Mixed-Privacy Forgetting in Deep Networks

Aditya Golatkar[2,1] Alessandro Achille[1] Avinash Ravichandran[1] Marzia Polito[1] Stefano Soatto[1]
[1]Amazon Web Services [2]UCLA
{aachille,ravinash,mpolito,soattos}@amazon.com aditya29@cs.ucla.edu

Abstract

We show that the influence of a subset of the training samples can be removed – or "forgotten" – from the weights of a network trained on large-scale image classification tasks, and we provide strong computable bounds on the amount of remaining information after forgetting. Inspired by real-world applications of forgetting techniques, we introduce a novel notion of forgetting in mixed-privacy setting, where we know that a "core" subset of the training samples does not need to be forgotten. While this variation of the problem is conceptually simple, we show that working in this setting significantly improves the accuracy and guarantees of forgetting methods applied to vision classification tasks. Moreover, our method allows efficient removal of all information contained in non-core data by simply setting to zero a subset of the weights with minimal loss in performance. We achieve these results by replacing a standard deep network with a suitable linear approximation. With opportune changes to the network architecture and training procedure, we show that such linear approximation achieves comparable performance to the original network and that the forgetting problem becomes quadratic and can be solved efficiently even for large models. Unlike previous forgetting methods on deep networks, ours can achieve close to the state-of-the-art accuracy on large scale vision tasks. In particular, we show that our method allows forgetting without having to trade off the model accuracy.

1. Introduction

When building a classification system, one rarely has all the data to be used for training available at the outset. More often, one starts by pre-training a model with some "core" dataset (e.g. ImageNet, or datasets close to the target task) and then incorporates various cohorts of task-specific data as they become available from diverse sources. In some cases, the wrong data may be incorporated inadvertently, or the owners may change their mind and demand that their data be removed. One can, of course, restart the training from scratch every time such a demand is made, but at a significant cost of time and disruption. What if one could remove the effect of cohort(s) of data *a-la-carte*, without re-training, in a way that the resulting model is functionally indistinguishable from one that has never seen the cohort(s) in question, and in addition has no residual information about it buried in the weights of the model? Of course, forgetting can always be trivially achieved by zeroing the weights or replacing them with random noise, but this comes at the expense of the accuracy of the model. Can we forget the cohort of interest without interfering with information about the other data and preserving, to the extent possible, the accuracy of the trained model? Recently, the problem of forgetting has received considerable attention [15, 16, 13, 19, 5, 24, 35, 41, 6, 39, 12, 7, 36], but solutions have focused on simpler machine learning problems such as linear logistic regression. Removing information from the weights of a standard convolutional network still remains an open problem, with some initial results working only on small scale problems [15, 16]. This is mainly due to the highly non-convex loss-landscape of CNNs, which makes the influence of a particular sample on the optimization trajectory and the final weights highly non-trivial to model.

In this paper we introduce Mixed-Linear Forgetting (ML-Forgetting), a method to train large scale computer vision models in such a way that information about a subset of the data can be removed on request – with strong bounds on the amount of remaining information – while at the same time retaining close to the state of the art accuracy on the tasks. To the best of our knowledge, this is the first algorithm to achieve forgetting for deep networks trained on large-scale computer vision problems without compromising the accuracy. To further improve the performance in realistic use-cases, we introduce the notion of forgetting in a *mixed-privacy setting*, that is, when we know that a subset $\mathcal{D}_c \subset \mathcal{D}$ of the training dataset, which we call *core data*, will not need to be forgotten. For example, the core data may be a large dataset of generic data used for pre-training (e.g., ImageNet) or a large freely available collection of task-specific data (e.g., a self-driving dataset) which is not likely subject to changes. We show that ML-Forgetting can naturally take advantage of this setting, to improve both accuracy and bounds on the amount of remaining information after forgetting.

One of the main challenges of forgetting in deep networks is how to estimate the effects of a given training sample on the parameters of the model, which has lead the research to focus on simpler convex learning problem such as linear or logistic regression, for which a theoretical analysis is feasible. To address this problem, Mixed-Linear Forgetting uses a first-order Taylor-series inspired decomposition of the network to learn two sets of weights: a core set $\mathbf{w}_c$ which is trained only with the core data $\mathcal{D}_c$ using a standard (non-convex) algorithm, and a set of linear $\mathbf{w}$ of user weights, which is trained to minimize a quadratic loss function on the changeable user data $\mathcal{D}$. The core weights are learned through standard training (since forgetting is not required on core data), while the user weights are obtained as the solution to a strongly convex quadratic optimization problem. This allows us to remove influence of a subset of the data with strong guarantees. Moreover, by construction, simply setting to zero the user weights removes influence of all changeable data with the lowest possible drop in performance, thus easily allowing the user to remove all of their data at the same time.

To summarize, our key contributions are:

1. We introduce the problem of forgetting (unlearning or data deletion or scrubbing) in a mixed-privacy setting which, compared to previous formalizations, is better taylored to standard practice, and allows for better privacy guarantees.

2. In this setting we propose ML-Forgetting. ML-Forgetting trains a set of non-linear *core* weights and a set of linear *user* weights, which allow it to achieve both good accuracy, thanks to the flexibility of the non-linear weights, and strong privacy guarantees thanks to the linear weights.

3. As a side effect, all the user data may be forgotten completely with the lowest possible drop in performance by simply erasing the user weights.

4. We show that ML-Forgetting can be applied to large-scale vision datasets, and enjoys both strong forgetting guarantees and test time accuracy comparable to standard training of a Deep Neural Network (DNN). To the best of our knowledge, this is the first forgetting algorithm to do so.

5. Furthermore, we show that ML-Forgetting can handle multiple sequential forgetting requests without degrading its performance, which is important for real world applications.

2. Related Work

Forgetting. The problem of machine unlearning is introduced in [8] as an efficient forgetting algorithm for statistical query learning. [32, 13] gives method for forgetting for particular class of learning algorithms, such as k-means clustering. Other methods involve splitting the data into multiple subsets and train models separately on combinations of them [6, 41]. This allows perfect forgetting, but incurs in heavy storage costs as multiple models/gradients need to be stored. In the context of model interpretability and cross-validation, [27, 14] provided a hessian based method for estimating the influence of a training point on the model predictions. [5] proposed a method for hide information about an entire class from the output logits, but does not remove information from the model weights. [19] proposed to remove information from the weights on convex problems using Newton's method, and uses differential privacy [1, 11, 10, 9] to certify data removal. [24] provides a projective residual update method using synthetic data points to delete data points from linear/logistic regression based models. [36] proposed an unlearning mechanism for logistic regression and gaussian processes in a Bayesian setting using variational inference. Recently, [35] proposed a gradient descent based method for data deletion in convex settings, with theoretical guarantees for multiple forgetting requests. They also introduce the notion of statistical indistinguishably of the entire state or just the outputs similar to the information theoretic framework of [16]. We use some of their proof techniques for our theoretical results.

