One-Shot Machine Unlearning with Mnemonic Code

Tomoya Yamashita TOMOYA.YAMASHITA@NTT.COM
NTT Social Informatics Laboratories

Masanori Yamada MASANORI.YAMADA@NTT.COM
NTT Social Informatics Laboratories

Takashi Shibata T.SHIBATA@IEEE.ORG
NTT Communication Science Laboratories

Editors: Vu Nguyen and Hsuan-Tien Lin

Abstract

Ethical and privacy issues inherent in artificial intelligence (AI) applications have been a growing concern with the rapid spread of deep learning. Machine unlearning (MU) is the research area that addresses these issues by making a trained AI model forget about undesirable training data. Unfortunately, most existing MU methods incur significant time and computational costs for forgetting. Therefore, it is often difficult to apply these methods to practical datasets and sophisticated architectures, e.g., ImageNet and Transformer. To tackle this problem, we propose a lightweight and effective MU method. Our method identifies the model parameters sensitive to the forgetting targets and adds perturbation to such model parameters. We identify the sensitive parameters by calculating the Fisher Information Matrix (FIM). This approach does not require time-consuming additional training for forgetting. In addition, we introduce class-specific random signals called mnemonic code to reduce the cost of FIM calculation, which generally requires the entire training data and incurs significant computational costs. In our method, we train the model with mnemonic code; when forgetting, we use a small number of mnemonic codes to calculate the FIM and get the effective perturbation for forgetting. Comprehensive experiments demonstrate that our method is faster and better at forgetting than existing MU methods. Furthermore, we show that our method can scale to more practical datasets and sophisticated architectures.

Keywords: Deep Learning, Machine Unlearning, Mnemonic Code

1. Introduction

Ethical and privacy issues inherent in AI applications have been a growing concern with the rapid spread of deep learning. For example, if an AI model has undesirable information from an ethical standpoint, this will be a barrier to applying the AI model in society. Ethical perspectives are largely based on social conditions, and the definition of "undesirable information" may change over time. Also, there may be cases where users or public organizations request the deletion of their information to the AI model that uses their data for training. In such cases, AI models must be modified immediately to respond to social changes and deletion requests.

MU is a research area responding to such demand (Nguyen et al., 2022). MU aims to make a trained AI model forget about undesirable training data. When we obtain an effective MU method, it will provide a stepping stone to solving the problems of ethics, data leakage, and so on. Thus far, while various MU methods have been proposed, most of them

© 2024 T. Yamashita, M. Yamada & T. Shibata.

incur significant time and computational costs due to the additional training or the use of large amounts of training data for forgetting. These methods often cannot apply to more practical datasets and sophisticated architectures such as ImageNet, Transformer, and so on. To make MU even more practical, a simple-yet-effective MU method is required.

To tackle this problem, we propose a one-shot MU method that does not incur significant time and computational costs. In this paper, we focus on class removal, which forgets about a particular class in the training data, e.g., scenarios like removing someone's facial information from a face authentication AI system. In our method, we identify the model parameters sensitive to each class by calculating the FIM. FIM is often used in continual learning to avoid catastrophic forgetting (Kirkpatrick et al., 2017; Huszár, 2018; Ritter et al., 2018). Then, we add the effective perturbation to the model parameters that increase the loss of the forgetting class without accuracy degradation for the remaining classes. In addition, we introduce class-specific random signals called mnemonic code to reduce the cost of FIM calculation, which generally requires the entire training data and incurs significant computational costs. Mnemonic code was first introduced to associate the information of each class with fairly simple codes (Shibata et al., 2021). In our method, when training a model, we prepare the mnemonic code per class and embed them in the model by stochastically replacing the training data with the mnemonic codes. Then, in the forgetting phase, we use a small amount of mnemonic codes to calculate the FIM and get effective perturbation. Our method does not require additional training or large amounts of training data, contributing to lightweight MU [1].

