode its first 100 tokens.

Dataset	f	m	$\frac{m}{n}$	Test Acc $\pm\delta$	Decode Pre	Decode Rec	Decode Sim
News	SVM	11K	1.0	79.31 −1.27	0.94	0.90	0.94
		22K	2.0	78.11 −2.47	0.94	0.91	0.94
	LR	11K	1.0	79.85 −0.28	0.94	0.91	0.94
		22K	2.0	78.95 −1.08	0.94	0.91	0.94
IMDB	SVM	24K	0.95	89.44 −0.69	0.87	0.89	0.94
		36K	1.44	89.25 −0.88	0.49	0.53	0.71
	LR	24K	0.95	89.92 −0.56	0.79	0.82	0.90
		36K	1.44	89.75 −0.83	0.44	0.47	0.67

Table 7: Results of the capacity abuse attack on text datasets using a public auxiliary vocabulary.

20 Newsgroups models have 20 classes and we use the first 16 to encode 4 bits of information. Binary IMDB models can only encode one bit per synthetic input. We evaluate two settings for m. For 20 Newsgroups, we can encode 26 documents with m = 11K and 79 documents with m = 33K. For IMDB, we can encode 14 documents with m = 24K and 44 documents with m = 75K.

With this attack, the decoded documents have high quality (see Table 5). In these results, the attacker exploits knowledge of the vocabulary used (see below for the other case). For 20 Newsgroups, recovery is almost perfect for both SVM and LR. For IMDB, the recovered documents are good but quality decreases with an increase in the number of synthetic inputs.

Test accuracy. For image datasets, the decrease in test accuracy is within 0.5% for the binary classifiers. For LFW, test accuracy even increases marginally. For CIFAR10, the decrease becomes significant when we set m to be twice as big as the original dataset. Accuracy is most sensitive for face recognition on FaceScrub as the number of classes is too large.

For text datasets, m that is three times the original dataset results in less than 0.6% drop in test accuracy on 20 Newsgroups. On IMDB, test accuracy drops less than 0.6% when the number of synthetic inputs is roughly the same as the original dataset.

Using a public auxiliary vocabulary. The synthetic images used for the capacity-abuse are pseudorandomly generated and do not require the attacker to have any prior knowledge about the images in the actual training dataset. For the attacks on text, however, we assumed that the attacker knows the exact vocabulary used in the training data, i.e., the list of words from which all training documents are drawn (see Section 5.2).

We now relax this assumption and assume that the attacker uses an auxiliary vocabulary collected from publicly available corpuses: Brown Corpus,[2] Gutenberg Corpus [43],[3] Rotten Tomatoes [62],[4] and a word list from Tesseract OCR.[5]

Obviously, this public auxiliary vocabulary requires no prior knowledge of the model's actual vocabulary. It contains 67K tokens and needs 18 bits to encode each token. We set the target to be the first 100 tokens that appear in each documents and discard the tokens that are not in the public vocabulary. Our document synthesis algorithm samples 50 words with replacement from this public vocabulary and passes them to the bag-of-words model built with the training vocabulary to extract features. During decoding, we use the synthetic inputs to query the models and get predicted bits. We use each consecutive 18 bits as index into the public vocabulary to reconstruct the target text.

Table 7 shows the results of the attack with this public vocabulary. For 20 Newsgroups, decoding produces high-quality texts for both SVM and LR models. Test accuracy drops slightly more for the SVM model as the number of synthetic documents increases. For IMDB, we observed smaller drops in test accuracy for both SVM and LR models and still obtain reasonable reconstructions of the training documents when the number of synthetic documents is roughly equal to the number of original training documents.

Memorization capacity and model size. To further investigate the relationship between the number of model parameters and the model's capacity for maliciously memorizing "extra" information about its training dataset, we compared CNNs with different number of filters in the last convolution layer: $16, 32, 48, \ldots, 112$. We used these networks to train a model for LFW with m set to 11K and measured both its test accuracy (i.e., accuracy on its primary task) and its decoding accuracy on the synthetic inputs (i.e., accuracy of the malicious task).

Figure 4 shows the results. Test accuracy is similar for smaller and bigger models. However, the encoding capacity of the smaller models, i.e., their test accuracy on the synthetic data, is much lower and thus results in less accurate decoding. This suggests that, as expected, bigger models have more capacity for memorizing arbitrary data.