Deep Networks provide additional challenges to forgetting due to their highly non-convex loss functions. [15] proposed an information theoretic procedure to scrub the information from intermediate layers of DNN trained with stochastic gradient descent (SGD), exploiting the stability of SGD [21]. They also bound the amount of remaining information in the weights [3] after scrubbing. [16] extend the framework of [15] to activations. They also show that an approximation of the training process based on a first-order Taylor expansion of the network (NTK theory) can be used to the estimate the weights after forgetting. This approximation works well on small scale vision datasets. However, the approximation accuracy and the computational cost degrade for larger datasets (in particular the cost is quadratic in the number of samples). We also use linearizarion but, crucially, instead of linearly approximating the training dynamics of a non-linear network, we show that we can directly train a linearized network for forgetting. This ensures that our forgetting procedure is correct, and it allows us to scale easily to standard real-world vision datasets.

Linearization. Using a first-order Taylor expansion (linearization) of the network to study its behavior has gained interest recently in the NTK theory [25, 30] as a tool to study the dynamics of DNNs in the limit of infinite filters.

[33] shows that aside from a theoretical tool, it is possible to directly train a (finite) linearized network using an efficient algorithm for the Jacobian-Vector product computation. [2] show that with some changes to the architecture and training process, linearized models can match the performance of non-linear models on many vision tasks, while still maintaining a convex loss function.

3. Preliminaries and Notations

We use the empirical risk minimization (ERM) framework throughout this paper for training. Let $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$ be a dataset where $\mathbf{x}_i \in \mathcal{X}$ denotes the input datum (for example, images) and $\mathbf{y}_i \in \mathcal{Y}$ the corresponding output (for example, one hot vector in classification). Given an input image $\mathbf{x}$, let $f_{\mathbf{w}}(\mathbf{x}) : \mathcal{X} \times \mathbb{R}^d \to \mathcal{Y}$ (for instance, a DNN) be a function parameterized by $\mathbf{w} \in \mathbb{R}^d$ used to model the relation $\mathcal{X} \to \mathcal{Y}$. Given a input-target pair $(\mathbf{x}, \mathbf{y}) \in (\mathcal{X}, \mathcal{Y})$, we denote empirical risk or the training loss for $(\mathbf{x}, \mathbf{y})$ by $\ell(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^d \to \mathbb{R}^+$. We will sometimes abuse notation and use $\ell(\mathbf{w})$ by dropping $\mathbf{x}, \mathbf{y}$. For a training dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n \subset (\mathcal{X}, \mathcal{Y})^n$, we denote the empirical risk/total training loss on $\mathcal{D}$ by $L_{\mathcal{D}}(\mathbf{w}) \triangleq \frac{1}{|\mathcal{D}|}\sum_{i \in \mathcal{D}} \ell(f_{\mathbf{w}}(\mathbf{x}_i), \mathbf{y}_i)$. We will interchangebly use $L_{\mathcal{D}}(f_{\mathbf{w}})$ with $L_{\mathcal{D}}(\mathbf{w})$. Let $\mathcal{A}_\tau(L_{\mathcal{D}}(\mathbf{w}_0)) : (\mathcal{X}, \mathcal{Y})^n \times \mathbb{R}^d \times \mathbb{N} \to \mathbb{R}^d$, denote the weights obtained after τ steps of a training algorithm $\mathcal{A}$ using $\mathbf{w}_0$ as the initialization (for examples, SGD in our case). We denote with $\|\mathbf{w}\|$ the L_2 norm of a vector $\mathbf{w}$ and with $\lambda_{\text{Max}}(Q)$ the largest eigenvalue of a matrix Q. To keep the notation uncluttered, we also use the shorthand $\nabla_{\mathbf{w}} L(\mathbf{w}') \triangleq \nabla_{\mathbf{w}} L(\mathbf{w})|_{\mathbf{w}=\mathbf{w}'}$.

4. The Forgetting Problem

The weights $\mathbf{w}$ of a trained deep network $f_{\mathbf{w}}(\mathbf{x})$ are a (possibly stochastic) function of the training data $\mathcal{D}$. As such, they may retain information about the training samples which an attacker[1] can extract from knowledge of the weights or outputs at inference time. A forgetting procedure is a function $S(\mathbf{w}, \mathcal{D}, \mathcal{D}_F)$[2] (also called scrubbing function) which, given a set of weights $\mathbf{w}$ trained on $\mathcal{D}$ and a subset $\mathcal{D}_f \subset \mathcal{D}$ of images to forget, outputs a new set of weights $\mathbf{w}'$ which are indistinguishable from weights obtained by training without $\mathcal{D}_f$.

Readout functions. The success of the forgetting procedure, can be measured by looking at whether a discriminator function $R(\mathbf{w})$ that can guess – at better than chance probability – whether a set of weights $\mathbf{w}$ was trained with or without $\mathcal{D}_f$ or whether it was trained with $\mathcal{D}_f$ and then scrubbed. Following [15, 16] we call such functions *readout functions*. A popular example of readout function is the confidence of the network (that is, the entropy of the output softmax vector) on the samples in $\mathcal{D}_f$: Since networks tend to be overconfident on their training data [20, 28], a higher than expected confidence may indicate that the network was indeed trained on $\mathcal{D}_f$. We discuss more read-out functions in Section 8.1. Alternatively, we can measure the success of the forgetting procedure by measuring the amount of remaining mutual information $\mathcal{I}(S(\mathbf{w}); \mathcal{D}_f)$[3] between the scrubbed weights $S(\mathbf{w})$ and the data $\mathcal{D}_f$ to be forgotten. While this is more difficult to estimate, it can be shown that $\mathcal{I}(S(\mathbf{w}); \mathcal{D}_f)$ upper-bounds the amount of information that any read-out function can extract [15, 16]. That is, it is an upper-bound on the amount of information that an attacker can extract about $\mathcal{D}_f$ using the scrubbed weights $S(\mathbf{w})$.