In the experiments, we use artificial and natural datasets to evaluate the forgetting capability and the MU processing speed of our method. In addition, through FIM estimation experiments, we confirm that mnemonic codes can approximate the Oracle FIM of the entire training data precisely and largely contribute to one-shot effective forgetting. Also, we show that our method works effectively for pre-trained models by applying a few steps of fine-tuning using mnemonic codes. Furthermore, our lightweight method can scale to more practical datasets and sophisticated architectures (e.g. ImageNet and Transformer). Our contributions are as follows:

- We propose a lightweight and effective MU method that adds one-shot perturbation to the model parameter. In addition, our method uses mnemonic codes to accelerate the perturbation calculation.
- Experimental results demonstrate that our method outperforms existing MU methods regarding the forgetting capability and the MU processing speed.
- We show that a few mnemonic codes could approximate the Oracle FIM of the entire training data precisely and largely contribute to one-shot effective forgetting.
- We show that our method can work for pre-trained models and scale to more practical datasets and recent sophisticated architectures.

1. The code is available on https://github.com/tomyamkum/OneShotMU-with-MNCode.

2. Related Work

MU was first introduced by Cao and Yang (2015). The original MU in early date is defined as removing the influence of the forgetting data points from the AI model so that the resulting model is indistinguishable from the model trained on a dataset without them. Since the concept of MU was first proposed, several types of unlearning requests have been introduced, i.e., item removal, class removal, task removal, and so on (Nguyen et al., 2022). This paper focuses on class removal, which is forgetting about a particular class in the training data. MU approaches can be divided into two categories: exact unlearning and approximate unlearning (Nguyen et al., 2022). Our method corresponds to approximate unlearning. Here, we describe them and introduce existing research.

Exact unlearning. The exact unlearning approach can provide unlearning proof. A typical approach to exact unlearning is re-training the model from scratch. While this approach can forget the information thoroughly, it often requires significant time and computational costs. Bourtoule et al. (2021) reduced the re-training cost for forgetting by subdividing the model and training data called a shard. Yan et al. (2022) also reduced the re-training cost by subdividing the model and the training data. They divided the training data by class and utilized the one-class classifier to keep the impact of forgetting into one class, reducing the accuracy degradation.

Approximate unlearning. The approximate unlearning approach estimates the contribution of the data to the model parameters and processes the model parameters. Since this approach does not re-train the model from scratch, it can save on forgetting costs.

Guo et al. (2020) formulated the MU problem setting called certified removal from differential privacy and proposed a method that can be applied to linear models. They also mentioned that their method can be applied to deep learning models by applying it to the final linear layer. Golatkar et al. (2020a) proposed a MU method using FIM for forgetting. Golatkar et al. (2020b) went on to propose a method that uses Neural Tangent Kernel and FIM. Foster et al. (2024) proposes a MU method using FIM, and they reduce the computational cost by reducing the number of FIM calculations. Tarun et al. (2023) achieved forgetting by training on adversarial noise that has a high loss to the forgetting classes. Chundawat et al. (2023) proposed a MU method that does not require training data by using the adversarial noise of each class. Lin et al. (2023) realized unlearning by transferring the knowledge of the remaining classes from the original model. During training, they introduced an entanglement-reduced mask (ERM) to reduce the knowledge entanglement in CNN models and effectively transfer knowledge in the forgetting phase. Shibata et al. (2021) proposed Learning with Selective Forgetting: a novel framework in which new tasks are learned while the previously learned target classes are forgotten through selective continual learning.

These approximate unlearning approaches aim to make a model forget by modifying the model parameters rather than re-training from scratch, and some studies aim to reduce time and computational costs for forgetting. However, these methods require additional training or a large amount of data for forgetting, making them difficult to apply to large practical datasets and sophisticated architectures. In contrast, our method is a one-shot MU method with mnemonic code that does not require additional training or a lot of training data, making it extremely fast and lightweight.