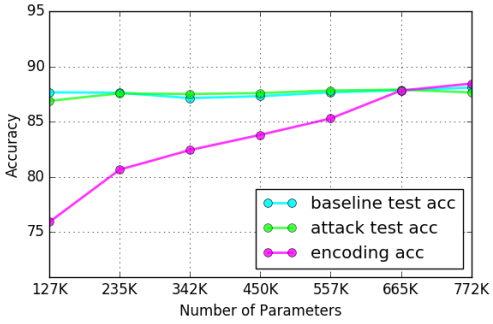

Figure 4: Capacity abuse attack applied to CNNs with a different number of parameters trained on the LFW dataset. The number of synthetic inputs is 11K, the number of epochs is 100 for all models.

Visualization of capacity abuse. Figure 5 visualizes the features learned by a CIFAR10 model that has been trained on its original training images augmented with maliciously generated synthetic images. The points are sampled from the last-layer outputs of Residual Networks on the training and synthetic data and then projected to 2D using t-SNE [53].

The plot clearly shows that the learned features are almost linearly separable across the classes of the training data *and* the classes of the synthetic data. The classes of the training data correspond to the primary task, i.e., different types of objects in the image. The classes of the synthetic data correspond to the malicious task, i.e., given a specific synthetic image, the class encodes a secret about the training images. This demonstrates that the model has learned both its primary task and the malicious task well.

7 COUNTERMEASURES

Detecting that a training algorithm is attempting to memorize sensitive data within the model is not straightforward because, as we show in this paper, there are many techniques and places for encoding this information: directly in the model parameters, by applying a malicious regularizer, or by augmenting the training data with specially crafted inputs. Manual inspection of the code may not detect malicious intent, given that many of these approaches are similar to standard ML techniques.

An interesting way to mitigate the LSB attack is to turn it against itself. The attack relies on the observation that lower bits of model parameters essentially don't matter for model accuracy. Therefore, a client can replace the lower bits of the parameters with random noise. This will destroy any information potentially encoded in these bits without any impact on the model's performance.

Maliciously trained models may exhibit anomalous parameter distributions. Figure 6 compares the distribution of parameters in a conventionally trained model, which has the shape of a zero-mean Gaussian, to maliciously trained models. As expected, parameters generated by the correlated value encoding attack are distributed

[2] http://www.nltk.org/book/ch02.html
[3] https://web.eecs.umich.edu/~lahiri/gutenberg_dataset.html
[4] http://www.cs.cornell.edu/people/pabo/movie-review-data/
[5] https://github.com/tesseract-ocr/langdata/blob/master/eng/eng.wordlist

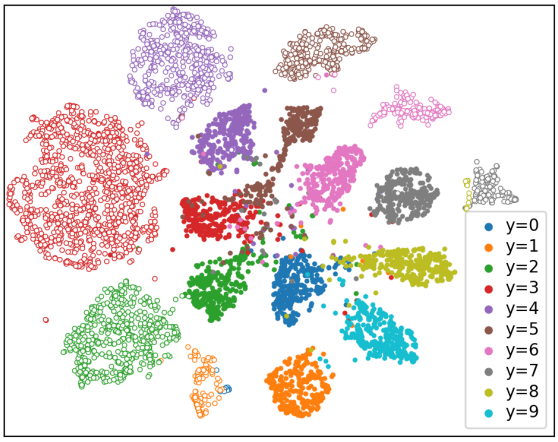

Figure 5: Visualization of the learned features of a CIFAR10 model maliciously trained with our capacity-abuse method. Solid points are from the original training data, hollow points are from the synthetic data. The color indicates the point's class.

very differently. Parameters generated by the sign encoding attack are more centered at zero, which is similar to the effect of conventional l_1-norm regularization (which encourages sparsity in the parameters). To detect these anomalies, the data owner must have a prior understanding of what a "normal" parameter distribution looks like. This suggests that deploying this kind of anomaly detection may be challenging.

Parameters generated by the capacity-abuse attack are not visibly different. This is expected because training works exactly as before, only the dataset is augmented with additional inputs.

