Machine Unlearning: A Pursuit for Privacy in Deep Learning Models
Final Project Report

Yuliia Suprun
Department of Computer Science, Rice University
ys70@rice.edu

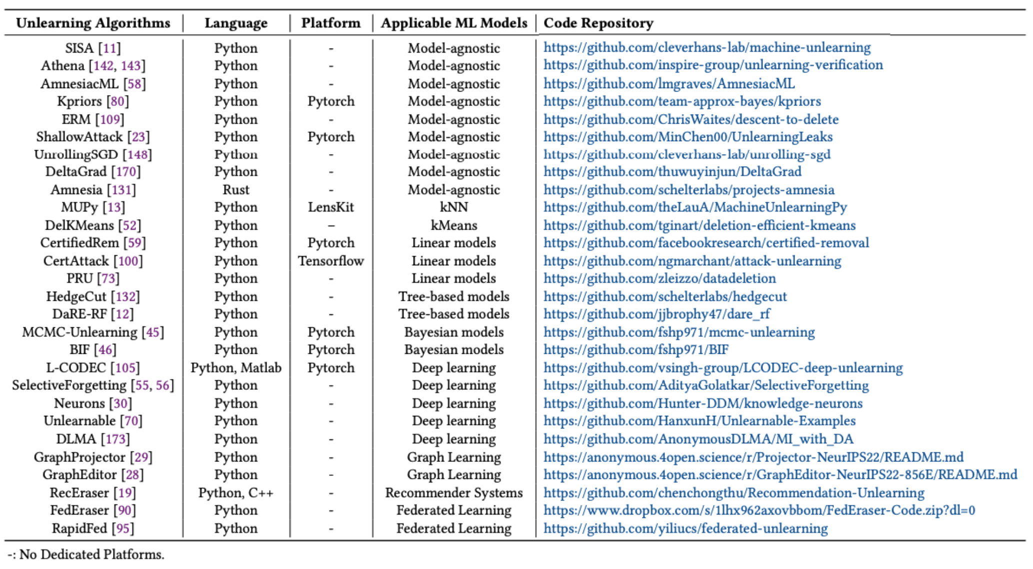
1. Introduction

Machine learning models are extensively used in critical domains, such as healthcare, for predicting disease progression based on patients' medical records. However, these models also carry significant risks, particularly in terms of privacy breaches. Unfortunately, merely removing sensitive data from databases is insufficient, as machine learning models often 'remember' old data. For instance, unauthorized parties could potentially infer the presence of sensitive information within these models, like medical records of high-profile individuals, through techniques like Membership Inference Attacks (MIA) [1, 9]. MIA aims to determine whether a specific data point was used in training a model, which could lead to serious personal and legal consequences. This memorization behavior poses a significant challenge in making models forget specific data, leading to the emergence of the machine unlearning paradigm [8] - a process designed to ensure the complete and irretrievable removal of designated data from a model, rendering it computationally infeasible to infer its prior existence in the training set, while preserving the model's performance. This concept has attracted significant attention, as evidenced by its selection for the NeurIPS'23 competition track [4], focusing on an age predictor model trained on facial images.

2. Related Work

A comprehensive analysis of machine unlearning methods is provided in "A Survey of Machine Unlearning" [8]. This survey categorizes various unlearning models based on the types of machine learning models they are applicable to and includes references to their code repositories. Notably, many of these algorithms are model-agnostic, allowing for broad applicability across different machine learning models.

Table 4: Published Algorithms and Models



Unlearning Algorithms	Language	Platform	Applicable ML Models	Code Repository
SISA [11]	Python	-	Model-agnostic	https://github.com/cleverhans-lab/machine-unlearning
Athena [142, 143]	Python	-	Model-agnostic	https://github.com/inspire-group/unlearning-verification
AmnesiacML [58]	Python	-	Model-agnostic	https://github.com/lmgraves/AmnesiacML
Kpriors [80]	Python	Pytorch	Model-agnostic	https://github.com/team-approx-bayes/kpriors
ERM [109]	Python	-	Model-agnostic	https://github.com/ChrisWaites/descent-to-delete
ShallowAttack [23]	Python	Pytorch	Model-agnostic	https://github.com/MinChen00/UnlearningLeaks
UnrollingSGD [148]	Python	-	Model-agnostic	https://github.com/cleverhans-lab/unrolling-sgd
DeltaGrad [170]	Python	-	Model-agnostic	https://github.com/thuwuyinjun/DeltaGrad
Amnesia [131]	Rust	-	Model-agnostic	https://github.com/schelterlabs/projects-amnesia
MUPy [13]	Python	LensKit	kNN	https://github.com/theLauA/MachineUnlearningPy
DelKMeans [52]	Python	–	kMeans	https://github.com/tginart/deletion-efficient-kmeans
CertifiedRem [59]	Python	Pytorch	Linear models	https://github.com/facebookresearch/certified-removal
CertAttack [100]	Python	Tensorflow	Linear models	https://github.com/ngmarchant/attack-unlearning
PRU [73]	Python	-	Linear models	https://github.com/zleizzo/datadeletion
HedgeCut [132]	Python	-	Tree-based models	https://github.com/schelterlabs/hedgecut
DaRE-RF [12]	Python	-	Tree-based models	https://github.com/jjbrophy47/dare_rf
MCMC-Unlearning [45]	Python	Pytorch	Bayesian models	https://github.com/fshp971/mcmc-unlearning
BIF [46]	Python	Pytorch	Bayesian models	https://github.com/fshp971/BIF
L-CODEC [105]	Python, Matlab	Pytorch	Deep learning	https://github.com/vsingh-group/LCODEC-deep-unlearning
SelectiveForgetting [55, 56]	Python	-	Deep learning	https://github.com/AdityaGolatkar/SelectiveForgetting
Neurons [30]	Python	-	Deep learning	https://github.com/Hunter-DDM/knowledge-neurons
Unlearnable [70]	Python	-	Deep learning	https://github.com/HanxunH/Unlearnable-Examples
DLMA [173]	Python	-	Deep learning	https://github.com/AnonymousDLMA/MI_with_DA
GraphProjector [29]	Python	-	Graph Learning	https://anonymous.4open.science/r/Projector-NeurIPS22/README.md
GraphEditor [28]	Python	-	Graph Learning	https://anonymous.4open.science/r/GraphEditor-NeurIPS22-856E/README.md
RecEraser [19]	Python, C++	-	Recommender Systems	https://github.com/chenchongthu/Recommendation-Unlearning
FedEraser [90]	Python	-	Federated Learning	https://www.dropbox.com/s/1lhx962axovbbom/FedEraser-Code.zip?dl=0
RapidFed [95]	Python	-	Federated Learning	https://github.com/yiliucs/federated-unlearning

-: No Dedicated Platforms.

