Unrolling SGD: Understanding Factors Influencing Machine Unlearning

Anvith Thudi[†]*, Gabriel Deza[†]*, Varun Chandrasekaran, Nicolas Papernot[†]
University of Toronto and Vector Institute †, University of Wisconsin-Madison

2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P) | 978-1-6654-1614-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/EUROSP53844.2022.00027

Abstract—**Machine unlearning is the process through which a deployed machine learning model is made to forget about some of its training data points. While naively retraining the model from scratch is an option, it is almost always associated with large computational overheads for deep learning models. Thus, several approaches to *approximately unlearn* have been proposed along with corresponding metrics that formalize what it means for a model to forget about a data point. In this work, we first taxonomize approaches and metrics of approximate unlearning. As a result, we identify *verification error*, *i.e.,* the ℓ_2 difference between the weights of an approximately unlearned and a naively retrained model, as an approximate unlearning metric that should be optimized for as it subsumes a large class of other metrics. We theoretically analyze the canonical training algorithm, stochastic gradient descent (SGD), to surface the variables which are relevant to reducing the verification error of approximate unlearning for SGD. From this analysis, we first derive an easy-to-compute proxy for verification error (termed *unlearning error*). The analysis also informs the design of a new training objective penalty that limits the overall change in weights during SGD and as a result facilitates approximate unlearning with lower verification error. We validate our theoretical work through an empirical evaluation on learning with CIFAR-10, CIFAR-100, and IMDB sentiment analysis.**

1. Introduction

The goal of machine unlearning is to provide a mechanism for removing the impact a datapoint in the training set had on the final model. This is motivated by multiple settings where forgetting a datapoint is paramount. For example, a model which has not unlearned may leak some of the private information contained in a point [1], [2]. This is particularly relevant in scenarios where a particular user who owns the data later revokes access to it–a possibility popularized by the *right-to-be-forgotten* in the GDPR [3]. The odds of leakage is exacerbated by the ability of a ML model to memorize parts of its dataset [4], [5].

Machine unlearning was first introduced by Cao *et al.* [6] to unlearn datapoints for simple hypothesis spaces which have known SQ learning algorithms. Machine unlearning has since been extended to deep neural networks (DNNs) [7], [8], [9], [10], [11], [12], [13], and its privacy implications have been studied [14], [15], [16]. There are currently two broad approaches to machine unlearning: retraining and approximate unlearning.

. *Equal Contribution

With retraining, the point to be unlearned is removed from the training set, and a new model is trained from scratch on this updated training set. This approach has the advantage of carrying a strong claim for why the new model was not influenced in any way by the point to be unlearned, as it is not trained on it. This is a non-trivial advantage when unlearning needs to be transparent, *e.g.,* in the context of right-to-be-forgotten requests [3]. Naively retraining from scratch can be made more efficient [7]. Nevertheless it is still an expensive process that requires changes to the training pipeline.

In approximate unlearning, the model owner instead starts from the existing model and seeks to modify its weights so as to obtain a slightly different model which satisfies an unlearning *criterion* (*e.g.,* poor performance on the point to be unlearned, or *similar* weights to a model naively retrained without the point to be unlearned, etc). This can be done for example through a form of gradient ascent [8], which is the opposite of gradient descent performed to train the model normally. Alternatively various approaches suggest hessian-based updates [11], [9]. Approximate unlearning has the advantage of being more computationally efficient than retraining but comes at the expense of a weaker guarantee: the model learnt may not be completely un-influenced by the unlearned datapoint.

Underlying the ambiguity with the various approximate claims to unlearning is simply the variety of such unlearning criterion that individual approaches consider and their associated metrics: each typically has their own metric for measuring unlearning, but it is not clear how to compare claims made with different approaches/metrics. Ideally, there would be one metric which captures most, if not all the intuitive properties of unlearning covered by metrics proposed to this day. Our work tackles this problem by first showing that *verification error*—the ℓ_2 difference in weights between an approximately unlearned model and a naively retrained model—implies a large class of the other metrics. Thus it carries a stronger intuition, and helps unify the pursuits of other work.

Nevertheless, verification error has its own faults. To measure it, one needs to compute the naively retrained model, which begs the question of why not just use the retrained model as the unlearned model? Furthermore, despite discounting randomness associated with training algorithms, training a model is noisy due to numerical instabilities introduced by back-end randomness in floating point computations; comparison with respect to any one retrained model leaves the metric itself noisy. It is also not immediately apparent what one should do then to reduce the verification error. Additionally, for verification error to subsume other metrics, the unlearning method should satisfy specific properties (refer § 4.1).

DOI 10.1109/EuroSP53844.2022.00027

Our work answers these questions. First by expanding the canonical training algorithm, stochastic gradient descent (SGD), with a Taylor series and then further analyzing it, we formalize an unlearning method—*single gradient unlearning*—that only depends on the initial weights (and obtain some other terms that capture approximation error); unlearning is *approximate* (due to the error term) but *inexpensive* as it depends on only the initial weights.

Second, the analysis leads to an approximation of an unlearned model's verification error which we call the *unlearning error*. The main advantage of this approximation compared to verification error is that it does not require computing the retrained model (*i.e.,* the ground truth that one would obtain by retraining from scratch to unlearn a point). This makes unlearning error cheaper to compute, and also alleviates the issue of the retrained model being noisy in the computation of verification error. We empirically show that unlearning error is strongly correlated with verification error in relevant contexts. This allows us to envision directly minimizing unlearning error during training, so as to improve our ability to later unlearn with smaller verification error. This is particularly appealing given that unlearning error is expressed in a form where the variables we need to change during training to decrease unlearning error are apparent.

These properties of unlearning error help us (and future efforts building on our work) design approximate unlearning mechanisms with lower verification error. Specifically, we propose our *standard deviation (SD) loss* which we show effectively decreases this unlearning error (and consequently, verification error). Intuitively, SD loss forces the model to converge with less overall change to the weights. This allows us to unlearn a point from models trained with the SD loss using single gradient unlearning.

To summarize, the main contributions of our work are:

1) A taxonomy of approximate unlearning which concludes with verification error as a metric to study as it subsumes a large class of unlearning criteria.
2) An analysis of SGD which (a) introduces an inexpensive mechanism for unlearning termed single gradient unlearning, and (b) uncovers the variables impacting verification error. This not only yields an easier-to-compute proxy for verification error, but also informs how to train models that are easier to unlearn well.
3) A way of decreasing this unlearning error (and thus in turn verification error) by the use of our SD loss with little impact to performance. We validate our approach empirically for models trained on on CIFAR-10, CIFAR-100, and IMDb sentiment classification.

2. Primer on Deep Learning & Notation

In our work, we focus on supervised learning [17]. We utilize a bold-faced font to denote vectors (and tensors). We consider a dataset D which consists of pairs $\{(\mathbf{x}_i, y_i)_{i\in[n]}\}$ (where $[n] = \{0, 1, \cdots, n-1\}$); $\mathbf{x}$ is a datapoint (*e.g.,* an image) and y is its label. We wish to train a model M, which is a parameterized function of weights $\mathbf{w}$ that we can modify (or learn); mathematically, the model is denoted as the function $M : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ denotes the space of inputs (*i.e.,* $\mathbf{x} \in \mathcal{X}$) and $\mathcal{Y}$ denotes the space of outputs (*i.e.,* $y \in \mathcal{Y}$). Ideally, the model should be denoted $M_{\mathbf{w}}$, but we omit the dependence on $\mathbf{w}$

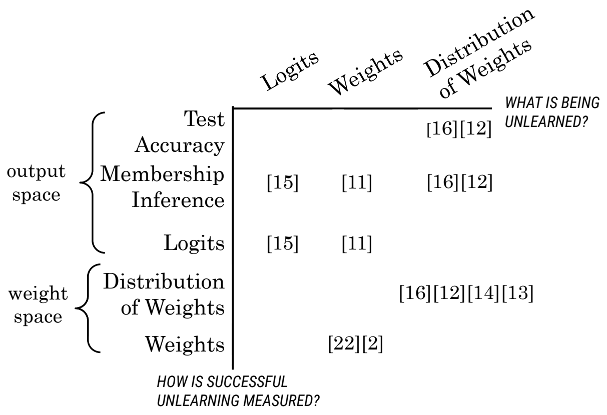


Figure 1. Taxonomy of prior work on post-hoc (post-training) approximate (avoiding retraining) unlearning methods. Unlearning methods are categorized in two ways: (1) What is the definition of unlearning used to motivate the removal of information (horizontal axis)? (2) How is the success of the unlearning method measured (vertical axis)?

when the context is clear. Our experiments consider deep neural networks (DNNs) given their success on various difficult tasks [18], and their large training costs.

To learn the weights that make M best classify D, we minimize a loss function $\mathcal{L}$ that measures the error our model has when predicting the label y from an input $\mathbf{x}$. Examples of such loss functions include the cross-entropy (CE) loss [19] which is the de-facto choice for classification tasks. The cross-entropy loss is defined as:

$$\mathcal{L}_{CE}(M(\mathbf{x}), \mathbf{y}) = -\sum_{i=1}^{c} y_i \log(M(\mathbf{x})_i) \tag{1}$$

where $M(\mathbf{x})$ returns a probability simplex, and $M(\mathbf{x})_i$ is the i^{th} value in this simplex, and $\mathbf{y} = (y_1, \cdots, y_c)$ s.t. $y_i \in \{0, 1\}$ is the one-hot vector defining the label of $\mathbf{x}$, and c is the number of classes. Since a closed form solution cannot be found analytically for non-convex models such as DNNs, the weights $\mathbf{w}$ are learnt in an iterative manner. Many established optimization schemes are derived from mini-batch stochastic gradient descent (SGD) [20]. Formally speaking, mini-batch SGD can be described as below:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_t, \hat{\mathbf{x}}_t}$$

where weights at step t are obtained using the weights from step $t-1$, $\hat{\mathbf{x}}_t$ is the mini-batch of data used at step t, and η denotes the learning rate hyperparameter.

3. Taxonomy of Approximate Unlearning

We begin by discussing the semantics behind unlearning: the problem of *forgetting* a datapoint $\mathbf{x}^* \in D$ from a model M which was trained on it (§ 3.1). We proceed to discuss metrics to measure the efficacy of unlearning (§ 3.2), and methods to achieve them (§ 3.3). Broadly speaking we follow the taxonomy given in Figure 1, with emphasis on specific examples of the metrics (y-axis) and methods (x-axis).

3.1. Defining Unlearning

Let $\mathcal{H}_D$ define the distribution of all models a training rule could return when trained on a dataset D. $\mathcal{H}_D$ is a distribution and not a single point as SGD has inherent stochasticity and there is randomness with back-end floating point operations. Similarly let $\mathcal{H}_{D'}$ represent the distribution of all the models the same training rule returns when trained on dataset $D' = D \backslash \mathbf{x}^*$ where $\mathbf{x}^* \in D$ is the datapoint that is to be unlearned. Lastly, let $U(M, \mathbf{x}^*)$ be some process (randomized or deterministic) that takes a model $M \sim \mathcal{H}_D$ and a data point $\mathbf{x}^*$, and returns another model M''. Now if $\mathcal{S} = U(\mathcal{H}_D, \mathbf{x}^*)$ is the distribution of $\mathcal{H}_D$ after the transformation by U, we say the process U is an *exact unlearning process* iff $\mathcal{S} = \mathcal{H}_{D'}$, *i.e.,* the distribution of output models from $U(\mathcal{H}_D, \mathbf{x}^*)$ is the same distribution as $\mathcal{H}_{D'}$. In this case, $M'' \sim U(M, \mathbf{x}^*)$ is called an unlearned model.

Though this definition looks precise, there is ambiguity in what metric space these models belong to (and consequently for the distributions). A model can be viewed either as just a mapping of inputs to outputs $(\mathbf{x}, M(\mathbf{x}))$ in which case $\mathcal{H}_D$, $\mathcal{S}$, $\mathcal{H}_{D'}$ are distributions over a function space (*i.e.,* continuous function with the supremum metric), or as the specific parameters **w** for an architecture, in which case $\mathcal{H}_D$, $\mathcal{S}$, $\mathcal{H}_{D'}$ are distributions over the weight space (*e.g.,* some finite dimensional real vector space with the euclidean norm). The ambiguity leads to two notions of *exact unlearning*.