8 RELATED WORK

Privacy threats in ML. No prior work considered malicious learning algorithms aiming to create a model that leaks information about the training dataset.

Ateniese et al. [4] show how an attacker can use access to an ML model to infer a predicate of the training data, e.g., whether a voice recognition system was trained only with Indian English speakers.

Fredrikson et al. [26] explore model inversion: given a model f_θ that makes a prediction y given some hidden feature vector $x_1, \ldots, x_n$, they use the ground-truth label $\tilde{y}$ and a subset of $x_1, \ldots, x_n$ to infer the remaining, unknown features. Model inversion operates in the same manner whether the feature vector $x_1, \ldots, x_n$ is in the training dataset or not, but empirically performs better for training set points due to overfitting. Subsequent model inversion attacks [25] show how, given access to a face recognition model, to construct a representative of a certain output class (a recognizable face when each class corresponds to a single person).

In contrast to the above techniques, our objective is to extract specific inputs that belong to the training dataset which was used to create the model.

Homer et al. [32] developed a technique for determining, given published summary statistics about a genome-wide association study, whether a specific known genome was used in the study. This is known as the *membership inference* problem. Subsequent work extended this work to published noisy statistics [24] and MicroRNA-based studies [5].

Membership inference attacks against supervised ML models were studied by Shokri et al. [68]. They use black-box access to a model f_θ to determine whether a given labeled feature vector (x, y) was a member of the training set used to produce θ. Their attacks work best when f_θ has low generalizability, i.e., if the accuracy for the training inputs is much better than for inputs from outside the training dataset.

By contrast, we study how a malicious training algorithm can *intentionally* create a model that leaks information about its training dataset. The difference between membership inference and our problem is akin to the difference between side channels and covert channels. Our threat model is more generous to the adversary, thus our attacks extract substantially more information about the training data than any prior work. Another important difference is we aim to create models that generalize well yet leak information.

Evasion and poisoning. Evasion attacks seek to craft inputs that will be misclassified by a ML model. They were first explored in the context of spam detection [28, 50, 51]. More recent work investigated evasion in other settings such as computer vision—see a survey by Papernot et al. [63]. Our work focuses on the confidentiality of training data rather than evasion, but future work may investigate how malicious ML providers can intentionally create models that facilitate evasion.

Poisoning attacks [9, 18, 38, 57, 65] insert malicious data points into the training dataset to make the resulting model easier to evade. This technique is similar in spirit to the malicious data augmentation in our capacity-abuse attack (Section 5). Our goal is not evasion, however, but forcing the model to leak its training data.

Secure ML environments. Starting with [49], there has been much research on using secure multi-party computation to enable several parties to create a joint model on their separate datasets, e.g. [11, 16, 22]. A protocol for distributed, privacy-preserving deep learning was proposed in [67]. Abadi et al. [1] describe how to train differentially private deep learning models. Systems using trusted hardware such as SGX protect training data while training on an untrusted service [21, 61, 66]. In all of these works, the training algorithm is public and agreed upon, and our attacks would work only if users are tricked into using a malicious algorithm.

CQSTR [74] explicitly targets situations in which the training algorithm may not be entirely trustworthy. Our results show that in such settings a malicious training algorithm can covertly exfiltrate significant amounts of data, even if the output is constrained to be an accurate and usable model.

Privacy-preserving classification protocols seek to prevent disclosure of the user's input features to the model owner as well as disclosure of the model to the user [12]. Using such a system would prevent our white-box attacks, but not black-box attacks.

ML model capacity and compression. Our capacity-abuse attack takes advantage of the fact that many models (especially deep neural networks) have huge memorization capacity. Zhang et al. [75] showed that modern ML models can achieve (near) 100% training accuracy on datasets with randomized labels or even randomized