Quadratic forgetting. An important example is forgetting in a linear regression problem, which has a quadratic loss function $L_{\mathcal{D}}(\mathbf{w}) = \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}} \|\mathbf{y}_i - \mathbf{w}^T \mathbf{x}_i\|^2$. Given the weights $\mathbf{w} = \mathcal{A}_\tau(L_{\mathcal{D}}(\mathbf{w}))$ obtained after training on $L_{\mathcal{D}}(\mathbf{w})$ using algorithm $\mathcal{A}$, the optimal forgetting function is given by:

$$\mathbf{w} \mapsto \mathbf{w} - H_{\mathcal{D}_r}^{-1} \nabla_{\mathbf{w}} L_{\mathcal{D}_r}(\mathbf{w}), \tag{1}$$

where $H_{\mathcal{D}_r}^{-1}, \nabla_{\mathbf{w}} L_{\mathcal{D}_f}(\mathbf{w})$ is the hessian and gradient of the loss function computed on the remaining data respectively. When $\mathcal{A}_\tau(L_{\mathcal{D}}(\mathbf{w})) = \mathbf{w}^* \triangleq \text{argmin}_{\mathbf{w}} L_{\mathcal{D}}(\mathbf{w})$, we can replace $L_{\mathcal{D}_r}(\mathbf{w}^*)$ with $-L_{\mathcal{D}_f}(\mathbf{w}^*)$ in which case it can be interpreted as a reverse Newton-step that unlearns the data $\mathcal{D}_f$ [19, 15]. Since, as we will see later, the "user weights" of ML-Forgetting minimize a similar quadratic loss function, eq. (1) also describes the optimal forgetting procedure for our model. The main challenge for us will be how to accurately compute the forgetting step since the Hessian matrix can't be computed or stored in memory due to the high-number of parameters of a deep network (Section 6).

Convex forgetting. Unfortunately, for more general machine learning models we do not have a close form expression for the optimal forgetting step. However, it can be shown [27] that eq. (1) is always a first-order approximation of the optimal forgetting. [19] shows that for strongly convex Lipschitz loss functions, the discrepancy between eq. (1) and the optimal forgetting is bounded. Since this discrepancy – even if bounded – can leak information, a possible solution is to add a small amount of noise after forgetting:

$$\mathbf{w} \mapsto \mathbf{w} + H_{\mathcal{D}_r}^{-1}(\mathbf{w}) \nabla_{\mathbf{w}} L_{\mathcal{D}_f}(\mathbf{w}) + \sigma^2 \epsilon, \tag{2}$$

where $\epsilon \sim N(0, I)$ is a vector of random Gaussian noise, which aims to destroy any information that may leak due to

[1]An attacker is any agent intent to extract information about the data used for training.

[2]We will abuse the notation and write $S(\mathbf{w})$ when its arguments are clear from the context.

[3]$\mathcal{I}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{P_{\mathbf{x},\mathbf{y}}}[\log(P_{\mathbf{x},\mathbf{y}}/P_{\mathbf{x}}P_{\mathbf{y}})]$ is the mutual information between $\mathbf{x}$ and $\mathbf{y}$, where $P_{\mathbf{x},\mathbf{y}}$ is the joint distribution and $P_{\mathbf{x}}, P_{\mathbf{y}}$ are the marginal distributions.

small discrepancies. Increasing the variance σ of the noise destroys more information, thus making forgetting more secure, but also reduces the accuracy of the model since the weights are increasingly random. The curve of possible Pareto-optimal trade-offs between accuracy and forgetting can be formalized with the Forgetting Lagrangian [15].

Alternatively, to forget data in a strongly convex problem, one can fine-tune the weights on the remaining data using perturbed projected-GD [35]. Since projected-GD converges to the unique minimum of a strongly convex function regardless of the initial condition (contrary to SGD, which may not converge unless proper learning rate scheduling is used), this is guaranteed to remove all influence of the initial data [35]. The downside is that gradient descent (GD) is impractical for large-scale deep learning applications compared to SGD, projection based algorithms are not popular in practice, and the commonly used loss functions are not generally Lipschitz.

Non-convex forgetting. Due to their highly non-convex loss-landscape, small changes of the training data can cause large changes in the final weights of a deep network. This makes application of eq. (2) challenging. [15] shows that pre-training helps increasing the stability of SGD and derives a similar expression to eq. (2) for DNNs, and also provides a way to upper-bound the amount of remaining information in a DNN. [15] builds on recent results in linear approximation of DNNs, and approximate the training path of a DNN with that of its linear approximation. While this improves the forgetting results, the approximation is still not good enough to remove all the information. Moreover, computing the forgetting step scales quadratically with the number of training samples and classes, which restricts the applicability of the algorithm to smaller datasets.

5. Mixed-Linear Forgetting

Let $f_{\mathbf{w}}(\mathbf{x})$ be the output of a deep network model with weights $\mathbf{w}$ computed on an input image $\mathbf{x}$. For ease of notation, assume that the core dataset and the user dataset share the same output space (for example, the same set of classes, for a classification problem). After training a set of weights $\mathbf{w}_c$ on a core-dataset $\mathcal{D}_c$ we would like to further perturb those weights to fine-tune the network on user data $\mathcal{D}$. We can think of this as solving the two minimization problems:

$$\mathbf{w}_c^* = \arg\min_{\mathbf{w}_c} L_{\mathcal{D}_c}(f_{\mathbf{w}_c}) \tag{3}$$

$$\mathbf{w}_u^* = \arg\min_{\mathbf{w}_u} L_{\mathcal{D}}\big(f_{\mathbf{w}_c^*+\mathbf{w}_u}\big) \tag{4}$$

where we can think of the user weights $\mathbf{w}_u$ a perturbation to the core weights that adapts them to the user task. However, since the deep network $f_{\mathbf{w}}$ is not a linear function in of the weights $\mathbf{w}$, the loss function $L_{\mathcal{D}}\big(f_{\mathbf{w}_c^*+\mathbf{w}_u}\big)$ can be highly non-convex. As discussed in the previous section, this makes forgetting difficult. However, if the perturbation $\mathbf{w}_u$ is small, we can hope for a linear approximation of the DNN around $\mathbf{w}_c^*$ to have a similar performance to fine-tuning the whole network [33], while at the same time granting us easiness of forgetting.