Def 1 $d_{\mathbf{w}}(\mathcal{S}, \mathcal{H}_{D'}) = 0$ where $d_{\mathbf{w}}$ measures difference in the *distribution of weights*

Def 2 $d_{out}(\mathcal{S}, \mathcal{H}_{D'}) = 0$ where d_{out} measures difference in the distribution of input-output maps, or simply difference in the *distribution of outputs*

Naively retraining without $\mathbf{x}^*$ as the unlearning process U guarantees both of these definitions (as we exactly obtain $\mathcal{H}_{D'}$), and is the undisputed baseline for exact unlearning. However the issue with naive retraining is the large computational overhead associated with it (*i.e.,* cost to train a model), especially for large models [21].

Approximate unlearning methods try to alleviate these cost-related concerns. Instead of retraining, these methods execute computationally less expensive operations on the final weights [11], [8], [10], apply some architectural change [12], or filter outputs [12]. Approximate unlearning also relaxes the definition of **Def 1** or **Def 2** to requiring the distance between $\mathcal{H}_{D'}$ and $\mathcal{S}$ to be small rather than being exactly zero.

Observe that there can be many different metrics d over both of these spaces. Since comparing distributions is expensive (especially if it involves running the training algorithm multiple times to obtain multiple samples), approximate unlearning methods often move away from requiring a measure of the distance d between distributions to instead a measure of an alternative quantity that captures the difference in the weight space or output space on a point basis. For example, a popular approach is to use membership inference [1] to gauge how close two models (before and after unlearning) are in the output space. The popular classes of metrics used are listed on the y-axis of our taxonomy in Figure 1, and what aspect of the model the approximate unlearning methods change to unlearn a datapoint are listed on the x-axis.

3.2. Unlearning Metrics

We now provide some examples of the various unlearning metrics used in past work. This follows the main categories found on the y-axis of Figure 1. Note the first three are metrics over the weight space (*i.e.,* the metrics take as inputs weights or distributions of weights), and the fourth is in the output space (*i.e.,* metrics take as inputs the outputs of the model).

1. ℓ_2 Distance: Here, we compute the weights of a naively retrained model M' and compare them to the weights of the model M'' that we obtained after using an approximate unlearning process. If they are close, then this suggests that M'' is close to an exactly unlearned model, and can approximate the unlearned model. The standard approach to measure distance in the weight space is the ℓ_2 distance; we term this *verification error* throughout the paper. Such an approach is used by Wu *et al.* [22].

Despite this metric being simple, there are certain drawbacks. First, to compute the verification error, one first needs to compute M' (through naive retraining), which is computationally expensive. If one could obtain M', then they could avoid approximate unlearning altogether and use that as the unlearned model. Another issue is that one can retrain on the same sequence of data from the same initialization and obtain different terminal weights [23] (which is caused by randomness and numerical instabilities in floating point operations).

2. KL Divergence: To bypass this issue of hardware randomness in training, Golatkar *et al.* [9] consider measuring the similarity of the weights of M'' with respect to a distribution of weights of M'. They achieve this by looking at *Kullback-Leiber (KL) divergence* of the two distributions (as they consider M'' as being the result of another stochastic process on the original model M). The drawback to this are that explicitly calculating the KL divergence requires knowing the final distributions of models trained on D'. This in turn involves sampling many final models trained on D' in order to fit some distribution to the known distribution of M'.

3. Privacy Leakage of Weight Distributions: Another common metric for measuring unlearning that also looks at the distribution of weights is the *privacy leakage from the distribution of weights of* M'' (*e.g.,* [24]): ideally, the distribution of weights of M' leaks no information about $\mathbf{x}^*$ as it was not trained on it (though this is not necessarily the case [15]), and so the smaller the privacy leakage of $\mathbf{x}^*$ from the distribution of weights of M'', the closer M'' is to being unlearned. Sekhari *et al.* [10] analogously present privacy leakage for unlearning in the framework of differential privacy (DP): they bound the leakage of information about a particular $\mathbf{x}^*$ (or more generally a set of datapoints to unlearn) with privacy parameters ε, δ and define this as (ε, δ)-unlearning. Guo *et al.* [11] work with a similar setup to Sekhari *et al.*, though not identical as they consider only the ε bound part. In both cases, having (or decreasing) a bound on privacy leakage is presented as unlearning better.

4. Membership Inference: A privacy attack like membership inference (MI) only requires access to the model's predictions (outputs) to determine whether the unlearned model M'' was or was not trained on the point to be unlearned $\mathbf{x}^*$. This approach is used by Graves *et al.* [8] and Baumhauer *et al.* [12]. The reasoning here is that as M' was properly retrained without $\mathbf{x}^*$, a MI attack should return that it was not trained with $\mathbf{x}^*$; if a MI attack on M'' also consistently outputs that M'' was not trained with $\mathbf{x}^*$, then one could argue that their outputs are similar. However, such approaches are not *precise*: it is possible a MI attack will give a false positive that M' had indeed trained on $\mathbf{x}^*$, in which case simply lowering the MI likelihood of $\mathbf{x}^*$ for M'' does not necessarily imply it has unlearned. We revisit the limitations of MI in § 4.3.

3.3. Unlearning Methods

We turn our attention to the following question: *how exactly does one unlearn*? Current approaches are broadly categorized on the x-axis of Figure 1.

Logits: Baumhauer *et al.* [12] consider a specific type of model taking the form $M(\mathbf{x}) = \mathbf{w}f(\mathbf{x})$, where f is some black-box feature extractor, and $\mathbf{w}$ is some parameterized matrix which is what is modified by training. They then unlearn a class from the linear layer *only*. This unlearning is done by defining a filtration matrix and appending it to the model which shifts the outputs to what one would get if one did not train with that class.

Weights: Graves *et al.* [8] proposed amnesiac machine learning, which is an unlearning protocol that logs all the training updates. Then, upon receiving an unlearning request for $\mathbf{x}^*$, one proceeds to add back all the training updates that involved $\mathbf{x}^*$ to the final weights to obtain an unlearnt model M''.

Distribution of Weights: The scrubbing procedure introduced by Golatkar *et al.* [9], [13] is derived by adding a weighted term to the loss (to measure the KL divergence of the distribution of final weights), and minimizing the added term. Alternatively, DP provides bounds on how indistinguishable a given model is to one not trained on any one of its training datapoints, in essence stating that a model has already unlearned. However Sekhari *et al.* [10] show that differentially private learning only allows one to delete $O(\frac{n}{\sqrt{d}})$ data points while retaining meaningful bounds, and propose an unlearning algorithm for (strongly) convex loss functions that improves the number of datapoints one can delete by adding a hessian update and noise to the final weights.

4. Verification Error & Other Metrics

The taxonomy in § 3 captures the different unlearning metrics, which in turn represent the different aspects of a model that can change with the training data. Understanding the relationship between these metrics is paramount to concretely understanding the unlearning guarantees they provide. To this end, we focus on understanding what metrics verification error can and can not capture.

4.1. Verification Error implies L^p Weight Metrics

We will prove that verification error provides a bound on the supremum norm (or L^∞ norm[1]) of the difference of the distribution of weights obtained by (a) naively retraining, and by (b) using an approximate unlearning method[2]. Note that a bound on the L^∞ norm implies bounds on all L^p norms for $p \geq 1$ (refer Propostion 6.10 in [25] and by definition, distributions are in L^1). Informally, this allows us to motivate verification error as a metric that bounds a large class of weight space metrics.

Similarly, a bound on the supremum norm also implies guarantees similar to DP (in particular $\varepsilon = 0$ and $\delta = b$ where b is the bound on the supremum norm) by the reverse triangle inequality. The opposite, though, is not true when $\varepsilon \neq 0$ (as the bound now depends on the magnitude of the function and thus is not uniform). However, it should be noted that in our subsequent analysis, we make several assumptions about the stochasticity of training which does not allow us to derive strict DP-like guarantees. In particular, we will assume that noise in the final weights for a fixed training sequence is a function of only the number of steps, and changes negligibly when removing a single step from each epoch. We explain why we made these assumptions later in the section during our derivation. Similarly our analysis will focus on unlearning with batch size 1 for simplicity.

In particular, the focus here will be first comparing the probability distributions of final weights obtained by training m epochs starting at a fixed initial weight $\mathbf{w}_0$ with dataset D and dataset $D' = D \setminus \mathbf{x}^*$ with SGD (and not mini-batch SGD, *i.e.,* considering mini-batch sizes of 1). We will represent the density functions respectively as $\mathbb{P}(\mathbf{w})$ and $\mathbb{P}'(\mathbf{w})$ where both are functions of the weights of the model $\mathbf{w}$. Note that in what follows, $\mathbf{x}^*$ is fixed but can be any datapoint (*i.e.,* is a constant).

Let the size of D be $|D| = n$, and let $I = \{\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \cdots, \mathbf{x}_{i_{mn}}\}$ denote a particular ordering of the datapoints when training on D for m epochs (where $i_j \in [n]$). Note that we have $(n!)^m$ such combinations; for D', we have $(n-1)!^m$ such combinations. For each I, there exists an I' (corresponding to $D' = D \setminus \mathbf{x}^*$), which is formed by removing all instances of $\mathbf{x}^*$ (in-place) from I. Note that for a given I', we have n^m corresponding choices of I (as for each epoch we have n choices to place $\mathbf{x}^*$ per epoch, and m epochs in total). Let $\mathbf{w}_I$ represent the deterministic final weights resulting from training with the data ordering I, starting from $\mathbf{w}_0$. Let $\mathbf{g}$ denote the random variable which represents the noise on the final weights when training for $m \cdot n$ steps. Thus, the random variable representing the final weights when training with I is $\mathbf{z}_I = \mathbf{w}_I + \mathbf{g}$. Similarly, we define $\mathbf{z}_{I'} = \mathbf{w}_{I'} + \mathbf{g}'$ where $\mathbf{g}'$ represents the noise when training $(n-1) \cdot m$ steps. Note that, if training is deterministic, we have $\mathbf{w}_{I'} = \mathbf{w}_I - \mathbf{d}_I$ where $\mathbf{d}_I$ is some constant that only depends on $\mathbf{w}_0$ and the ordering I (as under deterministic operations $\mathbf{w}_{I'}$ and $\mathbf{w}_I$ are constants depending only on $\mathbf{w}_0$ and the ordering I and thus $\mathbf{w}_I - \mathbf{w}_{I'} = \mathbf{d}_I$ is a constant that only depends on them). Lastly, we make an assumption that the noise from training with $m \cdot n$ steps is equivalent to that training from $(n-1) \cdot m$ steps, *i.e.,* that $\mathbf{g} = \mathbf{g}'$. This is based on the fact that this noise should

1. See [25] for more on L^p spaces.

2. In our final result we will require the unlearning method to only depend on the initial weights and batch ordering.

only be a function of the number of training steps, and that noise per step is small (see [23, § 6] for a discussion on noise during training)[3]. Thus we have $\mathbf{z}_{I'} = \mathbf{z}_I - \mathbf{d}_I$.

Given that SGD samples individual datapoints uniformly, the probability of obtaining a given data ordering is constant. Further, if $\mathbb{P}_I(\mathbf{w})$ is the density function corresponding to $\mathbf{z}_I$ and $\mathbb{P}'_{I'}(\mathbf{w})$ is the density function corresponding to $\mathbf{z}_{I'}$ (which is equal to $\mathbb{P}_I(\mathbf{w} - \mathbf{d}_I)$), then $\mathbb{P}(\mathbf{w}) = \frac{1}{n!^m} \sum_I \mathbb{P}_I$ and $\mathbb{P}'(\mathbf{w}) = \frac{1}{(n-1)!^m} \sum_{I'} \mathbb{P}'_{I'}(\mathbf{w})$.

Lemma 1. *If every $\mathbb{P}_I$ is Lipschitz with Lipschitz constant L (which is true if $\mathbf{g}$ is gaussian), and if we let $d = \frac{1}{n!^m} \sum_I ||\mathbf{d}_I||_2$, then:*

$$||\mathbb{P}(\boldsymbol{w}) - \mathbb{P}'(\boldsymbol{w})||_2 \leq L \cdot d \tag{2}$$

We refer the reader to Appendix A for the detailed derivation. This result shows that by accounting for all the individual distributions (for each ordering I), we were able to obtain a Lipschitz condition on the combined distribution (of final weights obtained by training m epochs starting at a fixed intial weight $\mathbf{w}_0$). The Lipschitz condition is expressed with respect to the datasets including or excluding the point to be unlearned.