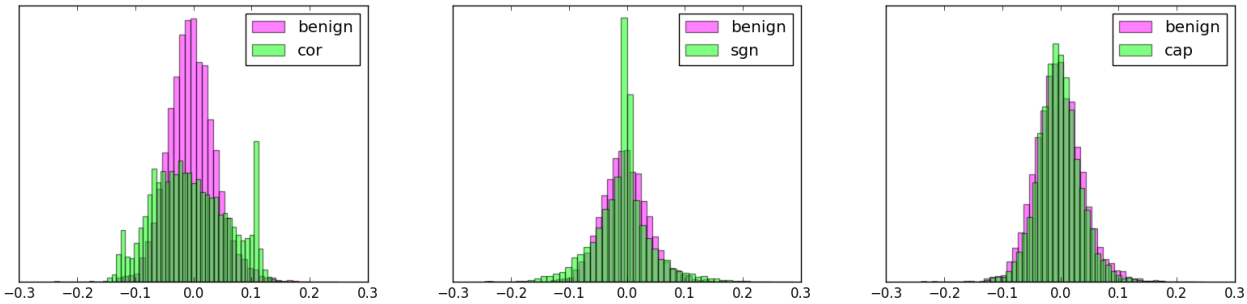

Figure 6: Comparison of parameter distribution between a benign model and malicious models. Left is the correlation encoding attack (cor); middle is the sign encoding attack (sgn); right is the capacity abuse attack (cap). The models are residual networks trained on CIFAR10. Plots show the distribution of parameters in the 20th layer.

features. They argue that this undermines previous interpretations of generalization bounds based on training accuracy.

Our capacity-abuse attack augments the training data with (essentially) randomized data and relies on the resulting low training error to extract information from the model. Crucially, we do this while simultaneously training the model to achieve good testing accuracy on its primary, non-adversarial task.

Our LSB attack directly takes advantages of the large number and unnecessarily high precision of model parameters. Several papers investigated how to compress models [13, 15, 29]. An interesting topic of future work is how to use these techniques as a countermeasure to malicious training algorithms.

9 CONCLUSION

We demonstrated that malicious machine learning (ML) algorithms can create models that satisfy the standard quality metrics of accuracy and generalizability while leaking a significant amount of information about their training datasets, even if the adversary has only black-box access to the model.

ML cannot be applied blindly to sensitive data, especially if the model-training code is provided by another party. Data holders cannot afford to be ignorant of the inner workings of ML systems if they intend to make the resulting models available to other users, directly or indirectly. Whenever they use somebody else's ML system or employ ML as a service (even if the service promises not to observe the operation of its algorithms), they should demand to see the code and understand what it is doing.

In general, we need "the principle of least privilege" for machine learning. ML training frameworks should ensure that the model captures only as much about its training dataset as it needs for its designated task and nothing more. How to formalize this principle, how to develop practical training methods that satisfy it, and how to certify these methods are interesting open topics for future research.

Funding acknowledgments. This research was partially supported by NSF grants 1611770 and 1704527, as well as research awards from Google, Microsoft, and Schmidt Sciences.

REFERENCES

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *CCS*, 2016.
[2] Algorithmia. https://algorithmia.com, 2017.
[3] Amazon Machine Learning. https://aws.amazon.com/machine-learning, 2017.
[4] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *IJSN*, 10(3):137–150, 2015.
[5] M. Backes, P. Berrang, M. Humbert, and P. Manoharan. Membership privacy in MicroRNA-based studies. In *CCS*, 2016.
[6] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, and S. Loureiro. A security analysis of Amazon's Elastic Compute cloud service. In *SAC*, 2012.
[7] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. *TOCS*, 33(3):8, 2015.
[8] A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.
[9] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. In *ICML*, 2012.
[10] BigML. https://bigml.com, 2017.
[11] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson. High-performance secure multi-party computation for data mining applications. *IJIS*, 11(6):403–418, 2012.
[12] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.
[13] C. Bucilă, R. Caruana, and A. Niculescu-Mizil. Model compression. In *KDD*, 2006.
[14] S. Bugiel, S. Nürnberger, T. Pöppelmann, A.-R. Sadeghi, and T. Schneider. AmazonIA: When elasticity snaps back. In *CCS*, 2011.
[15] W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing convolutional neural networks in the frequency domain. In *KDD*, 2016.
[16] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu. Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations Newsletter*, 4(2):28–34, 2002.
[17] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
[18] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *KDD*, 2004.
[19] DeepDetect. https://www.deepdetect.com, 2015–2017.
[20] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. SÃ¿nderby, D. Nouri, et al. Lasagne: First release. http://dx.doi.org/10.5281/zenodo.27878, 2015.
[21] T. T. A. Dinh, P. Saxena, E.-C. Chang, B. C. Ooi, and C. Zhang. M2R: Enabling stronger privacy in MapReduce computation. In *USENIX Security*, 2015.
[22] W. Du, Y. S. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *ICDM*, 2004.
[23] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12(Jul):2121–2159, 2011.
[24] C. Dwork, A. Smith, T. Steinke, J. Ullman, and S. Vadhan. Robust traceability from trace amounts. In *FOCS*, 2015.
[25] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *CCS*, 2015.
[26] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized Warfarin dosing. In *USENIX Security*, 2014.
[27] Google Cloud Prediction API, 2017.
[28] J. Graham-Cumming. How to beat an adaptive spam filter. In *MIT Spam Conference*, 2004.
[29] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.
[30] Haven OnDemand. https://www.havenondemand.com, 2017.
[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[32] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J. V. Pearson, D. A. Stephan, S. F. Nelson, and D. W. Craig. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLOS Genetics*, 2008.