2.1. Retraining-free methods

Retraining-free unlearning methods offer the advantage of modifying a trained model without the need for complete retraining. One common approach involves the use of the Fisher Information Matrix (FIM), which approximates a model's output sensitivity to parameter perturbations. For example, Fisher Forgetting uses FIM to induce forgetting by injecting noise into parameters based on their importance to the forget set [6]. However, this approach can be computationally expensive and may degrade the model's accuracy on retained data. Selective Synaptic Dampening (SSD) offers a faster and more efficient method that selectively weakens crucial parameters for the forget set while preserving overall model accuracy [5].

2.2. Retraining-based methods

Retraining-based methods are considered the state-of-the-art in machine unlearning due to their effectiveness in balancing data removal and model performance. These methods often involve training a model on a modified dataset where the data to be forgotten is either removed or altered. A prominent example is the Competent/Incompetent Teachers Framework, which employs a student-teacher model to selectively transfer knowledge, ensuring the removal of specific data without complete retraining [3]. Another approach is the amnesiac unlearning method, which relabels the forget set with incorrect labels and retrains the network to induce forgetting while maintaining performance on the retained data [7].

3. Preliminaries

3.1. Differential Privacy

Differential Privacy (DP) plays a pivotal role in the context of machine unlearning, providing a framework to ensure the confidentiality of individual data within datasets.

Definition 2.1. Example-level Differential Privacy: A training algorithm $A : \mathcal{D} \to \mathcal{R}$ is (ϵ, δ) example-level DP if, for all pairs of datasets D and D' from $\mathcal{D}$ that differ by the addition or removal of any single training example, and for all output regions $R \subseteq \mathcal{R}$, the following holds:

$$\text{Pr}[A(D) \in R] \leq e^{\epsilon}\text{Pr}[A(D') \in R] + \delta.$$

In this setting, the output space $\mathcal{R}$ could be, for example, the d-dimensional Euclidean space for a neural network with d parameters. DP can be viewed as a hypothesis test with the null hypothesis that A was trained on D and the alternative hypothesis that A was trained on D'. False positives (type-I errors) occur when the null hypothesis is true, but is rejected, while false negatives (type-II errors) occur when the alternative hypothesis is true, but is rejected.

The privacy parameter ϵ can be estimated under a fixed δ as:

$$\hat{\epsilon} = \max\left(\log\frac{1-\delta-\text{FPR}}{\text{FNR}}, \log\frac{1-\delta-\text{FNR}}{\text{FPR}}\right),$$

where FPR and FNR are the estimates of the true FPR and FNR under a membership inference attack that examines the DP-trained model. Such attacks can be both black-box and white-box, attempting to infer whether the model was trained on D or D' [9].

Definition 2.2. Group-level Differential Privacy: Extending the concept to group-level, a training algorithm $A : \mathcal{D} \to \mathcal{R}$ is (ϵ, δ, k) group-level DP if, for all pairs of datasets D and D' from $\mathcal{D}$ that differ by the addition or removal of up to k training examples, and for all output regions $R \subseteq \mathcal{R}$, the following holds:

$$\text{Pr}[A(D) \in R] \leq e^{\epsilon}\text{Pr}[A(D') \in R] + \delta.$$

Group-level DP differs from example-level DP in that it accounts for the addition or removal of up to k arbitrary training examples, thus providing a stronger privacy guarantee. Thus, group-level DP can be stronger than example-level DP.

3.2. Machine Unlearning

The concept of machine unlearning can be defined using a framework similar to that of Differential Privacy (DP).

Firstly, we introduce the notion of a *forget set* $S \subseteq D$, which consists of the training examples that the model trained on dataset D should "forget". An unlearning algorithm $U(\cdot)$ takes as input the model trained on D using algorithm A, the forget set S, and potentially the dataset D, and outputs a model that has "forgotten" S. The effectiveness of forgetting is measured relative to the model retrained from scratch on $D \setminus S$, denoted as $A(D \setminus S)$. This leads us to the following formal definition:

Definition 3.1. Machine Unlearning: Given a fixed dataset D, a forget set $S \subseteq D$, and a randomized learning algorithm A, an unlearning algorithm U is (ϵ, δ)-unlearning with respect to (D, S, A) if for all regions $R \subseteq \mathcal{R}$, the following inequalities hold:

$$\Pr[A(D \setminus S) \in R] \leq e^{\epsilon}\Pr[U(A(D), S, D) \in R] + \delta,$$

and

$$\Pr[U(A(D), S, D) \in R] \leq e^{\epsilon}\Pr[A(D \setminus S) \in R] + \delta.$$

Intuitively, when ϵ and δ are very small, the above definition implies that the distributions of $A(D \setminus S)$ and $U(A(D), S, D)$ are nearly indistinguishable from one another.

As with DP, the ϵ privacy parameter at a fixed δ can be estimated using the equation defined above, employing estimates of FPR and FNR. In the Kaggle competition, empirical estimates of FPR and FNR were used for efficiency to evaluate the submission scores.

3.3. Forgetting Quality

We now discuss how we can use the above definition of unlearning to define an evaluation metric for unlearning algorithms. The objective is to measure the difference between two distributions: one obtained by the ”oracle” unlearning algorithm, which retrains from scratch without the forget set S, denoted as $A(D \setminus S)$, and the other obtained by an unlearning algorithm U, denoted as $U(A(D), S, D)$.