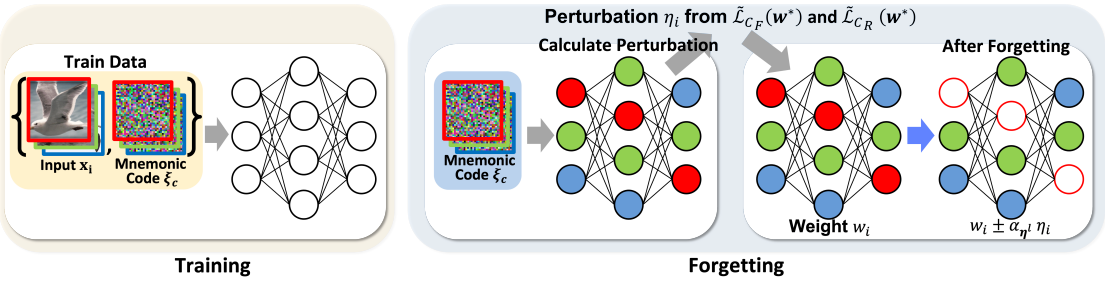


Figure 1: **Overview of our method.** We train the deep learning model with mnemonic codes in the training phase. The sensitive model parameters for each class are represented by color. In the forgetting phase, the target class is forgotten by perturbating the model parameters sensitive to that class.

3. Method

We propose a one-shot MU method with mnemonic code. An overview of our method is shown in Fig. 1. In our method, we identify the model parameters sensitive to each class and add effective perturbation that increases the loss of the forgetting class without accuracy degradation for the remaining classes. We identify the sensitive parameters by calculating the FIM. Furthermore, we introduce class-specific random signals called mnemonic code to accelerate the FIM calculation. When training a model, we prepare the mnemonic code per class and embed the codes in the model by stochastically replacing the training data with them. In forgetting, we use a small amount of mnemonic codes to calculate the FIM and get effective perturbation for forgetting. The following sections explain the mnemonic code and our training procedure. Then, we explain how to identify the model parameters sensitive to each class and how to obtain the effective perturbation for forgetting.

3.1. Proposal

Training with mnemonic code. Mnemonic code is a class-specific random signal introduced in Shibata et al. (2021). Our method uses mnemonic code to accelerate the FIM calculation in the forgetting procedure. When training a model, we prepare the mnemonic code per class and stochastically replace the training data with the mnemonic codes. Mnemonic codes have a class label, and the training data are replaced with the mnemonic codes of the same class. We generate each mnemonic code from a normal distribution, as with Shibata et al. (2021). The training algorithm is shown in Algorithm 1. We provide a theoretical analysis of training with mnemonic codes $\boldsymbol{\xi}$ to derive the effective perturbation for forgetting. The data distribution used in training is as follows:

$$p(\boldsymbol{x}) = t_{\text{mix}} p^{\xi}(\boldsymbol{x}) + (1 - t_{\text{mix}}) p^{d}(\boldsymbol{x}), \tag{1}$$

where $p^d(\boldsymbol{x})$ is the genuine data distribution and $p^{\xi}(\boldsymbol{x})$ is the data distribution of mnemonic codes. Here, $t_{\text{mix}} \in [0, 1]$ is the probability of replacing the training data, and in this paper, we set t_{mix} below 0.3. The setting of t_{mix} is described in Sec. 4.2.

Forgetting procedure. We attempt to forget the target class based on the above data distribution $p(\boldsymbol{x})$. Specifically, we design the one-shot perturbation $\boldsymbol{\delta}$ that increases the

Algorithm 1 Training with mnemonic code

Input: dataset $\boldsymbol{x} \sim p^d(\boldsymbol{x})$, model parameter $\boldsymbol{w}$, loss $\mathcal{L}$
Parameter: mnemonic code replacing probability t_{mix}, learning rate lr
Output: trained model parameters

```
ξ ~ N(0, 1)
for e in epochs do
    for i in datasize do
        t ~ U(0, 1)
        if t < t_mix then
            x̃_i = ξ_c
        else
            x̃_i = x_i
        end if
    end for
    w = w − lr∇_w L(x̃; w)
end for
```

loss of the forgetting class without accuracy degradation for the remaining classes. We first analyze how the perturbation affects the loss of the forgetting class and then consider the remaining classes.