Now, we are also interested in the difference between the density function obtained after the application of an approximate unlearning method on M (*i.e.,* $\mathbb{P}''(\mathbf{w})$), and $\mathbb{P}'(\mathbf{w})$. If the approximate unlearning method only considers $\mathbf{w}_0$, I to define an unlearning update $\mathbf{u}_I$, and obtains the approximately unlearned weights by adding $\mathbf{u}_I$ to $\mathbf{w}_I$, *i.e.,* $\mathbf{w}''_I = \mathbf{w}_I + \mathbf{u}_I$, then:

Corollary 1. *If every $\mathbb{P}_I$ is lipschitz with constant L (true if $\mathbf{g}$ is gaussian), and if we let $v = \frac{1}{n!^m} \sum_I ||\mathbf{d}_I + \mathbf{u}_I||_2$, then:*

$$||\mathbb{P}''(\boldsymbol{w}) - \mathbb{P}'(\boldsymbol{w})||_2 \leq L \cdot v \tag{3}$$

The detailed proof is in Appendix A. This naturally follows from the same accounting procedure used for the previous lemma, but now $\mathbf{d}_I$ have been swapped for $\mathbf{d}_I + \mathbf{u}_I$. This gives us a bound on the supremum (or L^∞) norm by the average verification error v (times some constant) on the difference of the probability distributions after approximate unlearning $\mathbb{P}''(\mathbf{w})$ and ideal retraining $\mathbb{P}'(\mathbf{w})$. Note, however, that there is a necessary assumption on the form of this approximate unlearning method: to obtain Equation 3, we require the unlearning update to only be dependant on the initial weight $\mathbf{w}_0$ and I. We will introduce such an unlearning approach in § 5. Finally, an analogous bound going in the opposite direction is provided in Appendix A.

4.2. Convergence in the Outputs over Finite Sets

It is intuitive that having similar weights would mean having similar outputs, as clearly if the weights of two models are identical then so are their outputs. We can formalize this by looking specifically at the verification error defined as $v = ||\mathbf{w}'' - \mathbf{w}'||_2$ where $\mathbf{w}''$ are the weights of M'' (the unlearned model) and $\mathbf{w}'$ are the weights of M' (the model unlearned by retraining). With v defined as above, we have for any $\mathbf{x} \in D$ that $\lim_{v\to 0} ||M'(\mathbf{x}) - M''(\mathbf{x})||_2 = 0$ under continuity of the outputs of the models as functions of the weights. Here, by continuity, we mean that fixing the input $\mathbf{x}$ to a model results in the output $M(\mathbf{x})$ being a continuous function of the weights. Thus, $\lim_{v\to 0}$ entails point-wise convergence of the outputs of M'' to the outputs of M'. Furthermore, note that smooth functions are locally lipschitz. Because DNNs are smooth functions, this relation in the limit is thus equivalent to a linear convergence on an upper bound on the ℓ_2 difference between the two models' outputs. Now note, considering only a finite set of inputs $\{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ (*i.e.,* a dataset), we can take the maximum local lipschitz constant of all $M(\mathbf{x}_i)$ as a function of its weights for a fixed input $\mathbf{x}_i$, and hence have linear convergence on the outputs for all points in the set (not just pointwise). This means changes to the differences in weights lead to proportional changes to an upper bound on the ℓ_2 difference in the models' outputs over a dataset.

However the opposite does not hold. Consider the following counterexample: if one permutes the weights of a neural network, the model's outputs remain identical [26]. Yet, the verification error would not be 0 after such a permutation, and so having all the outputs be the same does not entail having the same weights.

4.3. Connection to Membership Inference

We now describe an apparent lack of a consistent relationship between verification error and MI-based metrics such as privacy risk score (or PRS) [27] (which will be further supported by results in § 8.4). Note that PRS was shown to accurately represent the confidence a datapoint was used during training *i.e.,* likelihood of MI. We implemented PRS by taking a shadow model trained only on half the training set, and allow the MI adversary to access half the test set. We then constructed the conditional probability distribution given in Equation 15 of [27], where we estimate the per label conditional probability by dividing the range of the entropy losses (Equation 8 in [27]) for a given label into bins. Once the conditional distribution for training and test are constructed, we evaluate the PRS for a given point by computing Equation 13 of [27]. We now proceed to empirically show how a decrease in PRS *does not* result in a decrease in verification error.

Specifically, we test the correlation between PRS and verification error after applying an approximate unlearning method to a datapoint $\mathbf{x}^*$. We choose amnesiac machine learning [8] because it was introduced with a MI-based criterion.[4] We train the original model M, the naively retrained model M', and model M'' unlearned using amnesiac unlearning on the CIFAR-10 [29] and CIFAR-100 datasets [29], using the ResNet-18 and VGG-19 architectures [30], [31]. We obtain 27 triplets of M,M'',M' by changing the training settings (*i.e.,* different amounts of training, batch sizes, learning rates etc.). For each triplet, we measured the PRS of the unlearned point $\mathbf{x}^*$ on M'',

3. Thus, adding or removing m steps (1 per epoch) when compared to the $m \cdot n$ total steps when training with ordering I is insignificant

4. We reproduced the approach because it was originally evaluated on a MI attack by Yeom *et al.* [28].

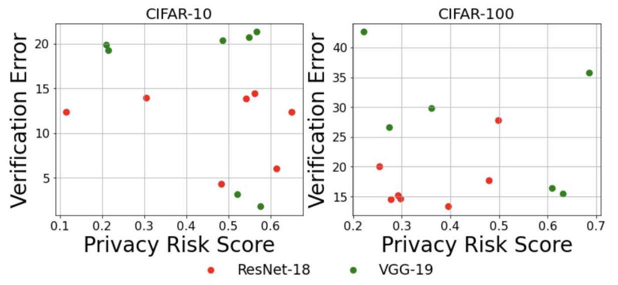


Figure 2. Correlation between privacy risk score and verification error in different training setups for CIFAR-10 and CIFAR-100. The correlations are -0.29 and -0.02, respectively.

and the verification error $||\mathbf{w}'' - \mathbf{w}'||_2$ (obtained from M'' and M' respectively). The results are in Figure 2.

The main takeaway is there is no monotonic relation between PRS and verification error. However it may be that under certain constraints (on the model, loss, training algorithm, etc.) there is a monotonic or even linear relationship between verification error and PRS, and future work may look into studying this.

5. Defining the Unlearning Error

Recall that verification error's strongest limitation is the expense associated with calculating it (*i.e.,* the requirement for naive retraining). To circumvent this issue, we set out to approximate verification error. Our analysis seeks to understand the deterministic impact of a datapoint $\mathbf{x}^*$ on the final weights of a model when trained with SGD starting at initial weights $\mathbf{w}_0$. We expand the recursive updates performed by SGD to isolate terms that can be easily unlearned (because they do not depend on the order of datapoints) from terms that are difficult to unlearn. We then obtain and analyze a closed form approximation of these terms. This results in (a) a proxy metric (which we term *unlearning error*) that captures the effects of the verification error, and (b) an inexpensive approximate unlearning method that only depends on $\mathbf{w}_0$ and so allows us to use the bounds presented in § 4 [5].

5.1. Expanding SGD

To understand the impact of a datapoint $\mathbf{x}^*$ on the final weights, we need to expand a sequence of SGD updates. We begin with the definition of a single SGD learning update:

$$\mathbf{w}_1 = \mathbf{w}_0 - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_0}$$

where $\mathbf{w}_0$ denotes the weights at step 0 and $\hat{\mathbf{x}}_0$ denotes the data sampled at step 0. Note, we make no constraints on what $\mathbf{w}_0$ is (that is where training starts from), *i.e.,* could start from a pre-trained model). We obtain $\mathbf{w}_2$, the weight obtained at step 2, as follows:

$$\mathbf{w}_2 = \mathbf{w}_0 - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_0} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_1, \hat{\mathbf{x}}_1}$$

$$\implies \mathbf{w}_2 = \mathbf{w}_0 - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_0} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0 - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_0}, \hat{\mathbf{x}}_1}$$

5. Do note future methods may expand our theory to consider methods not only dependent on $\mathbf{w}_0$.

This can further be expanded and approximated as follows:

$$\mathbf{w}_2 \approx \mathbf{w}_0 - \eta\left(\frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_0} + \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_1} + \frac{\partial^2 \mathcal{L}}{\partial^2 \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_1}\left(-\eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_0}\right)\right)$$

Our end goal is informed by our analysis in § 4: we *must* focus on an analysis of verification error in SGD that is only a function of the initial weights $\mathbf{w}_0$ and the ordering of batches of data I. To scale the above for a sequence of t updates, notice that the hessian terms from the expansion recursively depend on each other. Thus we have the final approximation:

$$\mathbf{w}_t \approx \mathbf{w}_0 - \eta \sum_{i=0}^{t-1} \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_i} + \sum_{i=1}^{t-1} f(i) \tag{4}$$

where $f(i)$ is defined recursively as:

$$f(i) = -\eta \frac{\partial^2 \mathcal{L}}{\partial^2 \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_i}\left(-\eta \sum_{j=0}^{i-1} \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_j} + \sum_{j=0}^{i-1} f(j)\right) \tag{5}$$

with $f(0) = 0$.

Understanding the approximation. From Equation 4, observe that one can represent the weights obtained after training t updates as the sum of two sums. We now wish to understand how $\mathbf{x}^*$ affects this expression (and thus the approximate outcome of training). The terms in the *first sum* are simply gradients taken with respect to the initial weights $\mathbf{w}_0$ and $\hat{\mathbf{x}}_\mathbf{i}$ following the order we give data to the model. Notice, however, that the exact order does not matter as we are simply adding them. In this case, it is clear that the effect of $\mathbf{x}^*$ (provided at any step in training) on this first sum is a gradient (or gradients) computed with respect to $\mathbf{w}_0$ and $\mathbf{x}^*$.

Single gradient unlearning: To reverse this effect and unlearn, we simply have to *add back these gradients* which amounts to adding $\frac{\eta m}{b} \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \mathbf{x}^*}$ to the final weights, where η is the learning rate, b is the batch size, and m is the number of epochs (which is the number of copies of this gradient that are present in the first sum).

The *second sum* is more complex as the expression is recursive, and so there is an inherent dependence on order. Understanding this sum will be our focus in § 5.2. One of the main takeaways is that to compute all the terms that contain a particular $\mathbf{x}^*$, one would have to spend at least as much compute as training. For this reason, *the process to unlearn the first sum will be our unlearning method*, and our analysis of the second sum (which we do not unlearn) will result in an *inexpensive proxy metric* for the verification error which makes clear key variables one should focus on to decrease verification error.

5.2. Approximating the Second Sum Series

The global structure of the second sum follows $\eta^2 \mathbf{c}_t + \eta^3 \mathbf{c}_{t-1} + \ldots + \eta^{2+t-1} \mathbf{c}_1$, where $\mathbf{c}_i$ represents some vector. Thus, we can focus primarily on the η^2 dependent term as, in practice, we often observe η being a magnitude or more less than 1, and so η^2 is the dominant term. Thus, the second sum can be approximated as:

$$\eta^2 \mathbf{c}_t = \sum_{i=1}^{t-1} \eta^2 \frac{\partial^2 \mathcal{L}}{\partial^2 \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_i} \sum_{j=0}^{i-1} \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_j} \quad (6)$$

We can further approximate $\sum_{j=0}^{i-1} \frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_j}$ by its expectation $\frac{i(\mathbf{w}_t - \mathbf{w}_0)}{t}$. We further normalize the vector, and thus have:

$$\eta^2 \mathbf{c}_t \approx \eta^2 \frac{||\mathbf{w}_t - \mathbf{w}_0||_2}{t} \sum_{i=1}^{t-1} \frac{\partial^2 \mathcal{L}}{\partial^2 \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_i} \frac{\mathbf{w}_t - \mathbf{w}_0}{||\mathbf{w}_t - \mathbf{w}_0||_2} i \quad (7)$$

By focusing on $||\mathbf{c}_t||_2$, note that

$$||\frac{\partial^2 \mathcal{L}}{\partial^2 \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_i} \frac{\mathbf{w}_t - \mathbf{w}_0}{||\mathbf{w}_t - \mathbf{w}_0||_2}||_2 \leq ||\frac{\partial^2 \mathcal{L}}{\partial^2 \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_i}||_2$$

Now note the definition of the induced ℓ_2 norm of the hessian, which is simply the square root of the largest eigenvalue of the hessian *i.e.,* $\sigma_{1,i}$ the first singular value of the hessian on $\hat{\mathbf{x}}_i$. For sake of simplicity we consider $\sigma = \max(\sigma_{1,i})$, and thus have the following inequality:

$$||\eta^2 \mathbf{c}_t||_2 \leq \eta^2 \frac{||\mathbf{w}_t - \mathbf{w}_0||_2}{t} \cdot \sigma \cdot \frac{t(t-1)}{2} \quad (8)$$

From this, we can observe that the second sum linearly depends on $||\mathbf{w}_t - \mathbf{w}_0||_2$, σ, and t. σ captures the non-linearity of the loss landscape from the hessian, and so we see that the more linear the loss landscape, the tighter this bound on the approximation. We also see from this that in the scenario $t = 1$ or $\sigma = 0$, this bound, and thus the term is 0, reinforcing the notion that if the loss landscape was just linear we could focus solely on the first sum of gradients.