Motivated, by this, we introduce the following model, which we call Mixed-Linear Forgetting model (ML-model):

$$f^{\text{ML}}_{\mathbf{w}_c^*,\mathbf{w}_u}(\mathbf{x}) \triangleq \underbrace{f_{\mathbf{w}_c^*}(x)}_{\text{trained on } \mathcal{D}_c} + \overbrace{\nabla_{\mathbf{w}} f_{\mathbf{w}_c^*}(\mathbf{x}) \cdot \mathbf{w}_u}^{\text{train on } \mathcal{D}}. \tag{5}$$

The model $f^{\text{ML}}_{\mathbf{w}_c^*,\mathbf{w}_u}(\mathbf{x})$ can be seen as first-order Taylor approximation of the effect of fine-tuning the original deep network $f_{\mathbf{w}_c^*+\mathbf{w}_u}(\mathbf{x})$. It has two sets of weights, a set of non-linear core weights $\mathbf{w}_c$, which enters the model through the non-linear network $f_{\mathbf{w}_c}(\mathbf{x})$, and a set linear user-weights $\mathbf{w}_u$ which enters the model linearly. Even though the model is linear in $\mathbf{w}_u$, it is still a highly non-linear function of $\mathbf{x}$ due to the non-linear activations in $\nabla_{\mathbf{w}} f(\mathbf{w}_c^*)$.

We train the model solving two separate minimization problems:

$$\mathbf{w}_c^* = \arg\min_{\mathbf{w}_c} L^{\text{CE}}_{\mathcal{D}_c}(f_{\mathbf{w}_c}), \tag{6}$$

$$\mathbf{w}_u^* = \arg\min_{\mathbf{w}_u} L^{\text{MSE}}_{\mathcal{D}}\big(f^{\text{ML}}_{\mathbf{w}_c^*+\mathbf{w}_u}\big). \tag{7}$$

Eq. (6) is akin to pretraining the weights $\mathbf{w}_c$ on the core dataset $\mathcal{D}_c$, while eq. (7) fine-tunes the linear weights on all the data $\mathcal{D}$. This ensures the weights $\mathbf{w}_c$ will only contain information about the core dataset $\mathcal{D}_c$, while all information about the user data $\mathcal{D}$ is contained in $\mathbf{w}_u$. Also note that we introduce two separate loss functions for the core and user data. To train the user weights we use a mean square error (i.e., L_2) loss [23, 34, 17]:

$$L^{\text{MSE}}_{\mathcal{D}}(\mathbf{w}_u) = \frac{1}{2|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \big\| f_{\mathbf{w}_c^*}(\mathbf{x}) + \nabla_{\mathbf{w}} f_{\mathbf{w}_c^*}(\mathbf{x}) \cdot \mathbf{w}_u - \mathbf{y} \big\|^2 + \frac{\mu}{2}\|\mathbf{w}_u\|^2 \tag{8}$$

where $\mathbf{y}$ is a one-hot encoding of the class label. This loss has the advantage that the weights $\mathbf{w}_u$ are the solution to a quadratic problem, in which case the optimal forgetting step can be written in closed form (see eq. 1). On the other hand, since we do not need to remove any information from the the weights $\mathbf{w}_c$, we can train them using any loss in eq. (3). We pick the standard cross-entropy loss, although this choice is not fundamental for our method.

5.1. Optimizing the Mixed-Linear model

Ideally, we want the ML-model to have a similar accuracy on the user data to a standard non-linear network. At the same time, we want the ML-model to perform significantly better than simply training a linear classifier on top of the last layer features of $f_{\mathbf{w}_c^*}$, which is the trivial baseline method to train a linear model for an object classification task. In Figure 1 (see Section 8 for details) we see that this is indeed the case: while linear, the ML-model is still flexible enough to fit the data with a comparable accuracy to the fully non-linear model (DNN). However, some considerations are in order regarding how to train our ML-model.

Training the core model. Eq. (3) reduces to the standard training of a DNN on the dataset $\mathcal{D}_c$ using cross-entropy loss. We train using SGD with annealing learnig rate. In case $\mathcal{D}_c$ is composed of multiple datasets, for example ImageNet and a second dataset closer to the user task, we first pretrain on ImageNet, then fine-tune on the other dataset.

Training the Mixed-Linear model. Training the linear weights of the Mixed-Linear model in eq. (4) is slighly more involved, since we need to compute the Jacobian-Vector product (JVP) of $\nabla_{\mathbf{w}} f_{\mathbf{w}_c^*}(\mathbf{x}) \cdot \mathbf{w}_u$. While a naïve implementation would require a separate backward pass for each sample, [33, 37] show that the JVP of a batch of samples can be computed easily for deep networks using a slightly modified forward pass. The modified forward pass has only double the computational cost of a standard forward pass, and can be further reduced by linearizing only the final layers of the network. Using the algorithm of [33] to compute the model output, eq. (4) reduces to a standard optimization, which we perform again with SGD with annealing learning rate. Note that, since the problem is quadratic, we could use more powerful quasi-Netwon methods to optimize, however we avoid that to keep the analysis simpler, since optimization speed is not the focus of this paper.

Architecture changes. We observe that a straightforward application of [33] to a standard pre-trained ResNet-50 tend to under-perform in our setting (fine-tuning on large scale vision tasks). In particular, it achieves only slightly better performance than training a linear classifier on top of the last layer features. Following the suggetsion of [2], we replace the ReLUs with Leaky ReLUs, since it boosts the accuracy of linearized models.

6. Forgetting procedure

The user weights $\mathbf{w}_u$ are obtained by minimizing the quadratic loss function $L_{\mathcal{D}}^{\text{MSE}}$ in eq. (8) on the user data $\mathcal{D}$. Let $\mathcal{D}_f \subset \mathcal{D}$ denote a subset of samples we want to forget (by hypothesis $\mathcal{D}_c \cap \mathcal{D}_f = \emptyset$, i.e., the core data is not going to change) and let $\mathcal{D}_r = \mathcal{D} - \mathcal{D}_f$ denote the remaining data. As discussed in Section 4, in case of quadratic training loss the optimal forgetting step to delete $\mathcal{D}_f$ is given by:

$$\mathbf{w}_u \mapsto \mathbf{w}_u - H_{\mathcal{D}_r}^{-1}(\mathbf{w}_c) g_{\mathcal{D}_r}(\mathbf{w}_u), \tag{9}$$