$$
\begin{aligned}
&\mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^* + \boldsymbol{\delta}) \\
&\simeq \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*) + \frac{1}{2}\boldsymbol{\delta}^{\mathrm{T}} F_{\mathcal{C}_{\mathrm{F}}} \boldsymbol{\delta} \\
&= \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*) + \frac{1}{2}\boldsymbol{\delta}^{\mathrm{T}} \{t_{\mathrm{mix}} F^{\xi}_{\mathcal{C}_{\mathrm{F}}} + (1 - t_{\mathrm{mix}}) F^{d}_{\mathcal{C}_{\mathrm{F}}}\} \boldsymbol{\delta} \\
&\simeq \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*) + \frac{1}{2}\{t_{\mathrm{mix}} \sum_i \boldsymbol{f}^{\xi}_{\mathcal{C}_{\mathrm{F}},i} + (1 - t_{\mathrm{mix}}) \sum_i \boldsymbol{f}^{d}_{\mathcal{C}_{\mathrm{F}},i}\} \boldsymbol{\delta}_i^2,
\end{aligned} \tag{2}
$$

where $\boldsymbol{w}^*$ is the optimal parameter for the loss of the training data with mnemonic codes $p(\boldsymbol{x})$, $\mathcal{L}_{\mathcal{C}_{\mathrm{F}}}$ is the loss of the forgetting class $\mathcal{C}_{\mathrm{F}}$, $F_{\mathcal{C}_{\mathrm{F}}}$ is the FIM defined as follows,

$$
F_{\mathcal{C}_{\mathrm{F}},i,j} = \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x})} \left[\frac{\partial \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{x}; \boldsymbol{w})}{\partial w_i} \frac{\partial \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{x}; \boldsymbol{w})}{\partial w_j} \right], \tag{3}
$$

$F^{\xi}_{\mathcal{C}_{\mathrm{F}}}$ and $F^{d}_{\mathcal{C}_{\mathrm{F}}}$ are the FIMs calculated with the mnemonic codes $\boldsymbol{\xi}$ and the training data $\boldsymbol{x}$ in $\mathcal{C}_{\mathrm{F}}$, and $\boldsymbol{f}^{\xi}_{\mathcal{C}_{\mathrm{F}}}$ and $\boldsymbol{f}^{d}_{\mathcal{C}_{\mathrm{F}}}$ are the diagonal vectors of $F^{\xi}_{\mathcal{C}_{\mathrm{F}}}$ and $F^{d}_{\mathcal{C}_{\mathrm{F}}}$. We call $F^{d}_{\mathcal{C}_{\mathrm{F}}}$ as the Oracle FIM which is calculated with the entire training data. In Eq. 2, as with Kirkpatrick et al. (2017), Laplace's approximation is applied in the first line, and the diagonal approximation is applied in the last line [2]. Equation 2 shows that the fluctuation of the loss due to the perturbation $\boldsymbol{\xi}$ is determined by the linear sum of $\boldsymbol{f}^{\xi}_{\mathcal{C}_{\mathrm{F}}}$ and $\boldsymbol{f}^{d}_{\mathcal{C}_{\mathrm{F}}}$. Our method seeks the perturbation that increases the loss of the forgetting class. To design a lightweight MU

2. Laplace's approximation is to approximate the function by a Gaussian distribution and assumes that the first derivative of the approximated function is zero, i.e., $\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*) = 0$. We experimentally found that these values are of the same order as the first derivative of the loss for the training dataset: $\nabla \mathcal{L}(\boldsymbol{w}^*)$, which is generally assumed to be zero for trained models. We show the results in Appendix. B.

method, we aim to relax the restriction of using the entire training data to calculate the FIM in Eq. 2. In our method, we introduce the following loss instead of $\mathcal{L}_{\mathcal{C}_\mathrm{F}}(\boldsymbol{w}^* + \boldsymbol{\delta})$:

$$\tilde{\mathcal{L}}_{\mathcal{C}_\mathrm{F}}(\boldsymbol{w}^* + \boldsymbol{\delta}) = \mathcal{L}_{\mathcal{C}_\mathrm{F}}(\boldsymbol{w}^*) + \frac{1}{2}\sum_i \boldsymbol{f}^{\xi}_{\mathcal{C}_\mathrm{F},i}\boldsymbol{\delta}_i^2, \tag{4}$$

which does not need the training data to calculate the FIM. The validity of using the surrogate loss $\tilde{\mathcal{L}}_{\mathcal{C}_\mathrm{F}}$ instead of the loss $\mathcal{L}_{\mathcal{C}_\mathrm{F}}$ is assured by showing that the distance between $\boldsymbol{f}^{\xi}_{\mathcal{C}_\mathrm{F}}$ and $\boldsymbol{f}^{d}_{\mathcal{C}_\mathrm{F}}$ is sufficiently small. The details of the validity are discussed in Sec. 4.3.