Counting terms. Now, with an understanding of what the dominant terms in the second sum of Equation 4 are, we take a step back and ask whether it is possible to unlearn $\mathbf{x}^*$ from these. What we will do is count all the terms involving $\mathbf{x}^*$; we then show that it is computationally expensive to compute those terms in order to remove their influence (and thus we can not reasonably improve our single-gradient unlearning method). Specifically, we focus on the expression in Equation 6. We let i^* denote the first index where $\mathbf{x}^*$ appears, and we are interested in those terms that contain $\hat{\mathbf{x}}_{i^*}$ (and thus $\mathbf{x}^*$). These are the terms that we would need to remove in order to forget $\hat{\mathbf{x}}_{i^*}$ from the dominant terms in our second sum in Equation 4. Note then that this will actually be an undercount, as $\mathbf{x}^*$ could appear in another batch; the final cost we state will be less than what we actually need to completely forget $\mathbf{x}^*$.

The first thing to note is that $\hat{\mathbf{x}}_i^*$ does not appear in any terms in the expression until $i = i^*$. For this index, we have i^* terms dependent on $\hat{\mathbf{x}}_{i^*}$ as every term in the second sum in Equation 6 is multiplied by $\frac{\partial^2 \mathcal{L}}{\partial^2 \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_{i^*}}$. Now for all the indices $i > i^*$ (*i.e.,* $t - 1 - i^*$ indices), we have exactly one term with $\hat{\mathbf{x}}_{i^*}$ which comes from a $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_0, \hat{\mathbf{x}}_{i^*}}$ appearing in the second sum. In total, we have $(t - 1 - i^*) + i^* = t - 1$ terms, and note that all these terms are hessian vector products (because all the terms in Equation 6 are hessian vector products). In general, it is possible to implement hessian products in $O(n)$ [32].

Importantly, hessian vector products are at least as expensive as a gradient computation. When batch size $b = 1$, this would be equivalent to the cost for training, and in general is $\frac{1}{b}\times$ the cost of training as we only need to compute 1 of the b gradients per batch, *i.e.,* just with respect to $\mathbf{x}^*$. However, this can still be a significant amount of computational expense in practice (as t here is not the number of epochs but the number of individual training steps), and so from counting the terms with $\mathbf{x}^*$ we see that this second sum in Equation 4 (which we can approximate with Equation 6) is a bound to how well we can reasonably forget the effect of $\mathbf{x}^*$.

5.3. Unlearning Error

Based on the above discussion for estimating the second sum, and how it bounds the efficacy of unlearning (without adding significant computational costs), we define Equation 8 to be our *unlearning error* (e). In practice, we make a slight modification as this bound can be loose; we take σ to be the average of all the $\sigma_{1,i}$ rather than the $\max$, as to better approximate Equation 6. To be precise, setting $\sigma_{avg} = \frac{1}{t}\sum_{i=1}^{t} \sigma_{1,i}$, we define unlearning error as:

$$\boxed{e = \eta^2 \cdot \frac{||\mathbf{w}_t - \mathbf{w}_0||_2}{t} \cdot \sigma_{avg} \cdot \frac{t^2 - t}{2} \quad (9)}$$

Working with the average also allows one to only compute $\sigma_{1,i}$ every couple of steps, saving costs. Lastly, computing unlearning error e does not require us to have applied our unlearning method first; we can pre-emptively know what it will be.

6. Reducing the Unlearning Error

In § 5, we identified terms of SGD which force (post-training) approximate unlearning approaches to incur verification error. Our unlearning error captures these effects through the number of steps t, the average singular value σ_{avg}, and the difference between final and initial weights $||\mathbf{w}_t - \mathbf{w}_0||_2$. Our closed form approximation of SGD from the previous section leads to an unlearning method (see gray box in § 5) that only depends on the initial weights $\mathbf{w}_0$. We now introduce modifications to the training procedure, which are orthogonal to the unlearning method itself but reduce the values of the different variables contributing to unlearning error. This therefore yields a training algorithm which makes it *easier to later unlearn* with our method at a minimal verification error.

6.1. Strawman Approaches

We first consider two[6] strawman approaches, each trying to directly minimize one of three factors identified

6. We also tried several other ideas at reducing unlearning error which we do not describe in great detail here. One such idea was trying to reduce σ_{avg} by adding a regularization term with the sum of the diagonal entries of the Hessian of CE loss with respect to the logits squared (which are of the form $\{p_1(1-p_1), \cdots, p_c(1-p_c)\}$ where p_i is the softmax output of logit $M(\mathbf{x})_i$), but as will be theme with this section, we saw no benefit.

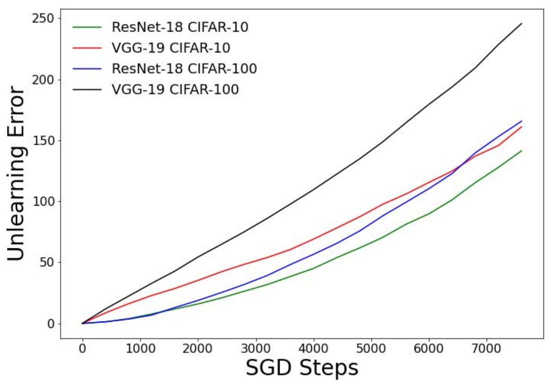


Figure 3. Unlearning error for 4 different settings as a function of the number of finetuning steps with no pretraining. Across all 4 settings, the unlearning error increases as a function of t. Each model was trained for $t = 7812$ steps (5 epochs)

in the unlearning error formulation: the number of steps t and the difference between final and initial weights $||\mathbf{w}_t - \mathbf{w}_0||_2$.

1. Training for less. From Equation 9, recall that unlearning error e depends on number of steps t. A simple method to reduce e would be to train for fewer steps. We verify empirically if training for less steps t reduces e.

Our evaluation is performed on two canonical architectures for computer vision, ResNet-18 [30] and VGG-19 [31], on two vision datasets, CIFAR-10 [29] and CIFAR-100 [29]. The datasets consist of 60,000 $32 \times 32 \times 3$ images that belong to 10 and 100 classes respectively. The classes include animals and objects. This leads to four settings (one for each model and dataset combination). For a dataset D, we denote by M_t the model trained on D for t training steps starting from M_0; in the experiments we compute the unlearning error from M_0 to M_t, and record the intermediate values of unlearning error e.

Figure 3 illustrates how e increases (practically linearly) with t. This implies that by training less, we decrease e almost proportionally. This however also degrades the model's performance (*i.e.,* prediction accuracy) which creates an undesirable trade-off. One way to alleviate this limitation would be to pre-train the model on a public dataset so as to start training at a better initialization and compensate for the lower number of training steps performed on data that may be unlearned. However, it is not always possible to find a public dataset for which no unlearning requests will be issued.

2. ℓ_2 regularization. The next factor involved in the unlearning error e is the weight difference $||\mathbf{w}_t - \mathbf{w}_0||_2$; decreasing this term (while not changing anything else) would also decrease unlearning error e. By the triangle inequality, we have that $||\mathbf{w}_t - \mathbf{w}_0||_2 \leq ||\mathbf{w}_t||_2 + ||\mathbf{w}_0||_2$ and so it would seem that by decreasing either $||\mathbf{w}_t||_2$ or $||\mathbf{w}_0||_2$, we might be able to obtain a tighter bound on $||\mathbf{w}_t - \mathbf{w}_0||_2$ and decrease it. This reasoning motivates using ℓ_2 regularization which decreases the norm of the final weights $||\mathbf{w}_t||_2$ by adding a $\lambda||w||_2$ regularization term (where λ is a regularization constant) to the CE loss.

To evaluate this regularizer, we use the same setup that was previously described. We repeat the experiments for different values of regularization strength: we pick

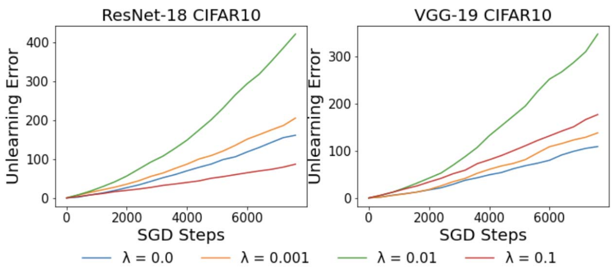


Figure 4. Unlearning error of 4 setups trained with increasing strength of ℓ_2 regularization. As shown, weight decay does not decrease the unlearning error consistently.

$\lambda \in \{0.0, 0.001, 0.01, 0.1\}$. In Figure 4, we track e for $t = 7812$ training steps (5 epochs) for the two CIFAR-10 models (Table 3 includes CIFAR-100 results) and the various strengths of ℓ_2 regularization. We observe that there was no consistent benefit to employing the loss penalty, and in most cases it in fact increased e. The main issue is that the bound $||\mathbf{w}_t - \mathbf{w}_0||_2 \leq ||\mathbf{w}_t||_2 + ||\mathbf{w}_0||_2$ is very loose and we can decrease the right hand side without decreasing the left hand-side (the part which we are interested in) as seen in Figure 16. Thus, we next refine our analysis to obtain a tighter bound and a more effective regularization penalty.

6.2. Our Proposal: Standard Deviation Loss

In § 6.1, we considered two strawman approaches: decreasing training steps t and ℓ_2 regularization. What we proceed to do in this section is one of the main contributions of this work: we introduce our own novel standard deviation regularizer. By way of studying a binary linear classifier, we show how it would (when training on an arbitrary, single data point) decrease the final change in weights by moving the minimum of the loss closer to the initial weights. Observe then that as long as this loss does not also increase the singular values (which we later empirically show in § 8.2), it decreases the unlearning error e. Informally, the loss yields a model whose final weights required smaller gradient descent steps to get to from the initial weights. This explains why unlearning the contributions of a point to these steps using our single gradient approach leads to a smaller unlearning error when the model was trained with our loss. This loss, which we will call *standard deviation (SD) loss*, is defined as follows:

$$\begin{aligned}&\mathcal{L}_{SD}(M(\mathbf{x}), \mathbf{y}) \\ &= \mathcal{L}_{CE}(M(\mathbf{x}), \mathbf{y}) + \gamma \sqrt{(\sum_{i=1}^{c}(M(\mathbf{x})_i - \mu)^2)/c}\end{aligned} \tag{10}$$

where c is the output dimension of the model M, γ is the regularization strength, and μ is the average value of the output logits $M(\mathbf{x})$ for a specific datapoint $\mathbf{x}$. Note that the SD loss is simple to integrate into a training framework as it just requires adding a regularization term to the loss function.

A Simple Binary classifier. Next, we illustrate our intuition behind the SD loss with the case of a binary linear classifier (*i.e.,* a linear model with just two outputs) trained to minimize its loss on a single datapoint; we show how the SD loss reduces the minimum distance (in the weight space) to the minimum of the loss (which is where the final weights are found). This results in a lower weight difference $||\mathbf{w}_t - \mathbf{w}_0||_2$, which in turn reduces unlearning error. This holds as long as the SD loss does not also simultaneously increase other variables involved in the definition of our unlearning error, *e.g.,* singular values. We confirm later in § 8.1 that this is the case (even for the neural networks we considered in our evaluation).

Let us consider the specific setup where $\mathbf{x}$ is a vector of length k representing a datapoint, $\mathbf{w} = \begin{bmatrix}\mathbf{w}_1\\ \mathbf{w}_2\end{bmatrix}$ is a $2{\times}k$ weight matrix, and M is a linear model with two outputs defined by $M(\mathbf{x}) = \mathbf{wx}$. For the sake of simplicity (and without loss of generality) let us assume the label of $\mathbf{x}$ is 0. Then, the SD loss of our model, where $a = M(\mathbf{x})_1$ and $b = M(\mathbf{x})_2$ (*i.e.,* the first and second output), is:

$$\mathcal{L}_{SD}(M(\mathbf{x}), 0) = -\log(\frac{e^a}{e^a + e^b}) + \gamma\sqrt{\frac{a^2 + b^2 - 0.5(a+b)^2}{2}} \tag{11}$$

where the second term is just a simplified version of the standard deviation.