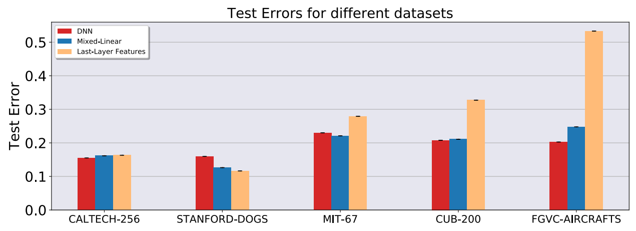

Figure 1. **Mixed-Linear model has comparable accuracy to standard DNN.** Plot of the test errors for different datasets using different models. We take a ResNet-50 pretrained on ImageNet and fine-tune it using different procedures, **(DNN)** We fine-tune the whole network on various datasets, **(Mixed-Linear)** We fine-tune the linearized ResNet-50 (eq. (5)) in a mixed private setting, **Last-Layer Features**: We simply fine-tune the final fully connected (FC) layer of the ResNet-50. We show that fine-tuning a linearized DNN using the mixed-privacy framework performs comparable to fine-tuning a DNN and outperforms simply fine-tuning the last FC layer.

where we define $g_{\mathcal{D}_r}(\mathbf{w}_u) \triangleq \nabla_{\mathbf{w}} L_{\mathcal{D}_r}(\mathbf{w}_u)$ and we can explicitly write the Hessian $H_{\mathcal{D}_r}(\mathbf{w}_c)$ of the loss eq. (8) as:

$$H_{\mathcal{D}_r}(\mathbf{w}_c) = \sum_{\mathbf{x} \in \mathcal{D}_r} \nabla_{\mathbf{w}} f_{\mathbf{w}_c}(\mathbf{x})^T \nabla_{\mathbf{w}} f_{\mathbf{w}_c}(\mathbf{x}) + \mu I. \tag{10}$$

where I is the identity matrix of size d. Thus, forgetting amounts to computing the update step eq. (9). Unfortunately, even if we can easily write the hessian $D_{\mathcal{D}_r}$ in closed form, we cannot store it in memory and much less invert it. Instead, we now discuss how to find an approximation of the forgetting step $H_{\mathcal{D}_r}^{-1}(\mathbf{w}_c) g_{\mathcal{D}_r}(\mathbf{w}_u)$ by solving an optimization problem which does not require constructing or inverting the hessian.

Since $H_{\mathcal{D}_r}(\mathbf{w}_c)$ is positive definite, we can define the auxiliary loss function

$$\hat{L}_{\mathcal{D}_r}(\mathbf{v}) = \frac{1}{2} \mathbf{v}^T H_{\mathcal{D}_r}(\mathbf{w}_c) \mathbf{v} - g_{\mathcal{D}_r}(\mathbf{w}_u)^T \mathbf{v} \tag{11}$$

By setting the gradient to zero, it is easy to see that the forgetting update $H_{\mathcal{D}_r}^{-1}(\mathbf{w}_c) g_{\mathcal{D}_r}(\mathbf{w}_u)$ is the unique minimizer of $\hat{L}_{\mathcal{D}_r}(\mathbf{v})$, so we can recast computing the forgetting update as simply minimizing the loss $\hat{L}_{\mathcal{D}_r}(\mathbf{v})$ using SGD. In general, the product $\mathbf{v}^T H_{\mathcal{D}_r}(\mathbf{w}_c)\mathbf{v}$ of eq. (11) can be computed efficiently without constructing the Hessian using the Hessian-Vector product algorithm [27]. However, in our case we have a better alternative due to the fact that we use MSE loss and that ML-model is linear in weight-space: Using eq. (10), we have that

$$\mathbf{v}^T H_{\mathcal{D}_r}(\mathbf{w}_c)\mathbf{v} = \sum_{\mathbf{x} \in \mathcal{D}_r} \|\nabla_{\mathbf{w}} f_{\mathbf{w}_c^*}(\mathbf{x})\mathbf{v}\|^2 + \mu \|\mathbf{v}\|^2, \tag{12}$$

where $\nabla_{\mathbf{w}} f_{\mathbf{w}_c^*}(\mathbf{x})\mathbf{v}$ is a Jacobian-Vector product which can be computed efficiently (see Section 5.1). Using this result, we compute the (approximate) minimizer of eq. (11) using SGD. When optimizing eq. (11), we compute $g_{\mathcal{D}_r}(\mathbf{w}_u)$ exactly on $\mathcal{D}_r$ and approximate eq. (10) by Monte-Carlo sampling. In Figure 4, we show this method outperforms full stochastic minimization of eq. (11).

Mixed-Linear Forgetting. Let $\Delta\mathbf{w}_u \triangleq \mathcal{A}_\tau(\hat{L}_{\mathcal{D}_r})$ be the approximate minimizer of eq. (11) obtained by training with SGD for τ iterations. Our Mixed-Linear (ML) forgetting procedure $S(\mathbf{w})$ for the ML-model in eq. (5) is:

$$\boxed{\mathbf{w}_u \mapsto \mathbf{w}_u - \Delta\mathbf{w}_u + \sigma^2\epsilon} \tag{13}$$

where $\epsilon \sim N(0, I)$ is a random noise vector [15, 16]. As mentioned in Section 4, we need to add noise to the weights since $\Delta\mathbf{w}_u$ is only an approximation of the optimal forgetting step, and the small difference may still contain information about the original data. By adding noise, we destroy the remaining information. Larger values of σ ensure better forgetting, but can reduce the performance of the model. In the next sections, we analyze theoretically and practically the role of σ.

Sequential forgetting. In practical applications, we may receive several separate requests to forget the data in a sequential fashion. In such cases, we simply apply the forgetting procedure in eq. (13) on the weights obtained at the end of the previous step. A key component is to ensure that the performance of the system does not deteriorate too much after many sequential requests, which we do next.

7. Bounds on Remaining Information

We now derive bounds on the amount of information that an attacker can extract from the weights of the model after applying the scrubbing procedure eq. (13). This will also guide us in selecting the optimal σ and the number of iterations τ to approximate the forgetting step that are necessary to reach a given privacy level (see fig. 2). Let $Y_{\mathcal{D}_F}$ denote some attribute of interest regarding $\mathcal{D}_f$ an attacker might want to access, then from Proposition 1 in [15] we have:

$$\underbrace{\mathcal{I}(Y_{\mathcal{D}_f}, S(\mathbf{w}))}_{\text{Recovered Information}} \leq \underbrace{\mathcal{I}(\mathcal{D}_f, S(\mathbf{w}))}_{\text{Remaining Information in Weights}}$$

where $S(\mathbf{w}) = S(\mathbf{w}, \mathcal{D}, \mathcal{D}_f)$ is the scrubbing/forgetting method which given weights $\mathbf{w}$ trained on $\mathcal{D}$ removes information about $\mathcal{D}_f$ (which in our case is given by eq. 13). Hence, bounding the amount of information about $\mathcal{D}_f$ that remains in the weights $S(\mathbf{w})$ after forgetting uniformly bounds all the information that an attacker can extract.