[33] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[34] indico. https://indico.io, 2016.

[35] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML*, 1998.

[36] Keras. https://keras.io, 2015.

[37] Kernel.org Linux repository rooted in hack attack. https://www.theregister.co.uk/2011/08/31/linux_kernel_security_breach/, 2011.

[38] M. Kloft and P. Laskov. Online anomaly detection under adversarial impact. In *AISTATS*, 2010.

[39] H. Krawczyk, R. Canetti, and M. Bellare. HMAC: Keyed-hashing for message authentication. https://tools.ietf.org/html/rfc2104, 1997.

[40] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[42] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Attribute and simile classifiers for face verification. In *ICCV*, 2009.

[43] S. Lahiri. Complexity of word collocation networks: A preliminary structural analysis. In *Proc. Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, 2014.

[44] K. Lang. NewsWeeder: Learning to filter netnews. In *ICML*, 1995.

[45] G. B. H. E. Learned-Miller. Labeled faces in the wild: Updates and new reporting procedures. Technical Report UM-CS-2014-003, University of Massachusetts, Amherst, May 2014.

[46] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[48] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio. Neural networks with few multiplications. In *ICLR*, 2016.

[49] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3), 2002.

[50] D. Lowd. Good word attacks on statistical spam filters. In *CEAS*, 2005.

[51] D. Lowd and C. Meek. Adversarial learning. In *KDD*, 2005.

[52] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proc. 49th Annual Meeting of the ACL: Human Language Technologies*, 2011.

[53] L. v. d. Maaten and G. Hinton. Visualizing data using t-SNE. *JMLR*, 9(Nov):2579–2605, 2008.

[54] Microsoft Azure Machine Learning. https://azure.microsoft.com/en-us/services/machine-learning, 2017.

[55] MLJAR. https://mljar.com, 2016–2017.

[56] MXNET. http://mxnet.io, 2015–2017.

[57] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In *RAID*, 2006.

[58] Nexosis. http://www.nexosis.com, 2017.

[59] H.-W. Ng and S. Winkler. A data-driven approach to cleaning large face datasets. In *ICIP*, 2014.

[60] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[61] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX Security*, 2016.

[62] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proc. ACL*, 2005.

[63] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman. Towards the science of security and privacy in machine learning. https://arxiv.org/abs/1611.03814, 2016.

[64] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *ECCV*, 2016.

[65] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. Tygar. Antidote: Understanding and defending against poisoning of anomaly detectors. In *IMC*, 2009.

[66] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: Trustworthy data analytics in the cloud using SGX. In *S&P*, 2015.

[67] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *CCS*, 2015.

[68] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *S&P*, 2017.

[69] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, 2003.

[70] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. https://arxiv.org/abs/1605.02688, 2016.

[71] S. Torres-Arias, A. K. Ammula, R. Curtmola, and J. Cappos. On omitting commits and committing omissions: Preventing git metadata tampering that (re)introduces software vulnerabilities. In *USENIX Security*, 2016.

[72] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Science & Business Media, 2013.

[73] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning. Managing security of virtual machine images in a cloud environment. In *CCSW*, 2009.

[74] Y. Zhai, L. Yin, J. Chase, T. Ristenpart, and M. Swift. CQSTR: Securing cross-tenant applications with cloud containers. In *SoCC*, 2016.

[75] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.