Given that both $A(D \setminus S)$ and $U(A(D), S, D)$ are distributions in weight space, influenced by initialization randomness and data ordering in mini-batches, the task is to find a measure of their discrepancy. The Kaggle competition considers black-box attacks based on outputs produced by models when processing the forget set S. These attacks yield estimates for the False Positive Rate (FPR) and False Negative Rate (FNR) for each example in the forget set. The effectiveness of an unlearning algorithm is inversely proportional to the performance of the best attack against it; the better the attack at distinguishing between $\mathcal{R}^s$ and $\mathcal{U}^s$, the less effective the unlearning algorithm is. This leads to an estimation of the ϵ value for the specific forget set example s by applying the FPR and FNR from the most effective attack using the following equation:

$$\epsilon_s = \max\left(\log\frac{1-\delta-\text{FPR}}{\text{FNR}}, \log\frac{1-\delta-\text{FNR}}{\text{FPR}}\right), \tag{1}$$

where δ is a predetermined value reflecting the acceptable probability of error in distinguishing between the two distributions. These ϵ values are then aggregated to compute the overall ”forgetting quality.”

Let $\mathcal{R}$ and $\mathcal{U}$ be the distributions of retrained and unlearned models, respectively. For a particular forget set example $s \in S$, let $\mathcal{R}^s$ and $\mathcal{U}^s$ denote the distributions of (scalar) outputs of retrained and unlearned models, respectively, when receiving s as input. The Kaggle competition estimates $\mathcal{R}^s$ and $\mathcal{U}^s$ empirically by running retraining and unlearning N times each (with different random seeds) and compute two sets: $\mathcal{R}^s = \{f(R_1(s)), ..., f(R_N(s))\}$ and $\mathcal{U}^s = \{f(U_1(s)), ..., f(U_N(s))\}$, where R_i and U_i denote the i^{th} retrained and the i^{th} unlearned model, respectively, $M(s)$ yields the outputs obtained by feeding example s into model M.

The function f is critical in this process as it translates model outputs into a scalar value that can be used to estimate the distributions $\mathcal{R}^s$ and $\mathcal{U}^s$ for any given example s from the forget set. In my experimentation, I employed a function, defined as *compute_forgetting_quality*, which takes an array of ϵ values representing the privacy degrees of forget set examples and computes their associated ”forgetting quality” scores. The function uses a binning mechanism to categorize ϵ values into discrete levels, with smaller ϵ values indicating better unlearning performance.

The Python function is provided below for clarity:

```
def compute_forgetting_quality(epsilons: np.ndarray) -> float:
    """Computes the forgetting quality given the privacy degrees
    of the forget set examples."""
    ns = bin_index_fn(epsilons)  # Maps epsilon values to bin indices
    hs = 2. / 2 ** ns  # Assigns scores based on bin indices
    return np.mean(hs)  # Averages the scores to obtain the final metric
```

The binning approach, implemented in the function *bin_index_fn*, groups ϵ values into predefined ranges, each of which corresponds to a specific level of unlearning difficulty. I used 13 bins. The score assignment, $\frac{2}{2^{ns}}$, then inversely relates the difficulty level to a score, so that a lower ϵ value results in a higher score. The use of the exponential function in the score calculation ensures that small improvements in ϵ at the lower end (which are harder to achieve) are rewarded more than equivalent improvements at the higher end.

The final forgetting quality metric is the mean of the scores across all examples. It is worth noting that for the purposes of the Kaggle competition, a larger number of models, $N = 512$, was used to obtain a robust estimate of ϵ values. In contrast, my own testing procedure used $N = 10$ to accelerate the process.

The pseudocode provided in Algorithm 1 outlines the computation of ϵ for a specific example. Figure 1 demonstrates the relationship between FNR, FPR, and the hypothesis test defining a decision boundary based on $f(\cdot)$.

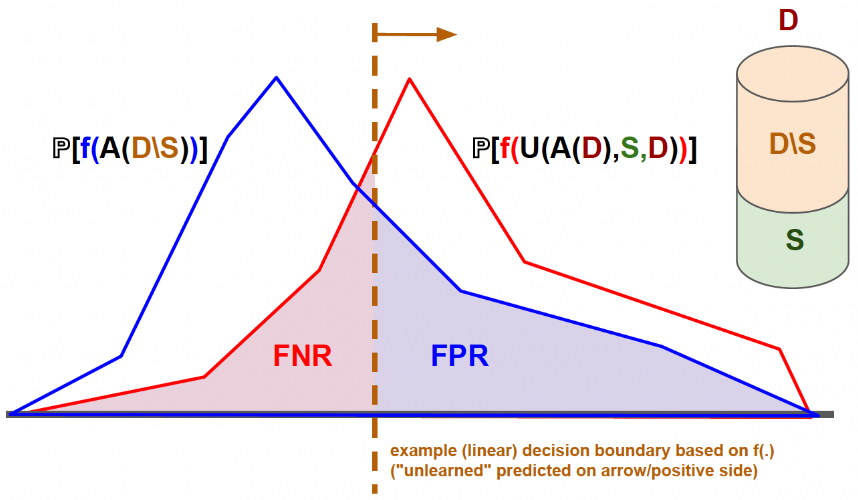

D
ℙ[f(A(D\S))]
ℙ[f(U(A(D),S,D))]
D\S
S
FNR
FPR
example (linear) decision boundary based on f(.)
("unlearned" predicted on arrow/positive side)

Figure 1. An illustration of FNR and FPR for a hypothesis test based on an example (linear) decision boundary and summary statistic, $f(\cdot)$. The densities represent the summary statistic computed on $A(D \setminus S)$ and $U(A(D), S, D)$. The level of unlearning (ϵ, δ) achieved by the unlearning procedure can be estimated from the FNR and FPR of an optimal hypothesis test via Definition 3.1.