The main empirical observation motivating the following analysis is the following: increasing the strength of γ moves the minimum of the SD loss with respect to the outputs a and b (given by Equation 11) closer to the line defined by $a = b$. This is illustrated in Figure 5 where we plot the gradients of our loss with respect to the outputs to observe where they are 0 (*i.e.,* when the direction flips). Similarly, Figure 6 illustrates how the minimum of the loss approaches the line $a = b$ (in black), and generally how the minimum is defined by a line $a = b + \epsilon$ where ϵ decreases with the strength of γ.

We now proceed to analytically show how SD loss reduces the minimum change in weights required to reach a minimum of the loss (for a binary linear classifier). We focus on the initialization of **w** by a constant (that is all the entries are the same), and thus $a_0 = \mathbf{w}_1 \cdot \mathbf{x} = \mathbf{w}_2 \cdot \mathbf{x} = b_0$. To find the minimum distance in the weights space to a minimum of the loss, we need to solve the constrained optimization problem

$$\begin{gathered}\min_{\mathbf{u}_1, \mathbf{u}_2} ||\mathbf{u}_1||_2^2 + ||\mathbf{u}_2||_2^2 \\ \text{subject to} \\ a_0 + \mathbf{u}_1 \cdot \mathbf{x} = b_0 + \mathbf{u}_2 \cdot \mathbf{x} + \epsilon \\ \implies (\mathbf{u}_1 - \mathbf{u}_2) \cdot \mathbf{x} - \epsilon = 0\end{gathered} \tag{12}$$

where $\mathbf{u}_1$ and $\mathbf{u}_2$ are the updates to $\mathbf{w}_1$ and $\mathbf{w}_2$ respectively to reach the minimum of the loss.

The solution, given by minimizing the lagrangian $\mathfrak{L} = \mathbf{u}_1 \cdot \mathbf{u}_1 + \mathbf{u}_2 \cdot \mathbf{u}_2 + \lambda((\mathbf{u}_1 - \mathbf{u}_2) \cdot \mathbf{x} - \epsilon)$, is $\mathbf{u}_1 = \frac{\epsilon}{2||\mathbf{x}||_2^2}\mathbf{x}$ and $\mathbf{u}_2 = \frac{-\epsilon}{2||\mathbf{x}||_2^2}\mathbf{x}$. Note that $||\mathbf{u}_1||_2^2 + ||\mathbf{u}_2||_2^2 = \frac{\epsilon^2}{2}||\mathbf{x}||_2^2$, and by reducing ϵ we reduce the magnitude of the change of weights needed to reach a minimum of the loss. As increasing the strength of our SD loss regularization (γ) does just that, we see that increasing γ means decreasing the minimum change in weights needed to reach a minimum (which is what the path following negative gradients approximates). Thus, if the SD loss does not (also) increase the singular values, it decreases the unlearning error. As a final remark, we showed that a state with $a = b$ is close to a minimum, but this implies a more general result of just being close to such a state where $a = b$ (*i.e.,* having low standard deviation in the outputs) means being close to the minimum.

7. Implementation

All of the experiments we describe next (and those performed in § 6.1) were conducted on T4 Nvidia GPUs with 16 GB of dedicated memory. We used Intel Xeon Silver 4110 CPUs with 8 cores each and 32GB of RAM. The underlying OS is Ubuntu 18.04.2 LTS 64 bit. We use `pytorch` v1.8.1 with CUDA 10.2 and `python 3.7`. We used the same datasets and models as described in § 6.1. We also evaluate our proposal on a different domain, *text*, using a pre-trained DistilBERT [33] fineunted on IMDb reviews [34] for sentiment analysis (a binary classification task); note IMDb has $25,000$ training and test datapoints consisting of sentences of varying lengths which we truncated to the first 512 tokens (the maximum DistilBert takes as input). Note the size and variety of the datasets and models used in our evaluation is comparable or exceeds previous work in the topic of unlearning [22], [9], [11].

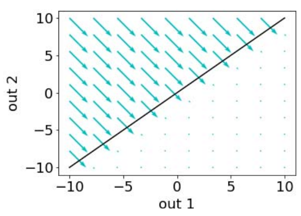
8. Evaluation

Through our evaluation, we wish to answer the following questions.

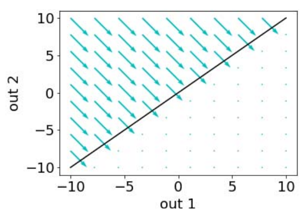
1) Does SD loss decrease unlearning error?
2) How are the various components of the unlearning error (t, σ_{avg}, $||\mathbf{w}_t - \mathbf{w}_0||_2$) affected by training with SD loss?
3) Does decreasing unlearning error (either using SD loss or changing t) result in a decrease in verification error? Particularly, are they linearly related (strong Pearson correlation)?
4) Can the effect of single gradient unlearning be measured by other metrics, such as PRS?

To answer the questions above, we utilize our single gradient unlearning approach to obtain the approximately unlearned model. Our salient results are:

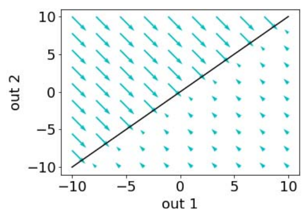
1) In § 8.1, we conclude that the proposed SD loss is effective at decreasing the unlearning error. This is achieved at a moderate penalty to accuracy (<3 percentage points in the setups we tested).
2) In § 8.2, we see that SD loss significantly decreases the final change in weights $||\mathbf{w}_0 - \mathbf{w}_t||_2$, and that the impact of SD loss for a given regularization strength is concentrated at the beginning of training
3) In § 8.3, we see that decreasing unlearning error with any of the methods we have considered also decreases verification error (*i.e.,* they are strongly correlated) showing that unlearning error serves as a good proxy metric.
4) In § 8.4, we observe that single gradient unlearning decreases the PRS of the baseline (*i.e.,* $\gamma = 0$). However, cases where $\gamma > 0$ has no consistent effect.



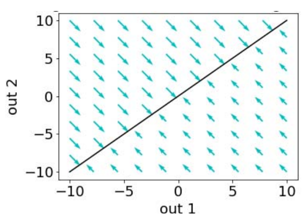
(a) $\gamma = 0.01$



(b) $\gamma = 0.1$

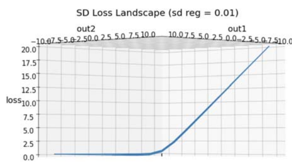


(c) $\gamma = 1.0$

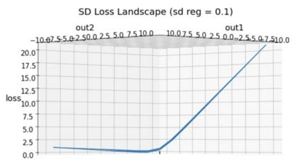


(d) $\gamma = 5.0$

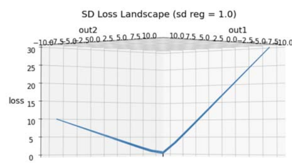
Figure 5. Plots of the negative gradients of SD loss with respect to the two outputs ($a = out1$, $b = out2$). The black line represents $a = b$. Observe how the minimum of the loss landscape (where the arrows switch direction) approaches the black line.



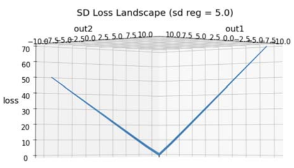
(a) $\gamma = 0.01$



(b) $\gamma = 0.1$



(c) $\gamma = 1.0$



(d) $\gamma = 5.0$

Figure 6. Plots of the SD loss landscape with respect to the two outputs ($a = out1$, $b = out2$). Note the black dot represents where $a = b$, and observe how the minimum of the loss landscape approaches the black dot

Note that while our theory is for batch size $b = 1$, in particular Lemma 1 and Corollary 1, our evaluation is for $b > 1$ (where results are consistent across batch sizes). We believe future work may extend our theory for $b > 1$.

8.1. SD Loss Decreases Unlearning Error

Recall that we introduce the SD loss as a method to reduce the unlearning error as training commences. We wish to empirically validate this. We utilize the implementation setup details described in § 7, and the following setup:

1) Train M_0 on D with SD loss for N pre-training steps to obtain M_N.
2) Train M_N for an additional t steps on $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \cdots, \hat{\mathbf{x}}_t$ (from D) to get M_{N+t}.
3) Compute the unlearning error (Equation 9) of M_{N+t} starting from M_N, except here $\mathbf{w}_t$ is the weights of M_{N+t} and $\mathbf{w}_0$ are the weights of M_N.

In this setup, N is a configurable parameter which allows us to understand how the amount of training affects the impact of SD loss on unlearning error (see § 8.2). For the vision models, we utilize $N = \{0, 15625, 31250, 46875, 62500, 78125\}$ steps (*i.e.,* $0, 10, 20, 30, 40, 50$ epochs, respectively). For the regularization strength, we use $\gamma = \{0, 1, 5, 10, 15, 20\}$ for CIFAR-10 and $\gamma = \{0, 50, 100, 150, 200, 250\}$ for CIFAR-100. In the case of DistilBERT fine-tuned on IMDB reviews, we observed we reached peak accuracy after just $N = 4688$ steps (or 3 epochs) and simply evaluated that for $\gamma = \{0.0, 0.01, 0.05, 0.1, 0.15, 1.0, 1.25, 1.5\}$ and discuss that independent of t.[7]

Results: In Figure 7, we plot the unlearning error as a function of the number of training steps t. Observe that, as expected, the unlearning error grows as the training duration increases. However, increasing the regularization parameter γ (from 0 *i.e.,* CE loss, to > 0 values) reduces the rate or growth, and thus the final unlearning error. This is consistent with the final unlearning errors we observed for DistilBERT in Table 2 where we see increasing regularization dropped final unlearning error by $40\times$.

The influence of SD loss on the accuracy of the final (vision) models is presented in detail in Table 4 in Appendix C; Table 1 highlights results from our experiments with CIFAR-10 for $N = 10$ epochs. Results for the DistilBERT experiments are in Table 2. For the vision models, note that as we increase the value of γ, the accuracy of the classifier learnt decreases. From Table 1, the decrease is within 5 percentage points for values of $\gamma \leq 5$. For values of $\gamma > 5$, the drop in accuracy is larger. This can be explained by having to fit to the dataset while also balancing the requirement of minimizing $||\mathbf{w}_t - \mathbf{w}_0||_2$. We will demonstrate this effect in more detail in § 8.2. However, observe that the experiments with DistilBERT indicate a subtle increase in accuracy even after the regularization had cut the unlearning error by $2.5\times$, suggesting some domains/models can handle the regulariation better. More detailed analysis is required to explain this phenomenon.

TABLE 1. UNLEARNING ERROR (e) AND TESTING ACCURACY ON CIFAR-10 FOR A PRETRAINING AMOUNT OF 10 EPOCHS AND t EQUAL TO 1 EPOCH.

Regularization	**ResNet-18**		**VGG-19**	
(γ)	e	**Acc (%)**	e	**Acc (%)**
0.0	132.49	77.11	141.73	77.69
1.0	101.79	74.5	123.17	76.81
5.0	98.3	73.21	113.27	77.16
10.0	90.71	69.13	107.9	74.07
15.0	86.17	68.46	78.82	65.32
20.0	11.5	34.74	27.94	31.18

8.2. An Ablation Study

Recall that the unlearning error is dependent on t, σ_{avg}, $\mathbf{w}_t$, and also the value of N. As we can calculate the unlearning error with respect to different stages of training by varying the value of N, we can obtain a more nuanced

7. $\gamma = 0$ means we just train with CE loss (which acts as our baseline)

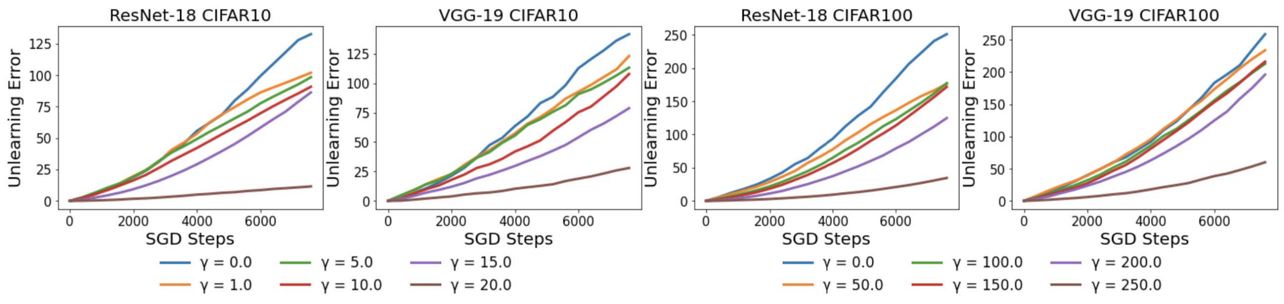


Figure 7. Unlearning error of 4 setups trained with the standard deviation loss with increasing levels of SD regularization. Across all 4 setups, a stronger regularization decreases the unlearning error at a (minimal) cost of the performance of the model.