We now upper-bound the remaining information $\mathcal{I}(\mathcal{D}_f, S(\mathbf{w}))$ after applying the forgetting procedure in eq. (13) to our ML-model, over multiple forgetting requests. Let $\cup_{k=1}^{K}{\mathcal{D}_f}^k$ be the total data asked to be forgotten at the end of K forgetting requests and let $\mathbf{w}_u^K$ be the weights obtained using the forgetting procedure in eq. (13) sequentially. Then we seek to provide a bound on the mutual information between the two, i.e., $\mathcal{I}(\cup_{k=1}^{K}\mathcal{D}_f^k) \triangleq \mathcal{I}(\cup_{k=1}^{K}\mathcal{D}_f^k, \mathbf{w}_u^K)$. We prove the following theorem.

Theorem 1 (Informal)**.** *Let $\Delta\mathbf{w}_u = \mathcal{A}_\tau(\hat{L}_{\mathcal{D}_r})$ be the approximate update step obtained minimizing $\hat{L}_{\mathcal{D}_r}$ (eq. 13) using τ steps of SGD with mini-batch size B. Let $\gamma = 1-\mu^2/\beta^2$, where β is the smoothness constant of the loss in eq. (7). Consider a sequence of K equally sized forgetting requests $\{\mathcal{D}_f^1, \mathcal{D}_f^2, \dots, \mathcal{D}_f^K\}$ and let $\mathbf{w}_u^K$ be the weights obtained after the K requests using eq. (13). Then we have the following bound on the amount of information remaining in the weights $\mathbf{w}_u^K$ about $\cup_{k=1}^{K}\mathcal{D}_f^k$*

$$\mathcal{I}(\cup_{k=1}^{K}\mathcal{D}_f^k) \leq \frac{\overbrace{\gamma^\tau}^{\text{forgetting steps}} c_0\Big(\overbrace{c_1\frac{r^2}{\sigma^2}+}^{\text{ratio to forget}} \overbrace{d}^{\text{num. params}}\Big) + \overbrace{\frac{c_2}{B\sigma^2}}^{\text{batch size}}}{1-(1+\alpha)\gamma^\tau}. \tag{14}$$

where $c_0, c_1, c_2 > 0$, $r = |\mathcal{D}_f^k|/|\mathcal{D}|$ and $0 < \alpha < 1/\gamma^\tau - 1$, $d = \dim(\mathbf{w})$ and $\gamma < 1$.

In [35] a similar probabilistic bound is given on the distance of the scrubbed weights from the optimal weights for strongly convex Lipschitz loss functions trained using projected GD. We prove our bound for the more general case of a convex loss function with L_2 regularization trained using SGD (instead of GD) and also bound the remaining information in the weights.

Role of σ. We make some observations regarding eq. (14). First, increasing the variance σ^2 of the noise added to the weights after the forgetting step further reduces the possible leakage of information from an imperfect approximation. Of course, the downside is that increasing the noise may reduce the performance of the model (see Figure 2 (top) for the trade-off between the two).

Forgetting with more iterations. Running the algorithm $\mathcal{A}$ for an increasing number of steps τ improves the accuracy of the forgetting step, and hence reduces the amount of remaining information. We confirm this empirically in Figure 2 (bottom). Note however that there is a diminishing return. This is due to the variance of the stochastic optimization overshadowing gains in accuracy from longer optimization (see the additive term depending on the batch size). Increasing the batch-size, B in eq. (13) reduces the variance of the estimation and leads to better convergence.

Fraction of data to forget. Finally, forgetting a smaller fraction $r = |\mathcal{D}_f^k|/|\mathcal{D}|$ of the data is easier. On the other hand, increasing the number of parameters d of the model may make the forgetting more difficult.

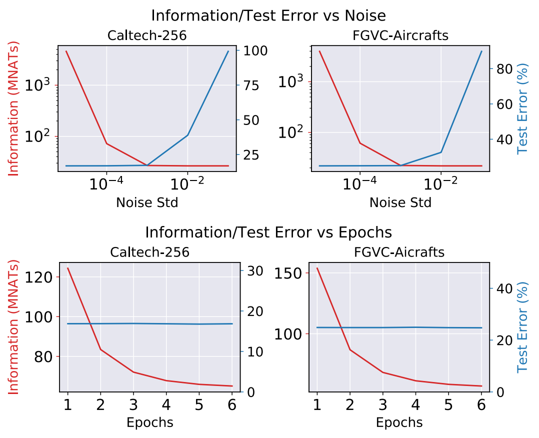

Figure 2. **Forgetting-Accuracy Trade-off** Plots of the amount of remaining information in the weights about the data to forget (red, left axis) and test error (blue, right axis) as a function of the (top) scrubbing noise and (bottom) number of optimization iterations used to compute the scrubbed weights in eq. (13). We aim to forget 10% of the training data through 10 forgetting requests on the Caltech-256 (left) and Aircrafts datasets (right). Note that the remaining information in the weights decreases with an increase in the forgetting noise or the number of epochs during forgetting as predicted by the bound in Theorem 1. Increasing the forgetting noise increases the test error after forgetting (top). In terms of the computational efficiency/speed, doing 2-3 passes over the data (i.e. 2-3 epochs) is sufficient for forgetting (in terms of the test error and the remaining information) rather than re-training from scratch for 50 epochs (bottom) for each forgetting request $\mathcal{D}_f^k$. Thus providing a 16-25× speed-up per forgetting request. We fine-tune the ML-Forgetting model for 50 epochs while training the user weights. Values for τ and σ can be chosen using these trade-off curves given a desired privacy level.

8. Experiments

We use a ResNet-50 [22] as the model $f_{\mathbf{w}}(\mathbf{x})$ in ML-Forgetting. Unless specified otherwise, we forget 10% of randomly chosen training data in all the experiments through 10 sequential forgetting requests each of size 1%. In the appendix, we also provide results for forgetting an entire class and show that our method is invariant to the choice of the subset to be forgotten. More experimental details can be found in the appendix.

Datasets used. We test our method on the following image classification tasks: Caltech-256 [18], MIT-67 [38], Stanford Dogs [26], CUB-200 [40], FGVC Aircrafts [31], CIFAR-10 [29]. Readout function and forgetting-accuracy trade-off plots for MIT-67, StanfordDogs, CUB-200 and CIFAR-10 can be found in the appendix.