Algorithm 1 Compute an example's ϵ from FPRs and FNRs of multiple attacks

```
Require: FPR, FNR: two lists of length $m$, storing the false positive and false negative rates (respectively) from running a collection of $m$ attacks on the outputs of $N$ unlearned and $N$ retrained models, for a specific forget example.
Require: $nan\text{-}max$: a function that takes as input a list and returns the max of its elements, discarding any that are $nan$.
Require: $\delta$: a float.
1: $i \leftarrow 0$
2: $per\text{-}attack\text{-}\epsilon \leftarrow []$
3: while $i < m$ do
4:     if $FPR[i] = 0$ and $FNR[i] = 0$ then
5:         $per\text{-}attack\text{-}\epsilon \leftarrow per\text{-}attack\text{-}\epsilon + [\infty]$    ▷ This attack perfectly separates the two distributions.
6:     else if $FPR[i] = 0$ or $FNR[i] = 0$ then
7:         pass    ▷ Discard attack if exactly one of $FPR[i]$ or $FNR[i]$ is 0.
8:     else
9:         $per\text{-}attack\text{-}\epsilon_1 \leftarrow \log(1 - \delta - FPR[i]) - \log(FNR[i])$    ▷ Compute this attack's $\epsilon$ via Equation 1
10:        $per\text{-}attack\text{-}\epsilon_2 \leftarrow \log(1 - \delta - FNR[i]) - \log(FPR[i])$
11:        $per\text{-}attack\text{-}\epsilon \leftarrow per\text{-}attack\text{-}\epsilon + [nan\text{-}max([per\text{-}attack\text{-}\epsilon_1, per\text{-}attack\text{-}\epsilon_2])]$
12:    end if
13:    $i \leftarrow i + 1$
14: end while
15: $\epsilon \leftarrow nan\text{-}max(per\text{-}attack\text{-}\epsilon)$    ▷ The $\epsilon$ for this example is that of the strongest attack
16: return $\epsilon$
```

4. Methods

This section details the various strategies and techniques that I tried to employ for this Kaggle competition.

4.1. Fine-tuning on the Retain Set

The fine-tuning approach focuses on additional training of the model using only the retain set. The intent is to reinforce the model's learning on the retained data, potentially causing the forget set data to be "overwritten." This baseline solution was originally proposed by the organizers of the Kaggle competition.

4.2. Advanced NegGrad

The Advanced NegGrad method employs a custom loss function, which imposes a penalty on the model when it correctly classifies examples from the forget set and conversely rewards the model for accurate predictions on the retain set. Specifically, it manipulates the gradient update rules to ascend for the forget set (encouraging misclassification) and to descend for the retain set (encouraging correct classification). This method was proposed in the work of Choi et al. [2]. However, I modified it a bit by adding a hyper-parameter λ, which helps adjust the "forgetting power" of this method.

The loss function can be mathematically represented as follows:

$$\text{JointLoss} = \text{RetainLoss} - \lambda \times \text{ForgetLoss}, \tag{2}$$

where λ is a hyperparameter that controls the trade-off between the retain and forget losses. In the context of the MUFAC dataset, empirical analysis suggested that setting $\lambda = 0.1$ yields optimal results, balancing the model's ability to forget the targeted data while maintaining its accuracy on the retained data.

4.3. Unlearning with Competent/Incompetent Teachers [3]

The competent/incompetent teacher framework addresses the unlearning task by leveraging the dynamics between two teachers — a competent one (denoted as T_s) and an incompetent one (denoted as T_d) — and a student model (denoted as S), as depicted in Figure 2 The student model begins with complete knowledge encapsulated within the parameters of the fully trained model. The objective is to selectively erase the student's knowledge of the forget set D_f, while preserving its understanding of the retain set D_r.

The process involves the incompetent teacher T_d imparting incorrect information about the forget set to the student, thereby encouraging the student to unlearn these data points. Conversely, the competent teacher T_s ensures that the student retains accurate knowledge pertaining to D_r.

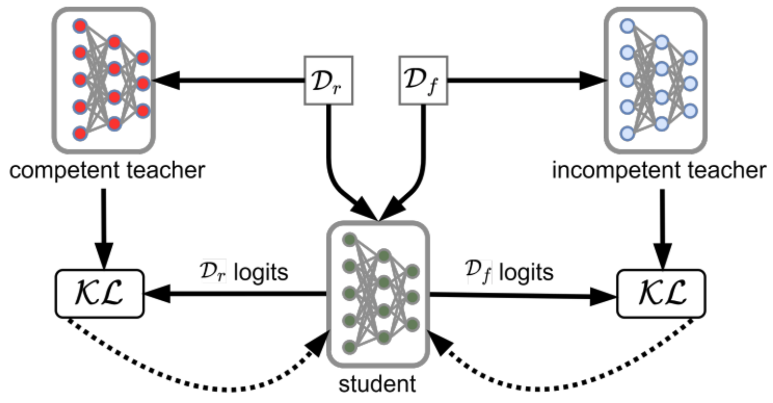


Figure 2. The proposed competent and incompetent teachers based framework for unlearning.

For a given data point x, the Kullback-Leibler Divergence between the dumb teacher T_d and the student S is defined as follows:

$$KL(T_d(x) \parallel S(x)) = \sum_i t_d^{(i)} \log\left(\frac{t_d^{(i)}}{s^{(i)}}\right), \tag{3}$$

where i indexes over data classes. Similarly, the KL Divergence between the competent teacher T_s and the student S is given by:

$$KL(T_s(x) \parallel S(x)) = \sum_i t_s^{(i)} \log\left(\frac{t_s^{(i)}}{s^{(i)}}\right). \tag{4}$$

The unlearning loss function L is formulated to combine these divergences, allowing the student to unlearn D_f while retaining D_r:

$$L(x, l_u) = (1 - l_u) \cdot KL(T_s(x) \parallel S(x)) + l_u \cdot KL(T_d(x) \parallel S(x)), \tag{5}$$

where l_u is the unlearning label associated with data sample x.

The proposed method operates on all samples from D_f and a subset of D_r, training the student to optimize the loss function L. This optimization transfers the "bad" knowledge about the forget data from T_d by minimizing the KL Divergence between S and T_d, while the accurate information related to D_r is transferred from T_s by minimizing the KL Divergence between S and T_s. As a result, the student model learns to replicate T_d for D_f, effectively unlearning the forget samples, while maintaining the general knowledge about retain samples.

4.4. Selective Synaptic Dampening (SSD) [5]

Selective Synaptic Dampening (SSD) is predicated on the assumption that certain parameters within a neural network are disproportionately influential for the forget set D_f, but not for the retain set D_r. This hypothesis is supported by research suggesting that neural networks memorize specific examples and that parameters, particularly in deeper layers, are often overfitted. Therefore, parameters critical only to D_f can be modified without significantly impacting the model's performance on D_r.