TABLE 2. UNLEARNING ERROR e, VERIFICATION ERROR v AND TESTING ACCURACY FOR DISTILBERT ON THE IMDB DATASET FOR VARYING REGULARIZATION CONSTANTS. NOTE THE PRETRAINING AMOUNT IS 3 EPOCHS AND t IS 1 EPOCH.

Regularization (γ)	e	v	**Acc (%)**
0	85.10	4.35	92.13
0.01	85.89	4.27	92.53
0.1	71.73	3.43	92.38
0.5	31.98	1.57	92.71
1.0	8.91	0.57	92.09
1.25	5.60	0.43	87.22
1.5	2.57	0.48	84.99

view of how it evolves with respect to these factors. All results in this subsection are presented for the vision models only. The results are plotted for $N = 15,625$ steps (10 epochs), and $t = 7812$ steps (5 epochs) in Figure 8, but the trends and ensuing discussion hold for all other values of N.

Singular values & change in weights. The impact of SD loss on σ_{avg} and the final change in weights and largest singular values are presented in Figure 8 (left). We see that the impact to the singular values is *minimal*. After some initial oscillation, SD loss leaves σ_{avg} unchanged. Thus, the effect on reducing the unlearning error comes from elsewhere; we can see that SD loss *does decrease* the change in weights $||\mathbf{w}_t - \mathbf{w}_0||_2$ significantly with increasing γ[8], as suggested by our analysis in § 6.2 (refer to Figure 8 (right)). Recall that the change in weights influences the unlearning error, and smaller values of $||\mathbf{w}_t - \mathbf{w}_0||_2$ induces smaller values of the unleraning error.

Influence of N**.** We wish to answer a more nuanced question: *how does the impact on SD loss vary as training progresses i.e., as N increases?* The results are presented in Table 4 in Appendix C. We observe that as N increases, the decrease in unlearning error caused by a given regularization diminishes (with the exception of ResNet-18 on CIFAR-10). Looking at VGG-19 on CIFAR-10 in Table 4, we see w.r.t the $N = 10$ *epochs of pre-training* row, $\gamma = 5$ decreases the unlearning error by 20% relative to the baseline ($\gamma = 0$). With respect to the $N = 30$ *epochs of pre-training* row, $\gamma = 5$ results in a 4% drop relative to the baseline ($\gamma = 0$). This reduced effect is due

8. Here w_t are the weights of the model M_{N+t}, and w_0 of the model M_N.

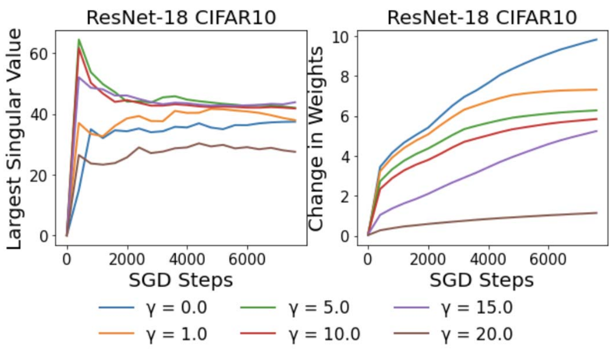


Figure 8. Largest singular value and change in weights over training for different regularization strengths for ResNet-18 on CIFAR-10. The change in weights decrease consistently and significantly as the regularization strength increases.

to the model converging (or being close to converging) to its final weights: recall that in our setup, the weights obtained after N steps of training are considered as $\mathbf{w}_0$; as N increases, the value of $\mathbf{w}_t$ (which are the weights obtained after $N + t$ steps of training) are close to $\mathbf{w}_0$.

8.3. Unlearning Error & Verification Error

Having established the relationship between unlearning error and SD loss, we wish to validate if unlearning error is a good proxy for verification error. To this end, we evaluate the relationship between unlearning error and verification error with a specific emphasis on training duration t, regularization strength γ, and a combination of the two datasets and model architectures. For this experiment, we utilize the setup (and notation) from § 8.1 (*i.e.,* both vision and text models) to compute unlearning error and the following setup to compute verification error:

1) Train M_N for $t-1$ steps on batches $\hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_t$. The resulting model is the naively retrained model M' without batch $\hat{\mathbf{x}}_1$ (where note $\hat{\mathbf{x}}_1$ varies as we vary N) with weights $\mathbf{w}'$.
2) Obtain the approximately unlearned model M'' (in particular $\mathbf{w}''$) using our single gradient method as follows:
$$\mathbf{w}'' = \mathbf{w}_{N+t} + \frac{\eta}{b}\frac{\partial \mathcal{L}}{\partial \mathbf{w}}|_{\mathbf{w}_{n+t},\hat{\mathbf{x}}_1}$$
where $\mathbf{w}_{N+t}$ were the weights of M_{N+t}.
3) Compute the verification error (*i.e.,* $||\mathbf{w}' - \mathbf{w}''||_2$).

Changing t: In Figure 3 in § 6.1, we saw that unlearning error increases with t. To this end, we utilize the ResNet-18 model trained using CIFAR-10. We wish to understand if verification error also increases with t. Figure 9 (left) shows the relationship between the verification error and the unlearning error as a function of the number of steps t taken, for $t = 15,625$ (5 epochs) steps after $N = 0$ steps. Observe that the two are strongly correlated (with a Pearson's coefficient of 0.934). When we increase the value of N to $250,000$ steps (80 epochs) to measure how this effect is at a different stage of training (for the same value of t as before), we observe the same effect: there is a strong correlation between verification and unlearning error (with a Pearson's coefficient of 0.9668); refer Figure 9 (right).

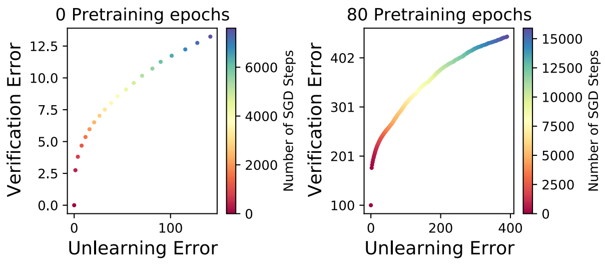


Figure 9. Unlearning and verification error as a function of the fine-tuning steps (indicated by the color gradient) for ResNet-18 on CIFAR-10 for $N = 0$ steps, $t = 7,812$ steps (5 epochs) and $N = 250,000$ steps (80 epochs), $t = 15,625$ (5 epochs). Unlearning and verification error are measured every 400 and 100 SGD updates, respectively. The Pearson correlation between unlearning and verification error is 0.934 and 0.9668, respectively. Note that t is the only variable giving a distribution. Both unlearning error and verification error increase as a function of the number of fine-tuning steps.

Changing γ: Similarly, from Figure 11, we can observe that varying the strength of our regularization (γ) results in a strong linear correlation between unlearning error and the verification error. For ResNet-18 trained on CIFAR-10, the Pearson coefficient is 0.96, and the Pearson coefficient is 0.81 for VGG-19 trained on CIFAR-10. Furthermore for DistilBERT fine-tuned on IMDB reviews we observe a Pearson Coefficient of 0.998 (see Figure 10). Also notice that for large values of γ (*i.e.,* the blue points near the origin), both verification and unlearning error are low. This further validates the efficacy of SD loss.

Considering all models: The goal now is to see if unlearning error can be used to compare which architectures and training setups are better at unlearning (*i.e.,* does unlearning error strongly correlate with verification error across training setups). This is seen in Figure 12 where we consider 160 different training setups for CIFAR-10 and CIFAR-100[9]; we observe that generally when a model with a specific setup has lower unlearning error, it has lower verification error, and this relation is almost linear.

9. We consider different amounts of pre-training $N = \{10, 20, \dots, 80\}$ epochs, batch sizes $\{32, 64, 128\}$, SD regularization strengths (values between the ranges shown in Figure 7), models (ResNet-18 and VGG-19) and batch size over which the hessian is calculated.

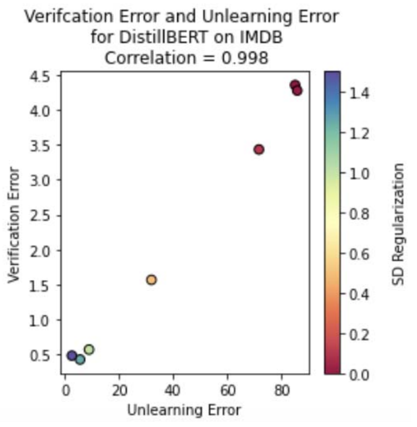


Figure 10. The Pearson correlation of verification error and unlearning error for DistilBERT on the IMDB dataset when varying the SD regularization strength.

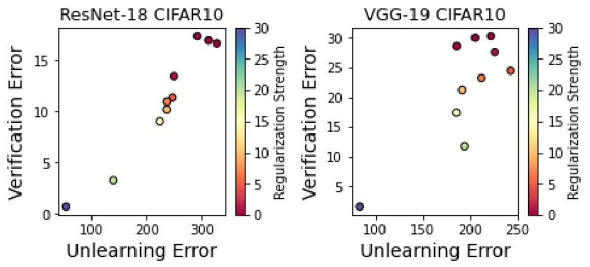


Figure 11. Pearson correlation between unlearning error and verification error for ResNet-18 and VGG-19 on CIFAR-10 with regularization strength γ being the only variable giving the distribution. The correlations are both high at 0.96 and 0.81 for ResNet-18 and VGG-19, respectively. Notice that the stronger the regularization (more blue) the more the verification and unlearning error decrease in both settings.

This can be observed by looking at the strong positive Pearson correlation between unlearning error and the verification error across the different settings and architecture considered.

The fact that unlearning error is a good proxy for the verification error serves as evidence that the mathematical basis for unlearning error presented in § 5.2 holds for the different architectures, domains, and datasets we considered, despite the approximations we made to arrive at the definition of unlearning error. Another takeaway from this result, and the fact that the correlations stand across different architectures, is that unlearning error can be used to compare how well different models with potentially different architectures unlearn, leading to the possible study of what architectures unlearn better. We leave this aspect to future work.

8.4. The Effect on PRS

We now study the effect of our single gradient unlearning method and training using SD loss on PRS, a metric which represents MI confidence. To do so, we calculate the PRS of the point to be unlearned before and after applying our unlearning method (when training was

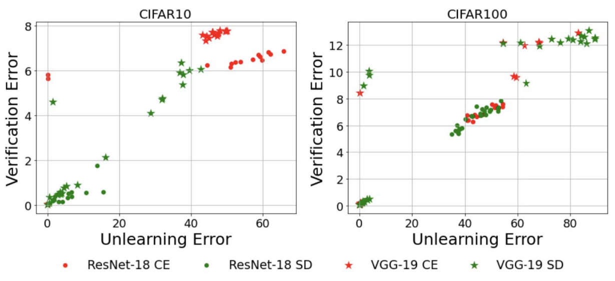


Figure 12. Unlearning and verification error for 160 settings across CIFAR-10 and CIFAR-100. The Pearson correlation between unlearning error and verification error for CIFAR-10 and CIFAR-100 are **0.9028** and **0.8813**, respectively. Settings trained with SD loss (green) for both datasets have lower unlearning and verification error then those trained with regular CE (red).

performed with varying strengths of SD regularization). The results are presented in Figure 13.

Observe that by applying single gradient unlearning on the baseline case (*i.e.,* $\gamma = 0$), we can consistently decrease the PRS by roughly $2\times$. On average, varying the regularizaion strength γ decreases the PRS. However we find no consistent monotonic relation between SD regularization and PRS, as seen by the varying Spearman correlations (which switch sign and are often weak). Note we use Spearman here as we are asking if there is any monotonic relation, not necessarily linear. Regardless, the fact that SD regularization is not needed to decrease the PRS with our unlearning method, but greatly decreases verification error, highlights the dependence between seemingly independent methods.

8.5. SISA Cost Comparison

An interesting question is whether exact unlearning could ever be cheaper than approximate unlearning. As our method is a very cheap form of approximate unlearning, and SISA is the cheapest exact unlearning method, we compare how the costs of exactly unlearning with SISA [7] compares to our approximate unlearning costs. The cost of unlearning with SISA is at best $\frac{2S}{(R+1)}\times$ cost-to-retrain (R is number-of-slices and S is number-of-shards). Additionally, this requires $O(S.R)$ storage. Our cost is computing a single gradient (cost-to-retrain$/N$ where N is number-of-steps) and requires no storage. When $\frac{2S}{(R+1)} < \frac{1}{N}$ SISA could be faster, but then R or S is at least $\frac{\sqrt{N}}{2} - 1$ (and typically $N \approx 100,000$). Thus R and S would be significantly higher than what was originally tested by Bourtoule *et al.*, and note that increasing S was observed to decrease performance, but future work may investigate training with large S and R.