From Eq. 4, we can see that large perturbation for the model parameters with large $\boldsymbol{f}^{\xi}_{\mathcal{C}_\mathrm{F}}$ cause the loss of the forgetting class $\mathcal{C}_\mathrm{F}$ to vary significantly. However, perturbating the model parameters in accordance with Eq. 4 can significantly reduce the accuracy for the remaining classes because we do not take into account the remaining classes. To avoid accuracy degradation for the remaining classes, we consider the sensitivity of each model parameter to the remaining classes, which can be derived in the same way as in Eq. 4. Specifically, we devise a strategy for achieving MU to add large perturbation to the model parameters such that the $f_{\mathcal{C}_\mathrm{F},i}$ is large while the $f_{\mathcal{C}_\mathrm{R},i}$ is small. We follow this strategy and propose the perturbation amplitude η_i for the model parameter w_i,

$$w_i = w_i \pm \alpha_{\boldsymbol{\eta}^l}\eta_i, \tag{5}$$

$$\eta_i = \frac{f_{\mathcal{C}_\mathrm{F},i}}{f_{\mathcal{C}_\mathrm{R},i}} = \frac{\frac{1}{|\#\mathcal{C}_\mathrm{F}|}\sum_{j\in\mathcal{C}_\mathrm{F}}\mathbb{E}\left[\left(\frac{\partial\mathcal{L}_j}{\partial w_i}\right)^2\right]}{\frac{1}{|\#\mathcal{C}_\mathrm{R}|}\sum_{k\in\mathcal{C}_\mathrm{R}}\mathbb{E}\left[\left(\frac{\partial\mathcal{L}_k}{\partial w_i}\right)^2\right]}, \tag{6}$$

$$\alpha_{\boldsymbol{\eta}^l} = \min\left(\lambda_1, \frac{\lambda_2}{\max_{\eta_i\in\boldsymbol{\eta}^l}\eta_i}\right), \tag{7}$$

where $\mathcal{L}_j$ and $\mathcal{L}_k$ are the losses of the mnemonic code of class $j \in \mathcal{C}_\mathrm{F}$ and class $k \in \mathcal{C}_\mathrm{R}$, $|\#\mathcal{C}|$ is the size of the class set $\mathcal{C}$. η_i is the perturbation amplitude added to the i-th model parameter, which is designed as a fraction, with the sensitivity to the forgetting class in the numerator and the sensitivity to the remaining classes in the denominator. If the value of η_i is large, the i-th model parameter is sensitive to the forgetting class and insensitive to the remaining classes. Therefore, Eq. 6 allows us to control the perturbation amplitude by its sensitivity to the forgetting and remaining classes. Note that there are two ways of adding positive and negative perturbation in Eq. 5. This is because the FIM only indicates the sensitivity of model parameters, which indicates how much the loss of the corresponding class changes when the model parameter changes. Therefore, η_i only indicates the perturbation amplitude for forgetting and has redundancy of positive or negative. In the forgetting phase, we add the amplitude η_i to the model parameters in positive and negative ways. Then, we adopt the resulting model that effectively achieves forgetting by measuring the error for the forgetting class and the accuracy for the remaining classes with mnemonic code. An overview of the forgetting algorithm is shown in Algorithm 2.

Design of coefficient $\alpha_{\boldsymbol{\eta}^l}$. $\alpha_{\boldsymbol{\eta}^l}$ is the coefficient in Eq. 5 and specified by the maximum value of $\boldsymbol{\eta}^l$ and hyperparameters λ_1 and λ_2. $\boldsymbol{\eta}^l$ is the set of η_i in the layer l, i.e. $\eta_i \in \boldsymbol{\eta}^l$. λ_2 specifies the maximum perturbation amplitude for each layer. In other words, λ_2 determines