Hessian and the Fisher Information Matrix The Fisher Information Matrix (FIM) provides a mechanism to identify these critical parameters [6]. Assuming the optimal parameters θ^* have been found that minimize the loss over D, the sensitivity of the model with respect to each parameter θ_k can be approximated by the diagonal of the FIM, which corresponds to the second-order derivative of the loss near the minimum. The FIM is defined as follows:

$$\mathbf{I}_D = \mathbb{E}\left[-\frac{\delta^2 \ln p(D|\theta)}{\delta\theta^2} \bigg|_{\theta^*} \right] = \mathbb{E}\left[\left(\frac{\delta \ln p(D|\theta)}{\delta\theta} \frac{\delta \ln p(D|\theta)}{\delta\theta} \right)^T \bigg|_{\theta^*} \right], \tag{6}$$

where $\mathbf{I}_D$ represents the FIM calculated over the dataset D.

Naïve Forgetting with FIM A straightforward approach to unlearning is to prune parameters based on their importance as calculated by the FIM. The pruning rule is described by:

$$\theta_i = \begin{cases} 0, & \text{if } \mathbf{I}_{D_f,i} > 0 \\ \theta_i, & \text{if } \mathbf{I}_{D_f,i} = 0 \end{cases} \quad \forall i \in [0, |\theta|], \tag{7}$$

where $\mathbf{I}_{D_f,i}$ is the i-th element of the diagonal of the FIM for D_f. This method, however, is too blunt as it fails to distinguish between parameters that are slightly or significantly more important for D_f, leading to potential degradation on D_r.

Refined Selective Dampening To address this limitation, SSD employs a selection criterion that considers parameter importance relative to D_r:

$$\theta_i = \begin{cases} 0, & \text{if } \mathbf{I}_{D_f,i} > \alpha \mathbf{I}_{D_r,i} \\ \theta_i, & \text{if } \mathbf{I}_{D_f,i} \leq \alpha \mathbf{I}_{D_r,i} \end{cases} \quad \forall i \in [0, |\theta|], \tag{8}$$

where α is a hyper-parameter tuning the threshold for parameter selection.

To introduce granularity to the unlearning process, SSD also replaces the binary pruning with a dampening step. Parameters are scaled according to their relative importance, leading to a more nuanced forgetting:

$$\beta = \min\left(\lambda \frac{\mathbf{I}_{D,i}}{\mathbf{I}_{D_f,i}}\theta_i, 1\right), \quad \theta_i = \begin{cases} \beta\theta_i, & \text{if } \mathbf{I}_{D_f,i} > \alpha \mathbf{I}_{D,i} \\ \theta_i, & \text{if } \mathbf{I}_{D_f,i} \leq \alpha \mathbf{I}_{D,i} \end{cases} \quad \forall i \in [0, |\theta|], \tag{9}$$

Modifications for Single Forget Requests It is important to note that while the original authors utilized $\mathbf{I}_{D,i}$ (the importance relative to the entire training dataset D) in their approach to be able to reuse this value for multiple forget requests, I decided to substitute it with $\mathbf{I}_{D_r,i}$ (the importance relative to the retain set D_r) because of the specific requirement of the Kaggle competition, which focuses on executing a single forget request. The rationale behind this choice is that $\mathbf{I}_{D_r,i}$ offers a more targeted measure of parameter significance for the subset of data we aim to preserve, thus enhancing the accuracy of the unlearning process.

$$\beta = \min\left(\lambda \frac{\mathbf{I}_{D_r,i}}{\mathbf{I}_{D_f,i}}\theta_i, 1\right), \quad \theta_i = \begin{cases} \beta\theta_i, & \text{if } \mathbf{I}_{D_f,i} > \alpha \mathbf{I}_{D_r,i} \\ \theta_i, & \text{if } \mathbf{I}_{D_f,i} \leq \alpha \mathbf{I}_{D_r,i} \end{cases} \quad \forall i \in [0, |\theta|], \tag{10}$$

Algorithm 2 Selective Synaptic Dampening (modified for a single forget request)

```
Require: $\phi_0, D_r, D_f; \alpha, \lambda$
1: Calculate $\mathbf{I}_{D_r}$.
2: Calculate $\mathbf{I}_{D_f}$.
3: for $i$ in range $|\theta|$ do
4:     if $\mathbf{I}_{D_f,i} > \alpha \mathbf{I}_{D_r,i}$ then
5:         $\theta'_i = \min\left(\frac{\lambda \mathbf{I}_{D_r,i}}{\mathbf{I}_{D_f,i}}, 1\right)\theta_i$
6:     end if
7: end for
8: return $\phi'$
```

4.5. Gradient-Informed Synaptic Adjustment (GISA)

Gradient-Informed Synaptic Adjustment (GISA) is the main contribution of my project, which was developed to address the limitations of Selective Synaptic Dampening (SSD), particularly its high sensitivity to hyperparameters and the specific nature of datasets. While SSD effectively prunes parameters deterministically, it can result in suboptimal model performance, especially when dealing with complex datasets, as observed in the Kaggle competition. GISA, in contrast, introduces a dynamic mechanism to adjust the learning rate for each parameter based on their relative importance. It is inspired by second-order optimization methods like the Newton's method.

Learning Rate Scaling in GISA The essence of GISA is to adjust the learning rate of each parameter based on its relative importance to the forget and retain sets. The scaling is performed such that parameters crucial for the forget set but not for the retain set are updated with a dampened learning rate, thus facilitating selective forgetting:

$$\eta_i = \eta_{\text{base}} \cdot (1 + \text{mean}(\max(\log(\frac{\mathbf{I}_{D_f,i}}{\mathbf{I}_{D_r,i} + \epsilon}), 0))), \tag{11}$$

where η_i is the scaled learning rate for parameter i, η_{base} is the base learning rate, $\mathbf{I}_{D_f,i}$ and $\mathbf{I}_{D_r,i}$ are the importances for the forget and retain sets respectively, and ϵ is a small constant to ensure numerical stability.

Custom Loss Function GISA employs a custom loss function designed to prioritize retain set performance while mitigating the model's fit to the forget set. This loss function is formulated as:

$$\text{JointLoss} = \frac{\text{RetainLoss}^2}{\text{ForgetLoss} + \epsilon}, \tag{12}$$

where RetainLoss is the loss calculated on the retain set, ForgetLoss is the loss on the forget set, and ϵ is a small constant added for numerical stability.