9. Discussion

We focus on discussing open questions raised.

When Is unlearning achieved: The immediate question raised by our work is deciding when an entity has unlearnt. We see from the disparity between verification error and PRS that decreasing one does not necessarily mean decreasing the other. More over with verification error and unlearning error, we can reason about how much it has decreased from a baseline error (*i.e.,* the error with regularization $\gamma = 0$), but we do not have a scale of how much is enough. We believe answering when unlearning is achieved is an application oriented problem and depends, for instance, on what the user wants unlearning to achieve.

Experimental constraints: We leverage the widely used implementation from `kaungliu` [10] which achieves a 93% test accuracy (on CIFAR-10) with ResNet18. We remove several enhancements, specifically:

1) Learning rate (η) scheduler: our derivation assumes a constant LR.
2) Data augmentation: introduces multiple copies of the unlearned data.
3) Momentum/ADAM: as our approximation was focused on SGD.

These restrictions are common when learning with differential privacy, and the drop to performance we experience on CIFAR-10 and CIFAR-100 is less significant. Nevertheless future work may be able to improve the performance by utilising other enhancements or dropping some of these restrictions. It is worth noting however that DistilBERT was still able to achieve high test accuracy.

Unlearning error and verification error for other unlearning methods: An interesting question is whether our unlearning error serves as a good proxy for verification error even when employing unlearning methods besides our own. We focused on amnesiac machine learning [8], and computed the analogous result of Figure 12 in Figure 14 where we once again vary different batch sizes, training amounts, regularization strength(s), etc. except now M'' is obtained via amnesiac machine learning. As was the case for our approach, we see very strong correlations between unlearning error and verification error of 0.91 on CIFAR-10 and 0.81 on CIFAR-100. We similarly computed the analogous result of Figure 9 in Figure 15 and got comparable results to using our unlearning method. This begs the question of whether unlearning error can be used to assess the verification error (by proxy) of other approximate methods for unlearning in SGD.

Expanding analysis of verification error: However, it should be noted that though amnesiac machine learning retains a strong correlation, it does not solely depend on $\mathbf{w}_0$ or I. This is because it computes gradients with respect to intermediate weights during training; training from the same $\mathbf{w}_0$ with the same batch ordering I, we get significantly different weights during the end of training. Thus, it violates the assumptions made by our analysis as it introduces significant noise we did not account for in our proofs. Thus the bounds presented in § 4 do not apply (*i.e.,* we are not sure if lowering verification error with these methods lowers the difference in probability distributions). However this does not mean similar bounds do not exist, and we leave it for future work to expand these results to other classes of unlearning methods and possibly expanding our existing assumptions.

Architectures and unlearning error: Recall from § 8.1, that the results of the ResNet architecture on CIFAR-10

10. https://github.com/kuangliu/pytorch-cifar

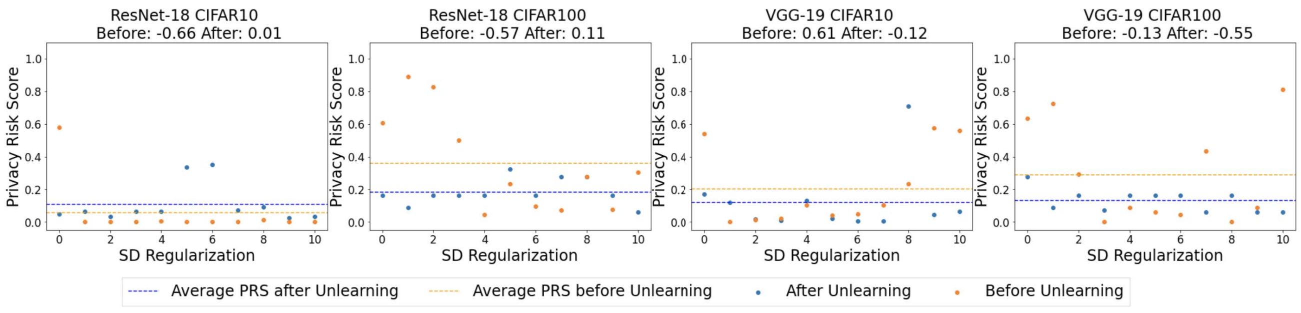


Figure 13. PRS of point to unlearn before and after unlearning as a function of the SD regularization trained with. Spearman correlation coefficients before and after unlearning are shown in the title of each of the plots. Note the inconsistencies in signs and magnitude of correlation before and after unlearning in all 4 settings.

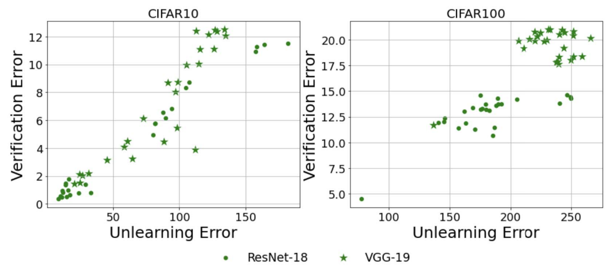


Figure 14. Unlearning and verification error for 104 settings across CIFAR-10 and CIFAR-100 using **amnesiac unlearning**. Correlations between unlearning error and verification error for CIFAR-10 and CIFAR-100 are resp. **0.91** and **0.81**.

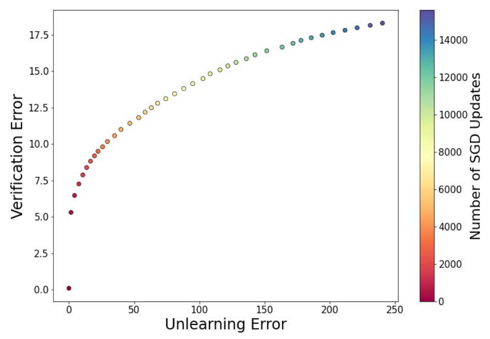


Figure 15. Unlearning error and verification error as a function of the finetuning steps (indicated by the color gradient) for ResNet-18 on CIFAR-10 using **amnesiac unlearning**

were atypical. The SD loss on ResNet-18 continued to significantly reduce unlearning error into the later stages of training and in general dropped unlearning error with a smaller cost to performance compared to VGG-19; we also see from Table 4 that though SD loss effect diminishes for ResNet-18 on CIFAR-100, it is significantly less so than VGG-19. Similarly, DistilBERT in fact was able to drop unlearning error by a magnitude while seeing less significant drops to accuracy. This begs the question, are some models or domains just better for unlearning? We believe an interesting direction for future work is to pinpoint what architectural or domain features make unlearning easier or harder.

Regularizers and unlearning error: Similarly future work might look into better regularizers, potentially ones focusing on reducing singular values, and improving the performance of our SD loss by a regularization scheduler that accounts for the degrading impact we noted in § 8.2.

10. Conclusion

In this paper we first discussed past work on approximate unlearning, noting how there was a great breadth to what metric one can use to define unlearning. Following this we showed how verification error captures a large class of unlearning metrics (under some assumptions), motivating using it to define an approximate unlearning method. However, as verification error cannot be optimized directly when devising an unlearning method (as it requires a perfectly unlearned model for computation), we decomposed SGD with a Taylor series to first propose our single-gradient unlearning method, and furthermore propose our *unlearning error* which we showed effectively proxies verification error when using our unlearning method. To then improve the effectiveness of our unlearning approach, we looked at the variables our proxy metric depends on and proposed our SD loss which we showed can effectively decrease the unlearning error (and thus the verification error) associated with our unlearning method. We expect that future work will try and improve the loss we used in our work, or see how it may help with different machine learning related tasks, and extend the bounds we presented in our work to other contexts and see if certain assumptions can be dropped.

Acknowledgements

We would like to acknowledge our sponsors, who support our research with financial and in-kind contributions: CIFAR through the Canada CIFAR AI Chair program, DARPA through the GARD program, Intel, Meta, NFRF through an Exploration grant, and NSERC through the Discovery Grant and COHESA Strategic Alliance. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute.

Availability

Relevant code for the above experiments can be found at https://github.com/cleverhans-lab/unrolling-sgd.

References

[1] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 3–18.

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

[3] A. Mantelero, "The eu proposal for a general data protection regulation and the roots of the 'right to be forgotten'," *Computer Law & Security Review*, vol. 29, no. 3, pp. 229–235, 2013.

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

[5] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1322–1333.

[6] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 463–480.

[7] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," *arXiv preprint arXiv:1912.03817*, 2019.

[8] L. Graves, V. Nagisetty, and V. Ganesh, "Amnesiac machine learning," *arXiv preprint arXiv:2010.10981*, 2020.

[9] A. Golatkar, A. Achille, and S. 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*, 2020, pp. 9304–9312.

[10] A. Sekhari, J. Acharya, G. Kamath, and A. T. Suresh, "Remember what you want to forget: Algorithms for machine unlearning," *arXiv preprint arXiv:2103.03279*, 2021.

[11] C. Guo, T. Goldstein, A. Hannun, and L. Van Der Maaten, "Certified data removal from machine learning models," *arXiv preprint arXiv:1911.03030*, 2019.

[12] T. Baumhauer, P. Schöttle, and M. Zeppelzauer, "Machine unlearning: Linear filtration for logit-based classifiers," *arXiv preprint arXiv:2002.02730*, 2020.

[13] A. Golatkar, A. Achille, and S. Soatto, "Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations," *CoRR*, vol. abs/2003.02960, 2020. [Online]. Available: https://arxiv.org/abs/2003.02960

[14] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang, "When machine unlearning jeopardizes privacy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 896–911. [Online]. Available: https://doi.org/10.1145/3460120.3484756

[15] V. Gupta, C. Jung, S. Neel, A. Roth, S. Sharifi-Malvajerdi, and C. Waites, "Adaptive machine unlearning," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[16] S. Zanella-Béguelin, L. Wutschitz, S. Tople, V. Rühle, A. Paverd, O. Ohrimenko, B. Köpf, and M. Brockschmidt, *Analyzing Information Leakage of Updates to Natural Language Models*. New York, NY, USA: Association for Computing Machinery, 2020, p. 363–375. [Online]. Available: https://doi.org/10.1145/3372297.3417880

[17] T. Hastie, R. Tibshirani, and J. Friedman, "Overview of supervised learning," in *The elements of statistical learning*. Springer, 2009, pp. 9–41.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[19] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 20, no. 2, pp. 215–232, 1958.

[20] J. Kiefer, J. Wolfowitz *et al.*, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, vol. 23, no. 3, pp. 462–466, 1952.

[21] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[22] Y. Wu, E. Dobriban, and S. B. Davidson, "Deltagrad: Rapid retraining of machine learning models," 2020.

[23] H. Jia, M. Yaghini, C. A. Choquette-Choo, N. Dullerud, A. Thudi, V. Chandrasekaran, and N. Papernot, "Proof-of-learning: Definitions and practice," *arXiv preprint arXiv:2103.05633*, 2021.

[24] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy." *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[25] G. B. Folland, *Real analysis: modern techniques and their applications*. John Wiley & Sons, 1999, vol. 40.

[26] B. W. Jimmy Ba, "Csc421/2516 lecture 3: Automatic differentiation & distributed representations," Lecture Notes, WS 2021.

[27] L. Song and P. Mittal, "Systematic evaluation of privacy risks of machine learning models," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[28] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy risk in machine learning: Analyzing the connection to overfitting," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 2018, pp. 268–282.

[29] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," *Master's Thesis*, 2009.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[32] B. A. Pearlmutter, "Fast exact multiplication by the hessian," *Neural computation*, vol. 6, no. 1, pp. 147–160, 1994.

[33] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[34] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 2011, pp. 142–150.