8.1. Readout functions

The forgetting procedure should be such that an attacker with access to the scrubbed weights $\mathbf{w}$ should not be able to construct some function $R(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}$, which will leak

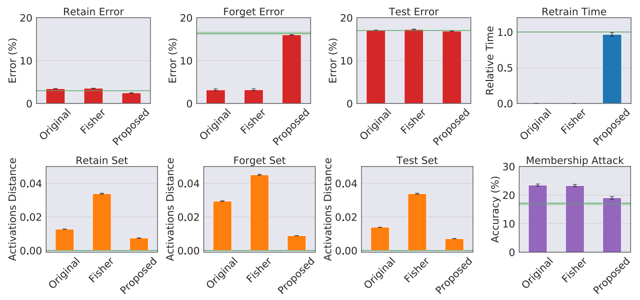

Figure 3. **Read-out functions for different forgetting methods.** We forget a subset of 10% of the training data through 10 equally-size sequential deletion requests using different forgetting methods, and show the value of the several readout functions for the resulting scrubbed models. Ideally, the value of the readout function should be the same as the value (denoted with the green area) obtained on a model retrained from scratch without those samples. Closer to the green area is better. **(Original)** denotes the trivial baseline where we do apply any forgetting procedure. **(Fisher)** Adds Fisher noise as as described in ([15]), **(ML-Forgetting)** The model obtained with our method after forgetting. In all cases, we observe that ML-Forgetting obtains a model that is indistinguishable from one trained from scratch without the data, whereas the other methods fail to do so. This is particularly the case for the *Retrain Time* readout functions, which exploits full knowledge of the weights and it is therefore more difficult to defend against.

information about the set to forget $\mathcal{D}_f$. More precisely the scrubbing procedure should be such that for all $R(\mathbf{w})$:

$$\mathrm{KL}\left(\mathbf{P}(\underbrace{R(S(\mathbf{w},\mathcal{D},\mathcal{D}_f))}_{\substack{\text{readout on weights}\\ \text{after forgetting } \mathcal{D}_f}}|\mathcal{D})\,\|\,\mathbf{P}(\underbrace{R(S_0(\mathbf{w}))}_{\substack{\text{readout on weights}\\ \text{after re-training on } \mathcal{D}_r}}|\mathcal{D}_r)\right) = 0 \tag{15}$$

where $S_0(\mathbf{w})$ is some baseline function that does not depend on $\mathcal{D}_f$ (it only depends on the subset to retain $\mathcal{D}_r = \mathcal{D} - \mathcal{D}_f$). Here $\mathbf{P}(\mathbf{w}|\mathcal{D})$, $\mathbf{P}(\mathbf{w}|\mathcal{D}_r)$ corresponds to the distribution of weights (due to the stochastic training algorithm) obtained after minimizing the empirical risk on $\mathcal{D}$, $\mathcal{D}_r$ respectively. $S(\mathbf{w},\mathcal{D},\mathcal{D}_f)$ corresponds to the scrubbing update defined in eq. (13). We choose $S_0(\mathbf{w}) = \mathbf{w} + z$, where $z \sim \mathcal{N}(0,\sigma^2 I)$. For an ideal forgetting procedure, the value of the readout functions (or evaluation metrics) should be same for a model obtained after forgetting $\mathcal{D}_f$ and re-trained from scratch without using $\mathcal{D}_f$. Some common choice of readout functions include (see Figure 3):

1. **Error on $\mathcal{D}_r, \mathcal{D}_f, \mathcal{D}_{\text{Test}}$:** The scrubbed and the re-trained model (from scratch on $\mathcal{D}_r$) should have similar accuracy on all the three subsets of the data

2. **Re-learn Time:** We fine-tune the scrubbed (model after forgetting) and re-trained model for a few iterations on a subset of the training data (which includes $\mathcal{D}_f$) and compute the number of iterations it takes for the models to re-learn $\mathcal{D}_f$. An ideal forgetting procedure should be such that the re-learn time should be comparable to the re-trained model

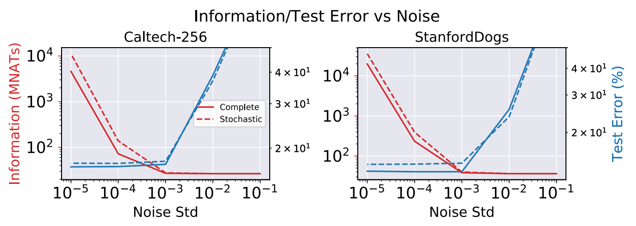

Figure 4. **Comparison of complete and stochastic residual gradient estimation for forgetting.** ML-Forgetting uses the complete residual gradient for the forgetting step, exploiting the fact that the loss function is quadratic. However, one can also estimate it stochastically, which is equivalent to fine-tuning on the remaining data to forget. Here we show that indeed both methods work but – when using the same number of steps – complete estimate gives a better solution due to smaller variance and faster convergence (lower test error and information leakage).

(we plot the relative re-train time in Figure 3). Re-learn time serves a proxy for the amount of information remaining in the weights about $\mathcal{D}_f$ (see Figure 3).

3. **Activation Distance:** We compute the distance between the final activations of the scrubbed weights and the re-trained model ($\mathbf{w}_{\mathcal{D}_r}$) on different subsets of data. More precisely we compute the following: $\mathbb{E}_{\mathbf{x}\in\mathcal{D}'}[\|\operatorname{softmax}(f_{\mathbf{w}}(\mathbf{x})) - \operatorname{softmax}(f_{\mathbf{w}_{\mathcal{D}_r}}(\mathbf{x}))\|_1]$, where $\mathcal{D}' = \mathcal{D}_r, \mathcal{D}_f, \mathcal{D}_{\text{Test}}$. We compare different $\mathbf{w}$ corresponding to the original weights without any forgetting, weights after adding Fisher noise and ML-forgetting (see Figure 3). This serves as a proxy for the amount of information remaining in the activations about $\mathcal{D}_f$.

4. **Membership Attack:** We construct a simple yet effective membership attack similar to [16] using the entropy of the model output. Ideally, a forgetting procedure should have the same attack success as a re-trained model (which is what we observe, see Figure 3).

8.2. Complete vs Stochastic residual gradient

In eq. (13) we compute the residual gradient $g_{\mathcal{D}_r}(\mathbf{w}_u)$ completely once over the remaining data instead of estimating that term stochastically using $\mathcal{A}$. In Figure 4, we compare both the methods of computing the residual gradient. We show that in the ideal region of noise (i.e. $\sigma \in [10^{-5}, 10^{-3}]$), both the remaining information and test error after forgetting (10% of the data through 10 requests) is lower when computing the residual gradient completely.