Gradient Update Rule With learning rates adjusted for each parameter, the gradient update rule in GISA is expressed as:

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \eta_i \cdot \nabla_{\theta_i}\text{JointLoss}, \tag{13}$$

where $\theta_i^{(t)}$ is the parameter value at iteration t, η_i is the scaled learning rate for parameter i, and ∇_{θ_i}JointLoss is the gradient of the custom loss function with respect to θ_i.

5. Experimental Setup

This section describes the experimental framework used to evaluate the machine unlearning methodologies. This includes descriptions of the datasets, the pre-trained model provided by the competition, the evaluation procedures adopted, and the approach to training and hyper-parameter tuning.

5.1. Datasets

The Kaggle competition featured a hidden dataset that was only accessible during submission attempts, with a constraint of five submissions per day. It contained 30,000 images and participants were asked to forget 15 subjects (each subject could have multiple images in the training dataset). This hidden dataset was also highly imbalanced, posing an additional challenge for effective machine unlearning.

Therefore, I had to use alternative datasets for local testing. My initial experiments utilized the CIFAR10 dataset. However, its simplicity compared to the hidden dataset led to a disparity in the effectiveness of the strategies.

To mitigate this limitation and closely simulate the competition conditions, I transitioned to using the MUFAC (Machine Unlearning for Facial Age Classifier) dataset [2], obtained from AI HUB in South Korea. This dataset comprises over 13,000 images of Asian facial features, annotated with age group classifications and personal identity markers, thus providing a more challenging and representative test environment. The MUFAC dataset is organized into three segments:

- **Training dataset:** Folders F0001 to F0299, with 10,025 images.
- **Validation dataset:** Folders F0801 to F0850, including 1,539 images.
- **Test dataset:** Folders F0851 to F0900, totaling 1,504 images.

For local testing, 2% of the training dataset was designated as the forget set. Moreover, I ensured that no duplicates occurred in the retain and forget sets, which is crucial for a valid unlearning process. The sizes of the forget and retain datasets were as follows:

- **Forget dataset:** 200 images.
- **Retain dataset:** 9,825 images.

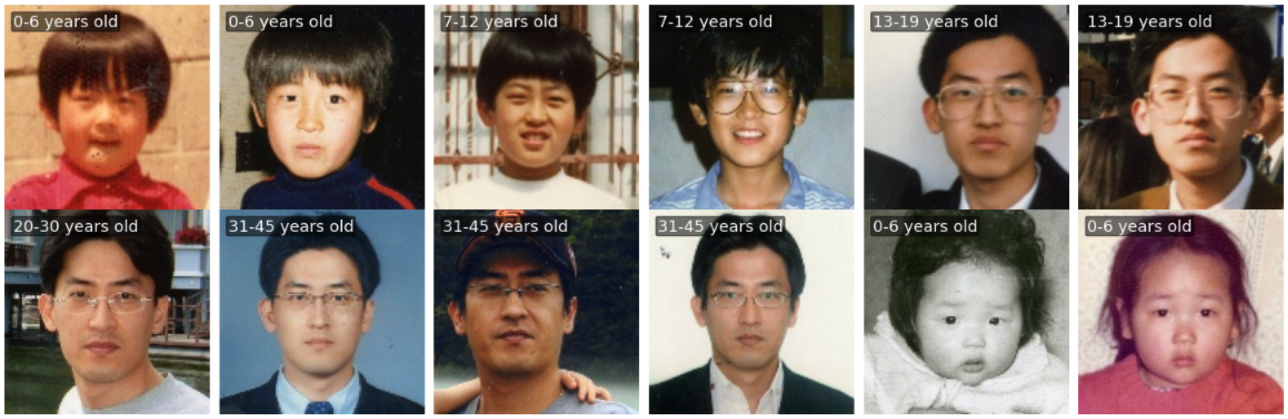


Figure 3. Sample images from the MUFAC dataset illustrating the various age categories.

Further insights into the experimentation with the MUFAC dataset can be found in the project notebook hosted on GitHub[1].

[1]`https://github.com/YuliiaSuprun/MachineUnlearning/blob/main/Local_Scoring_on_MUFAC.ipynb`

5.2. Original Model

The Kaggle competition provided Resnet-18 PyTorch model, which serves as the baseline model and should be forced to forget certain images. This original model is trained to predict age groups based on facial images and was developed with class weighting to address class imbalances, without the use of data augmentation techniques. The model, trained with torch version 1.13.1, achieves a high accuracy of 99% on the training set and 96% on the test set.

For local testing, an equivalent ”original” ResNet-18 model was pre-trained on the MUFAC dataset. The training process was conducted using the Adam optimizer with a learning rate of 0.0005 and a weight decay of 5×10^{-3}, spanning across 36 epochs. The resulting model attained a training accuracy of 78.5% and a test accuracy of 62.1%.

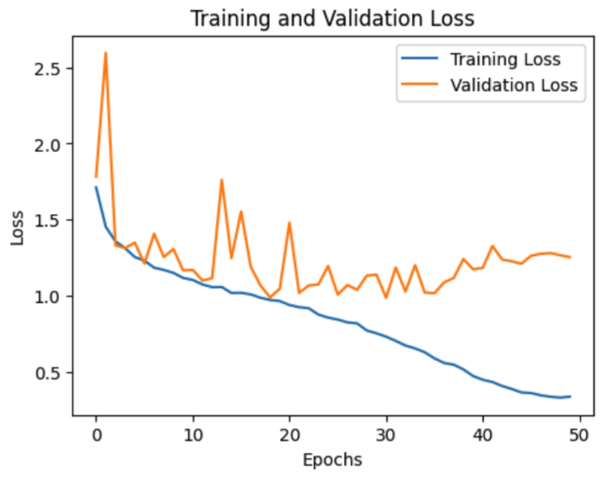


Figure 4. Training and Validation Loss.