Algorithm 2 Forgetting with mnemonic code

Input: trained model parameter $\boldsymbol{w}$, loss $\mathcal{L}$, forget class set $\mathcal{C}_{\mathrm{F}}$, remain class set $\mathcal{C}_{\mathrm{R}}$, mnemonic codes $\boldsymbol{\xi}$, layers $\{l_1, l_2, \cdots\}$
Parameter: λ_1, λ_2
Output: Forgotten parameters

$\boldsymbol{f}_{\mathcal{C}_{\mathrm{F}}} = \boldsymbol{0}$
$\boldsymbol{f}_{\mathcal{C}_{\mathrm{R}}} = \boldsymbol{0}$
for c in $\mathcal{C}_{\mathrm{F}}$ **do**
  $\boldsymbol{f}_{\mathcal{C}_{\mathrm{F}}} = \boldsymbol{f}_{\mathcal{C}_{\mathrm{F}}} + \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{\xi}_c; \boldsymbol{w})$
end for
for c in $\mathcal{C}_{\mathrm{R}}$ **do**
  $\boldsymbol{f}_{\mathcal{C}_{\mathrm{R}}} = \boldsymbol{f}_{\mathcal{C}_{\mathrm{R}}} + \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{\xi}_c; \boldsymbol{w})$
end for
$\boldsymbol{f}_{\mathcal{C}_{\mathrm{F}}} = \boldsymbol{f}_{\mathcal{C}_{\mathrm{F}}} / |\#\mathcal{C}_{\mathrm{F}}|$
$\boldsymbol{f}_{\mathcal{C}_{\mathrm{R}}} = \boldsymbol{f}_{\mathcal{C}_{\mathrm{R}}} / |\#\mathcal{C}_{\mathrm{R}}|$
$\boldsymbol{\eta} = \frac{\boldsymbol{f}_{\mathcal{C}_{\mathrm{F}}}}{\boldsymbol{f}_{\mathcal{C}_{\mathrm{R}}}}$
for l in layers **do**
  $\alpha_{\boldsymbol{\eta}^l} = \min\left(\lambda_1, \frac{\lambda_2}{\max_{\eta_i \in \boldsymbol{\eta}^l} \eta_i}\right)$
  $\boldsymbol{w}_1^l = \boldsymbol{w}^l + \alpha_{\boldsymbol{\eta}^l} \boldsymbol{\eta}^l$
  $\boldsymbol{w}_2^l = \boldsymbol{w}^l - \alpha_{\boldsymbol{\eta}^l} \boldsymbol{\eta}^l$
end for
if $A_{\mathrm{R}}(\boldsymbol{w}_1) + E_{\mathrm{F}}(\boldsymbol{w}_1) > A_{\mathrm{R}}(\boldsymbol{w}_2) + E_{\mathrm{F}}(\boldsymbol{w}_2)$ **then**
  return $\boldsymbol{w}_1$
else
  return $\boldsymbol{w}_2$
end if

the perturbation amplitude to the model parameter, which is the most effective for forgetting in each layer. λ_1 is introduced to avoid zero divisions in Eq. 7. Because the denominator of Eq. 7 is close to zero in layers where no parameters are sensitive to the forgetting class. λ_1 specifies the perturbation amplitude in such layers.

3.2. Preliminary experiment

The effect of mnemonic code on model accuracy. Our method stochastically replaces the training data with the mnemonic codes when training the model. In this section, we investigate the effects of mnemonic code on the model's test accuracy. We train several models by changing the probability of replacing the training data with the mnemonic codes $t_{\mathrm{mix}} \in [0, 1]$ and evaluate each model's test accuracy. In the preliminary experiments, we use a simple, fully connected network for MNIST and ResNet-18 (He et al., 2016) for CIFAR10 (Krizhevsky and Hinton, 2009), CUB200-2011 (CUB) (Wah et al., 2011), and Stanford Cars (STN) (Krause et al., 2013). The results are shown in Fig. 2. The results show that mnemonic codes do not significantly degrade test accuracy on CIFAR10, CUB, and STN, even if 80% of the training data is replaced with mnemonic codes. In the case