8.3. Effect of choosing different core datasets

For fine-grained datasets like FGVC-Aircrafts and CUB-200, we show that if the core data has some information about the user task, then it improves forgetting significantly both in terms of the remaining information and the test accuracy. In Figure 5, we show that using ImageNet + 30% of the Aircrafts (we assume that we are not asked to forget this 30% of the data) as core data and 100% of the Aircrafts as the user data, performs much better than simply using ImageNet as core. In Figure 5 (right), we also show that increasing the percentage of user distribution in the core data improves the test accuracy of the Mixed-Linear model.

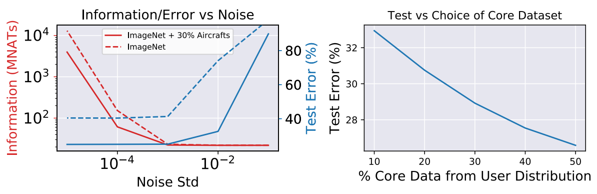

Figure 5. **Effect of using a core data close to the task.** Plot of the remaining information and test error on *Aircrafts* using (a) generic ImageNet core data, and (b) ImageNet pre-training + 30% of the *Aircrafts*. When the core data contain information close to the user task that the network does not need to forget, ML-Forgetting can exploit this to create better core-weights and a correspondingly better linearized model. This improves both the accuracy of the model and makes forgetting easier, as seen from the accuracy-forgetting curves in the plot.

9. Conclusion

We provide a practical forgetting procedure to remove the influence of a subset of the data from a trained image classification model. We achieve this by linearizing the model using a mixed-privacy setting which enables us to split the weights into a set of core and forgettable user weights. When asked to delete all the user data, we can simply discard the user weights. The quadratic nature of the training loss enables us to efficiently forget a subset of the user data without compromising the accuracy of the model. In terms of the time-complexity, we only need 2-3 passes over the dataset per forgetting query for removing information from the weights rather than the 50 re-training epochs, thus, providing a 16$\times$ or more speed-up per request (see Figure 2). We test the forgetting procedure against various read-out functions, and show that it performs comparably to a model re-trained from scratch (the ideal paragon). Finally, we also provide theoretical guarantees on the amount of remaining information in the weights and verify the behavior of the information bounds empirically through extensive evaluation in Figure 2.

Our forgetting procedure heavily relies on the strongly convex nature of the loss landscape which is induced by L_2 regularization (increasing it improves forgetting but compromises accuracy). The quality of forgetting also relies on the subset of data to be forgotten, however, we will leave this for the future work. Even though we provide a forgetting procedure for deep networks by linearizing them without compromising their accuracy, directly removing information from highly non-convex deep networks efficiently still remains an unsolved problem at large.

References

[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*, pages 308–318. ACM, 2016.

[2] Alessandro Achille, Aditya Golatkar, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. Lqf: Linear quadratic fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[3] Alessandro Achille and Stefano Soatto. Where is the Information in a Deep Neural Network? *arXiv e-prints*, page arXiv:1905.12213, May 2019.

[4] Raef Bassily, Mikhail Belkin, and Siyuan Ma. On exponential convergence of sgd in non-convex over-parametrized learning. *arXiv preprint arXiv:1811.02564*, 2018.

[5] Thomas Baumhauer, Pascal Schöttle, and Matthias Zeppelzauer. Machine unlearning: Linear filtration for logit-based classifiers. *arXiv preprint arXiv:2002.02730*, 2020.

[6] Lucas Bourtoule, Varun Chandrasekaran, Christopher Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. *arXiv preprint arXiv:1912.03817*, 2019.

[7] Jonathan Brophy and Daniel Lowd. Dart: Data addition and removal trees. *arXiv preprint arXiv:2009.05567*, 2020.

[8] Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, pages 463–480. IEEE, 2015.

[9] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. *Advances in neural information processing systems*, 21:289–296, 2008.

[10] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(Mar):1069–1109, 2011.

[11] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[12] Sanjam Garg, Shafi Goldwasser, and Prashant Nalini Vasudevan. Formalizing data deletion in the context of the right to be forgotten. *arXiv preprint arXiv:2002.10635*, 2020.

[13] Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. Making ai forget you: Data deletion in machine learning. In *Advances in Neural Information Processing Systems*, pages 3513–3526, 2019.

[14] Ryan Giordano, William Stephenson, Runjing Liu, Michael Jordan, and Tamara Broderick. A swiss army infinitesimal jackknife. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1139–1147, 2019.

[15] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312, 2020.

[16] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. In *European Conference on Computer Vision*, pages 383–398. Springer, 2020.

[17] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Interspeech*, volume 13, pages 1756–1760, 2013.

[18] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[19] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3832–3842. PMLR, 13–18 Jul 2020.

[20] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[21] Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1225–1234, New York, New York, USA, 20–22 Jun 2016. PMLR.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Like Hui and Mikhail Belkin. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. In *International Conference on Learning Representations*, 2021.

[24] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. Approximate data deletion from machine learning models. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 2008–2016. PMLR, 13–15 Apr 2021.

[25] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[26] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs.

[27] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[28] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being bayesian, even just a bit, fixes overconfidence in ReLU networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5436–5446. PMLR, 13–18 Jul 2020.

[29] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[30] Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels. *arXiv preprint arXiv:1911.00809*, 2019.

[31] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[32] Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Deletion-robust submodular maximization: Data summarization with "the right to be forgotten". In *International Conference on Machine Learning*, pages 2449–2458, 2017.

[33] Fangzhou Mu, Yingyu Liang, and Yin Li. Gradients as features for deep representation learning. In *International Conference on Learning Representations*, 2020.

[34] Vidya Muthukumar, Adhyyan Narang, Vignesh Subramanian, Mikhail Belkin, Daniel Hsu, and Anant Sahai. Classification vs regression in overparameterized regimes: Does the loss function matter? *arXiv preprint arXiv:2005.08054*, 2020.

[35] Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. In *Algorithmic Learning Theory*, pages 931–962. PMLR, 2021.

[36] Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillet. Variational bayesian unlearning. *Advances in Neural Information Processing Systems*, 33, 2020.

[37] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

[38] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[39] David Marco Sommer, Liwei Song, Sameer Wagh, and Prateek Mittal. Towards probabilistic verification of machine unlearning. *arXiv preprint arXiv:2003.04247*, 2020.

[40] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[41] Yinjun Wu, Edgar Dobriban, and Susan Davidson. Deltagrad: Rapid retraining of machine learning models. In *International Conference on Machine Learning*, pages 10355–10366. PMLR, 2020.