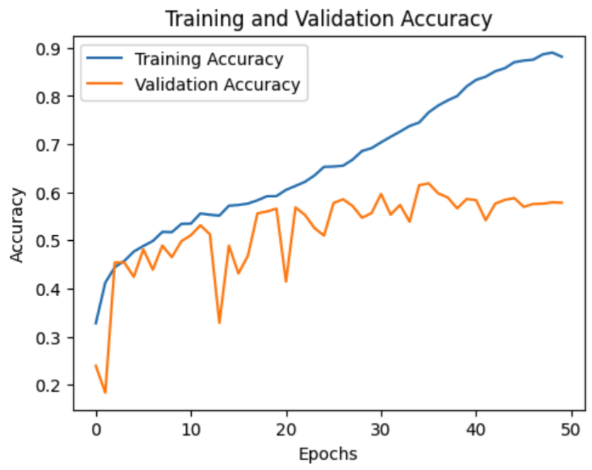


Figure 5. Training and Validation Accuracy.

The training and validation loss, along with the accuracy curves, are illustrated in Figures 4 and 5 respectively. The pre-trained checkpoint for this model is available in my GitHub repository for further reference[2].

5.3. Evaluation Procedure

- **Goal:** Evaluate how different the outputs are between the ”oracle” retraining-from-scratch model R and the unlearned model U.
- **Forgetting Quality (F):** Evaluated using the black-box attacks on the outputs of retrained and unlearned models using the forget set S. For each forget set example, multiple attacks estimate False Positive Rate (FPR) and False Negative Rate (FNR). The worst-case attack (highest distinction between the two distributions) is used to compute an overall ”forgetting quality.” For more details, please refer to the the ”Preliminaries” section.
- **Model Utility:** Measure the utility of an unlearning algorithm by comparing the accuracy of unlearned models on a ”retain set” and a held-out test set, compared to that of the ”retrained” models.
- **Efficiency:** Unlearning algorithms slower than 20% of the retraining-from-scratch time are rejected.
- **Final Score:** The score rewards unlearning algorithms that exhibit both high forgetting quality and good utility. It’s calculated as:

$$F \times \frac{RA_U}{RA_R} \times \frac{TA_U}{TA_R}$$

  where F is forgetting quality and RA and TA denote retain and test accuracy respectively for unlearned (U) and retrained (R) models.

[2]https://github.com/YuliiaSuprun/MachineUnlearning/tree/main/checkpoints

5.4. Training and Hyper-parameter Tuning

The training process and the selection of hyper-parameters were critical to the success of the unlearning strategies. Detailed information about how I trained the models and the settings I chose is available in my GitHub notebook[3].

All my experiments were run using Google Colab, which allowed me to use high-end GPUs like the A100 or T4. After trying out different combinations, I found that using the Adam optimizer with specific settings (learning rate of 0.0001 and weight decay of 5e-3) gave the best results for the MUFAC dataset across all gradient-based learning strategies. However, for the hidden Kaggle dataset, the SGD optimizer with a learning rate of 0.001, momentum of 0.9, and a weight decay of 5×10^{-4} initially yielded the best results.

Interestingly, a shift in the loss function to weighted Cross-Entropy changed the situation. With this new loss function, the SGD optimizer performed very poorly, and the Adam optimizer with the same hyper-parameters (learning rate of 0.0001 and weight decay of 5e-3) showed superior performance for the hidden dataset, greatly outperforming SGD.

In terms of training epochs, a single epoch was generally sufficient for most strategies. If I trained for more epochs, the accuracy would start to drop. The only time I needed more epochs was when I was using the Student Teacher framework because it deals with smaller amounts of data (it uses a subset of the retain set), so I used five epochs for that method.

6. Kaggle Competition Outcome

In the Kaggle competition, I secured the 110th position on the Leaderboard with a score of 0.0646242203, using a simple fine-tuning approach. Interestingly, my friend (*guy477*), who submitted an identical solution, was placed at the 68th position with a slightly higher score of 0.0657146906. This variation in ranking, despite identical submissions, might indicate the stochastic nature of the scoring mechanism. Our positions were significant accomplishments given that this competition had 1121 participating teams. Notably, only 12 teams managed to surpass a score of 0.07.

An intriguing observation was the performance of the team "Mathematically-challenged," authors of the SSD approach, who were ranked at the 809th position with a score of 0.0550496249. This suggests that the SSD method may not generalize effectively to unseen datasets, potentially due to its over-reliance on dataset-specific characteristics.

6.1. High-performing Kaggle Submissions

6.1.1 Competent/Incompetent Teachers with Similarity-based Sampling

The naive implementation of the Competent/Incompetent Teachers approach initially showed really bad performance, its efficacy was largely influenced by the size and nature of the retained subset. If the retain set was significantly larger than the forget set, the model exhibited poor forget quality. Therefore, I had to sample a subset from the retain set. However, with a smaller subset of retain set, the forget set had a great influence, leading to a decline in accuracy.

An interesting approach was proposed by the team that secured the 12th place on the Public Leaderboard, who introduced the concept of similarity-based sampling as an alternative to random sampling for the retain set. This innovative approach is detailed in their Kaggle discussion post[4]. By using this method, I was also able to significantly enhance the performance on the MUFAC dataset in my local experiments (look at Approach 6).

Instead of random initialization, they pre-trained the Bad Teacher on the retain set. The student model starts with the Good Teacher's weights (the original model) and learns by aligning its predictions with the Good Teacher for retain points and with the Bad Teacher for forget samples through KL-divergence minimization (as described in the "Methods" section).

The key innovations they introduced are:

1. **Similarity-Based Sampling of the Retain Set:** During unlearning, not all retain images contribute equally. By choosing retain images similar in feature space to the forget images, the method modifies the feature space only around the forget images.

2. **Weighted Unlearning:** Retain images similar to multiple forget images are given more weight in the unlearning process. This helps to reconstruct the feature space around the forget images more accurately.

3. **Smart Batch Construction:** To maximize the impact of forget data and the Bad Teacher, this strategic batch construction alternates between the destructive influence of the Bad Teacher and the reconstructive influence of the Good Teacher and retain images.

[3] `https://github.com/YuliiaSuprun/MachineUnlearning/blob/main/Local_Scoring_on_MUFAC.ipynb`

[4] `https://www.kaggle.com/competitions/neurips-2023-machine-unlearning/discussion/458648`

This approach yielded promising results on the local dataset and the public leaderboard. Regrettably, their solution did not appear on the final leaderboard due to what is presumed to be a timeout issue. The team awaits feedback from the competition organizers for a definitive explanation.