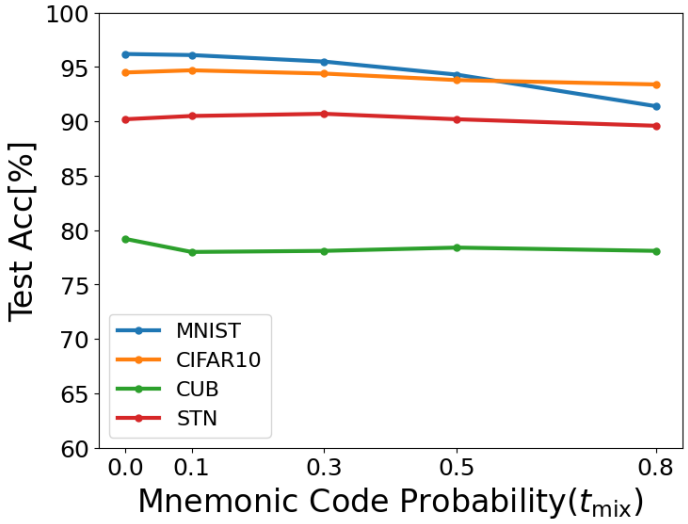


Figure 2: **Test accuracy of models trained with mnemonic code.** We evaluate the test accuracy of the model when varying t_{mix}: the probability of replacing the training data with the mnemonic codes.

of MNIST, we find that as t_{mix} increases, the model's test accuracy decreases [3]. However, the setting of t_{mix} in this paper is below 0.3, and we find that the accuracy degradation for $t_{\text{mix}} \leq 0.3$ is minute. In fact, the test accuracy degradation of the model trained with mnemonic codes ($t_{\text{mix}} \leq 0.3$) is less than 1%. These results show that training with mnemonic codes does not cause significant accuracy degradation.

4. Experiments

We evaluate our method from three perspectives: comparison with baselines (Sec. 4.2), the effect of mnemonic code (Sec. 4.3), and scalability (Sec. 4.4).

4.1. Common settings

We use two indicators to measure the performance of MU methods: i) the forgetting capability based on the test error for the forgetting class and the test accuracy for the remaining classes, and ii) the processing time for forgetting. We will describe the experimental flow. First, we train the deep learning model for 200 epochs. Then, we apply the MU method to the trained model on a specific class and evaluate the method. In this paper, we set the forgetting class $\mathcal{C}_{\text{F}}$ to $\{0\}$. The detailed train settings are described in Appendix A, and the evaluation for forgetting different classes is described in the supplemental material. As with existing studies, we evaluate the forgetting capability by two metrics: A_{R} and E_{F} (Shibata et al., 2021; Golatkar et al., 2020a,b; Tarun et al., 2023; Chundawat et al., 2023; Lin et al., 2023). $\boldsymbol{A}_{\mathbf{R}}$ is the test accuracy for the remaining classes, and $\boldsymbol{E}_{\mathbf{F}} = 100 - A_{\text{F}}$ is the test

3. We consider this is because the MNIST experiment uses a simple network, which was more strongly affected by mnemonic codes than ResNet-18.

Table 1: **Comparison with related studies.** We list the related MU methods and our method. We assess them from three perspectives: Processing Time, Data-Free, and MU Target. N_{R} and N_{F} are the numbers of the remaining and the forgetting data points. N_{new} is the number of the data points of the new task. E is the epochs of additional training for forgetting, and S is the steps to create adversarial noise. M is the number of model divisions. C_{R} and C_{F} are the numbers of the remaining and the forgetting classes.

Method	Processing Time	Data-Free	MU Target
CertifiedRemoval (Guo et al., 2020)	$\mathcal{O}(N_{\mathrm{R}} + N_{\mathrm{F}})$	✗	item
SISA (Bourtoule et al., 2021)	$\mathcal{O}(E \cdot \frac{N_{\mathrm{R}}}{M})$	✗	item
Arcane (Yan et al., 2022)	$\mathcal{O}(E \cdot \frac{N_{\mathrm{R}}}{C_{\mathrm{R}}+C_{\mathrm{F}}})$	✗	item
FastMU (Tarun et al., 2023)	$\mathcal{O}(S \cdot C_{\mathrm{F}} + E \cdot C_{\mathrm{F}} + N_{\mathrm{R}})$	✓	class
ZeroShotMU (Chundawat et al., 2023)	$\mathcal{O}((S+E)(C_{\mathrm{F}} + C_{\mathrm{R}}))$	✓	class
LwSF (Shibata et al., 2021)	$\mathcal{O}(E(N_{\mathrm{new}} + C_{\mathrm{R}}))$	✓	class/task
SFDN (Golatkar et al., 2020a)	$\mathcal{O}(N_{\mathrm{R}})$	✗	class/item
NTK-F (Golatkar et al., 2020b)	$\mathcal{O}(N_{\mathrm{R}} + N_{\mathrm{F}})$	✗	class/item
SSD (Foster et al., 2024)	$\mathcal{O}(N_{\mathrm{R}} + N_{\mathrm{F}})$	✗	class/item
ERM-KTP (Lin et al., 2023)	$\mathcal{O}(E \cdot N_{\mathrm{R}})$	✗	class
Ours	$\mathcal{O}(C_{\mathrm{R}} + C_{\mathrm{F}})$	✓	class