Appendix

1. Proofs

Lemma 1

We have:

$$
\begin{aligned}
&||\mathbb{P}(\mathbf{w}) - \mathbb{P}'(\mathbf{w})||_2 = \\
&||\mathbb{P}(\mathbf{w}) - \mathbb{P}'(\mathbf{w}) + \frac{(n-1)!^m}{n!^m}(n^m\mathbb{P}'(\mathbf{w}) - n^m\mathbb{P}'(\mathbf{w}))||_2 \\
&\leq ||\frac{1}{n!^m}\sum_I \mathbb{P}_I(\mathbf{w}) - \mathbb{P}'_{I'}(\mathbf{w})||_2 + |(\frac{(n-1)!^m n^m}{n!^m} - 1)|\mathbb{P}'(\mathbf{w}) \\
&\leq \frac{1}{n!^m}\sum_I ||\mathbb{P}_I(\mathbf{w}) - \mathbb{P}_I(\mathbf{w} - \mathbf{d}_I)||_2 \qquad (13)
\end{aligned}
$$

Where the step from the first line to the second line follows from noting that by having n^m copies of $(n-1)!^m\mathbb{P}'(\mathbf{w}) = \sum_{I'} \mathbb{P}'_{I'}(\mathbf{w})$ we can associate to each $\mathbb{P}_I(\mathbf{w})$ it's corresponding $\mathbb{P}'_{I'}(\mathbf{w})$, which comes from the previous remark on counting of I and I'. If we assume every $\mathbb{P}_I$ is lipschitz (which is true for Gaussian noise), then $||\mathbb{P}_I(\mathbf{w}) - \mathbb{P}_I(\mathbf{w} - \mathbf{d}_I)||_2 \leq L_I||\mathbf{d}_I||_2$; note by definition all $\mathbb{P}_I$ are just the same distributions but shifted (as we have the same noise $\mathbf{g}$ for each plus some varying constant $\mathbf{w}_I$), thus all the L_I are equal, i.e., $L_I = L$ for every I (as translating a function doesn't change its lipschitz constant). Lastly if we let d be the average of all the $\mathbf{d}_I$, that is $d = \frac{1}{n!^m} \sum_I ||\mathbf{d}_I||_2$, we have

$$||\mathbb{P}(\mathbf{w}) - \mathbb{P}'(\mathbf{w})||_2 \leq Ld \tag{14}$$

Corollary 1

To understand how this relates to verification error and approximate unlearning, let us consider an approximate unlearning method which only looks at $\mathbf{w}_0$ and I and to $\mathbf{w}_I$ obtains the approximately unlearned weights $\mathbf{w}''_I = \mathbf{w}_I + u_I$ where u_I is some unlearning update. Note then we can analogously define a $\mathbf{z}''_I = \mathbf{w}_I + \mathbf{u}_I + \mathbf{g}$ as before with corresponding density function $\mathbb{P}''_I(w)$. Defining $\mathbf{v}_I = \mathbf{u}_I + \mathbf{d}_I$ we get $\mathbf{z}_{I'} = \mathbf{z}''_I - \mathbf{v}_I$ where $\mathbf{v}_I$ represents the analytic (ignoring noise) verification error for the given ordering I (i.e., the difference between the weights obtained from the approximate unlearning $\mathbf{w}''_I$ and the retrained weights $\mathbf{w}_{I'}$). At this point it is clear all the previous steps follow and defining v as the average of all the $||\mathbf{v}_I||_2$ we get

$$||\mathbb{P}''(\mathbf{w}) - \mathbb{P}'(\mathbf{w})||_2 \leq Lv \tag{15}$$

where now $\mathbb{P}''(\mathbf{w}) = \frac{1}{n!^m} \sum_I \mathbb{P}''_I$ represents the probability density function of the weights after applying the approximate unlearning method on M to obtain M''. **Reverse Direction of Corollary 1** We now proceed to get a bound on the average verification error v based on a uniform bound between the density function after the approximate unlearning and the density function for ideal retraining.

Note, assuming the noise has 0 mean, v is simply the difference in expectation of $\mathbb{P}''(\mathbf{w})$ and $\mathbb{P}'(\mathbf{w})$, and let us for the time being assume that the union of the support of $\mathbb{P}''(\mathbf{w})$ and $\mathbb{P}'(\mathbf{w})$ is bounded, i.e., if $\mathbf{W}$ is the union of the supports, then $\int_{\mathbf{W}} ||\mathbf{w}||_2 d\mathbf{w} = a < \infty$. This is reasonable as we would expect the noise from training to be bounded; in fact in general we would only need the integral outside a bounded domain to be finite, but the reasoning is analogous to what follows and would simply add an extra constant term. Now if say $||\mathbb{P}''(\mathbf{w}) - \mathbb{P}'(\mathbf{w})||_2 < b$ for some scalar b for all w (i.e., b is a uniform bound), we have:

$$\begin{aligned} v &= ||\mathbb{E}(\mathbb{P}''(\mathbf{w})) - \mathbb{E}(\mathbb{P}'(\mathbf{w}))||_2 \\ &= ||\int_{\mathbf{W}} \mathbb{P}''(\mathbf{w})\mathbf{w}d\mathbf{w} - \int_{\mathbf{W}} \mathbb{P}'(\mathbf{w})\mathbf{w}d\mathbf{w}||_2 \\ &\leq \int_{\mathbf{W}} ||\mathbb{P}''(\mathbf{w}) - \mathbb{P}'(\mathbf{w})||_2 \cdot ||\mathbf{w}||_2 d\mathbf{w} \\ &\leq ba \end{aligned} \tag{16}$$

and so in this sense we see reducing the max value of $||\mathbb{P}(\mathbf{w}) - \mathbb{P}'(\mathbf{w})||_2$ gives a smaller bound on the expectation of verification error v.

2. Additional Figures

In Figure 16, the change in weights and singular values for varying strengths of the ℓ_2 regularization is shown. Figure 15 and 14 are identical setups to Figure 9 and 12 but using amnesiac unlearning to produce the model M''.

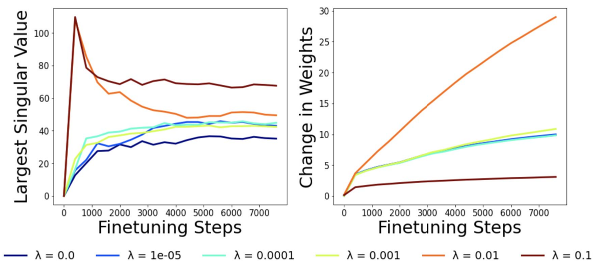


Figure 16. Change in weights and singular values during training for different strengths of ℓ_2 regularization for ResNet-18 trained on CIFAR-10

3. Additional Tables

In Table 3, the unlearning error and test accuracy for the 4 setups as a function of the ℓ_2 regularization strength and pretraining amount (in epochs) are shown. Table 4 shows a similiar experiment but with the proposed SD regularization.

TABLE 3. Unlearning error and testing accuracy for 4 different settings for varying regularization strengths of the ℓ_2 regularizer and pretraining amount compared to zero regularization.

Pretrain Epochs	Regularization	ResNet-18 CIFAR-10		VGG-19 CIFAR-10		Regularization	ResNet-18 CIFAR-100		VGG-19 CIFAR-100	
	CIFAR-10	Unlearning Error	Accuracy (%)	Unlearning Error	Accuracy(%)	CIFAR-100	Unlearning Error	Accuracy(%)	Unlearning Error	Accuracy(%)
0	0.0	148.31	10.84	200.66	10.2	0.0	154.51	0.95	359.05	1.18
	1e-05	145.44	10.68	174.23	10.87	1e-05	171.82	0.94	256.09	1.12
	0.0001	164.48	8.78	165.01	10.53	0.0001	159.26	1.22	501.29	0.92
	0.001	613.22	9.65	199.99	9.85	0.001	161.26	0.85	547.47	0.98
	0.01	702.94	9.54	772.91	10.01	0.01	651.45	1.18	1162.51	1.0
	0.1	2092.77	10.43	3980.42	10.52	0.1	854.97	1.11	3211.49	1.08
20	0.0	161.09	76.61	108.75	79.14	0.0	182.06	45.71	214.51	51.71
	1e-05	174.62	75.98	116.3	79.98	1e-05	174.36	46.69	201.91	52.07
	0.0001	142.46	76.94	141.75	79.51	0.0001	191.54	46.53	206.77	51.78
	0.001	205.24	76.11	137.72	80.24	0.001	171.26	47.29	214.15	52.35
	0.01	421.03	67.73	346.64	79.93	0.01	458.96	49.53	361.27	55.18
	0.1	86.73	77.95	176.45	80.85	0.1	–	7.0	383.14	24.72
40	0.0	152.67	76.94	106.7	81.71	0.0	169.7	44.68	214.5	51.4
	1e-05	176.08	76.71	111.17	82.5	1e-05	177.19	45.36	221.92	51.74
	0.0001	162.47	76.13	–	–	0.0001	201.82	45.89	225.43	52.91
	0.001	159.88	76.18	122.83	82.48	0.001	235.74	46.37	228.98	53.58
	0.01	189.76	71.75	224.72	80.93	0.01	265.38	45.76	271.12	60.22
	0.1	101.32	81.28	182.46	75.73	0.1	361.16	5.4	290.84	20.5

TABLE 4. Unlearning error and testing accuracy for 4 different settings for varying regularization strengths of the Standard Deviation regularizer and pretraining amount. Note $t = 1$ epoch here.

Pretrain Epochs	Regularization	ResNet-18 CIFAR-10		VGG-19 CIFAR-10		Regularization	ResNet-18 CIFAR-100		VGG-19 CIFAR-100	
	CIFAR-10	Unlearning Error	Accuracy (%)	Unlearning Error	Accuracy (%)	CIFAR-100	Unlearning Error	Accuracy (%)	Unlearning Error	Accuracy (%)
0	0.0	141.35	8.82	160.95	9.03	0.0	165.65	0.78	245.66	1.09
	1.0	110.32	9.66	348.85	10.51	50.0	142.59	1.01	381.88	1.38
	5.0	102.71	9.82	139.47	9.98	100.0	127.31	0.99	288.04	1.0
	10.0	90.66	10.06	203.44	10.15	150.0	110.48	0.95	186.17	1.0
	15.0	44.85	10.61	156.59	10.07	200.0	58.59	0.74	48.08	0.85
	20.0	21.63	10.32	69.34	9.5	250.0	30.81	0.85	34.56	1.21
10	0.0	132.49	77.11	141.73	77.69	0.0	251.12	44.07	258.63	48.39
	1.0	101.79	74.5	123.17	76.81	50.0	176.35	43.52	233.65	50.08
	5.0	98.3	73.21	113.27	77.16	100.0	177.28	43.12	212.71	47.49
	10.0	90.71	69.13	107.9	74.07	150.0	171.77	38.5	216.0	46.97
	15.0	86.17	68.46	78.82	65.32	200.0	124.74	26.68	195.89	36.75
	20.0	11.5	34.74	27.94	31.18	250.0	34.53	13.76	59.9	8.98
20	0.0	153.78	76.34	120.19	79.05	0.0	194.65	45.79	260.59	52.27
	1.0	109.07	74.14	135.44	80.93	50.0	140.1	45.77	220.46	52.55
	5.0	90.61	74.01	93.58	79.97	100.0	165.91	45.45	208.58	53.12
	10.0	86.31	69.88	100.09	78.21	150.0	181.78	44.89	233.99	52.42
	15.0	86.67	68.69	95.62	73.81	200.0	184.81	41.18	241.61	47.34
	20.0	10.74	36.29	32.23	38.88	250.0	116.28	27.7	131.24	23.15
30	0.0	153.26	76.48	113.37	81.5	0.0	210.35	45.68	250.87	52.17
	1.0	112.01	73.7	130.25	81.18	50.0	151.56	45.02	194.86	52.69
	5.0	94.44	72.48	108.84	80.24	100.0	178.74	45.57	229.36	52.88
	10.0	83.53	70.19	116.96	78.76	150.0	205.75	42.71	217.5	52.48
	15.0	88.96	69.17	92.93	77.13	200.0	203.18	43.1	256.51	51.81
	20.0	13.73	36.98	49.03	48.69	250.0	127.97	28.52	200.4	37.02
40	0.0	175.32	75.59	120.92	82.07	0.0	209.52	45.11	219.14	52.32
	1.0	97.11	74.86	109.44	80.89	50.0	176.31	45.45	250.64	52.36
	5.0	93.74	73.58	108.54	80.21	100.0	201.74	45.01	251.3	53.01
	10.0	83.11	70.33	92.38	79.31	150.0	211.01	42.12	233.39	52.49
	15.0	79.86	68.7	96.14	77.64	200.0	218.61	41.58	233.58	50.67
	20.0	8.24	41.02	58.35	44.14	250.0	198.57	41.05	253.88	45.69
50	0.0	184.99	76.89	118.56	81.39	0.0	235.23	45.51	221.19	52.47
	1.0	100.39	74.51	121.06	81.37	50.0	168.65	44.91	226.56	51.95
	5.0	88.43	73.57	114.11	80.31	100.0	206.35	44.57	224.88	52.67
	10.0	80.72	72.14	113.41	79.28	150.0	229.16	43.73	230.4	53.57
	15.0	76.42	71.39	87.12	77.17	200.0	234.73	43.04	255.56	51.29
	20.0	17.43	37.31	77.01	68.4	250.0	228.22	40.7	254.52	47.52