6.1.2 Contrastive Fine-tuning (2nd place)

The second-place solution in the competition was built on a two-stage training method, incorporating both a forgetting stage and an adversarial fine-tuning stage. The first stage (1 epoch) forced the model to forget by diverging the output logits from their original predictions. The second stage (8 epochs) performed adversarial fine-tuning on the forget and retain datasets.

Initial Forgetting Stage: The model was made to forget by optimizing the KL-divergence between the output logits and a uniform pseudo label.

Adversarial Fine-tuning Stage:

1. **Self-supervised Contrastive Learning for Forgetting:** In this stage, the aim was to disperse the forget samples in the feature space, making them as dissimilar as possible from the retain set. This was achieved through a contrastive loss mechanism, which involved treating an enhanced version of a forget sample as a positive pair and all retain set samples as negative pairs. The formula for this loss is shown below:

$$l_i = -\log\left(\frac{e^{\text{sim}(x_i, x_i')/\tau}}{\sum_{j=0}^{\text{batch}_2} e^{\text{sim}(x_i, y_j)/\tau}}\right)$$

$$L_{\text{forget}} = \frac{1}{\text{batchsize}_1}\sum_{i=0}^{\text{batch}_1} l_i$$

2. **Equal Probability Contrastive Loss for Retain Samples:** Each retain sample was given an equal chance of being paired with any forget sample, ensuring fair weighting in the contrastive loss formula. This method was designed to balance the influence of retain samples in the feature space. The formula for this loss is as follows:

$$l_i = -\frac{1}{\text{batchsize}_2}\sum_{t=0}^{\text{batch}_2}\log\left(\frac{e^{\text{sim}(x_i, y_t)/\tau}}{\sum_{j=0}^{\text{batch}_2} e^{\text{sim}(x_i, y_j)/\tau}}\right)$$

$$L_{\text{forget}} = \frac{1}{\text{batchsize}_1}\sum_{i=0}^{\text{batch}_1} l_i$$

3. **Ensemble Effect Through Random Shuffling:** The mismatch in steps between the forget and retain loaders, referred to as ‘retain_ld4fgt‘ for contrastive learning, introduced a stochastic element to the training, akin to an ensembling technique.

4. **Cross-Entropy Loss for Retain Round:** To enhance the model’s performance on the retain dataset, a simple retraining with cross-entropy loss was implemented.

This approach is detailed in their Kaggle discussion post[5]. I also implemented this approach locally (Approach 7) and it showed a superior performance on the MUFAC dataset (please refer to the next section for more details).

7. Results and Discussion

This section evaluates the performance of various machine unlearning methods. The evaluation procedure, as described in the ”Experimental Setup” section, was carried out by comparing the retain accuracy, test accuracy, forgetting quality, and the overall score computed with $N = 10$ unlearned models for each method on the MUFAC dataset. The results of these experiments can be found in the project notebook available on GitHub[6].

[5]https://www.kaggle.com/competitions/neurips-2023-machine-unlearning/discussion/458721
[6]https://github.com/YuliiaSuprun/MachineUnlearning/blob/main/Local_Scoring_on_MUFAC.ipynb

Method	Retain Accuracy (%)	Test Accuracy (%)	Forgetting Quality	Total Score
Fine-tuning on Retain Set	79.15	63.88	**0.07452**	**0.07048**
Advanced NegGrad (λ=0.1)	77.62	63.39	0.03613	0.03312
SSD (λ=0.1, α=10)	52.00	46.88	0.02303	0.01047
GISA	**80.89**	**64.41**	**0.07330**	**0.07119**
Naïve Teachers (retain_scale = 3)	71.13	59.99	0.02085	0.01653
Teachers with Similarity-based Sampling	75.61	**65.08**	0.05643	0.05190
Contrastive Fine-tuning	**80.21**	58.97	**0.07766**	**0.06914**

Table 1. Comparison of machine unlearning methods based on local experiments.

The scoring system considered both the ability of the algorithms to forget the targeted data without compromising the accuracy of the retained data. The table above presents a comprehensive comparison across several methods.

Highlighted in bold are the top-performing scores in each category. Notably, the methods of GISA, Fine-tuning, and Contrastive Fine-tuning stand out for their high forgetting quality scores and competitive accuracy metrics. The success of Fine-tuning in the Kaggle competition and the second-place achievement of Contrastive Fine-tuning suggest a high degree of generalizability. In contrast, SSD's poor performance, both locally and in the Kaggle competition when faced with an unseen test set, underscores its lack of generalization and the limitations of its blunt approach.

GISA (Gradient-Informed Synaptic Adjustment) method, the main contribution of this project, while not tested in the Kaggle competition, appears promising as it combines the ideas of Fine-tuning with the targeted approach of the SSD importance-based framework. Further research and testing on diverse datasets are needed to evaluate the full capabilities and generalizability of GISA.

References

[1] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer. Membership inference attacks from first principles. Apr. 2022. arXiv:2112.03570 [cs].

[2] D. Choi and D. Na. Towards machine unlearning benchmarks: Forgetting the personal identities in facial recognition systems, Nov. 2023. arXiv:2311.02240 [cs].

[3] V. S. Chundawat, A. K. Tarun, M. Mandal, and M. Kankanhalli. Can bad teaching induce forgetting? unlearning in deep networks using an incompetent teacher, May 2022.

[4] J. H. P. K. I. G. M. K. G. K. D. P. T. K. Z. L. S. H. J. C. S. J. J. V. D. I. M. S. E. J. W. S. D. M. D. W. R. Eleni Triantafillou, Fabian Pedregosa.
https://kaggle.com/competitions/neurips-2023-machine-unlearning, 2023.

[5] J. Foster, S. Schoepf, and A. Brintrup. Fast machine unlearning without retraining through selective synaptic dampening, Aug. 2023.

[6] A. Golatkar, A. Achille, and S. Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks, Mar. 2020. arXiv:1911.04933 [cs, stat].

[7] L. Graves, V. Nagisetty, and V. Ganesh. Amnesiac machine learning, Oct. 2020. arXiv:2010.10981 [cs].

[8] T. T. Nguyen, T. T. Huynh, P. L. Nguyen, A. W.-C. Liew, H. Yin, and Q. V. H. Nguyen. A survey of machine unlearning, Oct. 2022. arXiv:2209.02299 [cs].

[9] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. Mar. 2017. arXiv:1610.05820 [cs, stat].