Table 2: **Comparison results in A_{R}.** We evaluate the baseline and our methods three times and provide the mean and standard deviation. The highest values are shown in bold.

	MNIST	CIFAR10	CUB	STN
FastMU	96.5 ±0.1	90.4 ±0.5	73.1 ±1.3	88.0 ±0.1
LwSF	43.7 ±9.6	65.4 ±16.6	68.2 ±3.5	80.1 ±6.7
SFDN	94.1 ±0.7	93.4 ±0.2	78.2 ±0.6	88.3 ±0.6
SSD	**96.9** ±0.0	94.2 ±0.0	44.3 ±0.0	74.4 ±0.0
ERM-KTP	-	92.7 ±0.4	42.8 ±3.2	75.6 ±4.0
Ours	95.9 ±0.1	**94.4** ±0.1	**79.3** ±0.7	**91.7** ±0.3

error for the forgetting class, where A_{F} is the test accuracy for the forgetting class. During the forgetting phase, we measure the time for forgetting. The desired MU method is the one that achieves high A_{R} and E_{F} in a short forgetting time.

Datasets and architectures. In our experiments, we prepare MNIST, CIFAR10, CUB, STN, and ImageNet (Deng et al., 2009). To coincide the setting with Shibata et al. (2021), we use 40 classes for CUB and 49 classes for STN. In the comparison experiments with baselines, we use a simple, fully connected model for MNIST and ResNet-18 for CIFAR10, CUB, and STN. In the scalability evaluation experiments, we use ResNet-18, ResNeXt-50 (Xie et al., 2017), and Swin-Transformer (Liu et al., 2021) for ImageNet. Our experiments are done on a server with AMD Ryzen 9 3950X 16 cores, 64 GB RAM, and RTX 3090 GPU.

4.2. Comparison with baselines

We compare our method and the existing baselines. For the hyperparameter set, we search $\lambda_1 = [10^{-6}, 10^{-5}, \cdots, 1.0]$ and $\lambda_2 = [10^{-1}, 1.0, \cdots, 10^{5}]$ and select the combination that maximizes the sum of A_{R} and E_{F}. Also, we set the probability of replacing the training

Table 3: **Comparison results in E_{F}.** We evaluate the baseline and our methods three times and provide the mean and standard deviation. The highest values are shown in bold.

	MNIST	CIFAR10	CUB	STN
FastMU	98.0 ±0.3	**100** ±0.0	68.6 ±12.0	60.9 ±6.9
LwSF	94.4 ±1.7	**100** ±0.0	93.1 ±7.0	98.2 ±1.8
SFDN	**100** ±0.0	96.3 ±2.5	**100** ±0.0	**100** ±0.0
SSD	93.1 ±0.0	**100** ±0.0	**100** ±0.0	**100** ±0.0
ERM-KTP	-	**100** ±0.0	**100** ±0.0	**100** ±0.0
Ours	**100** ±0.0	**100** ±0.0	**100** ±0.0	**100** ±0.0

data with the mnemonic codes t_{mix} as 0.1 for MNIST, CUB, and STN, and 0.3 for CIFAR10. The details of the hyperparameter setting are described in the supplemental material. We select the baseline methods from those described in Sec. 2. To select the baseline methods, we assess them from three perspectives: i) processing time, ii) the necessity for training data, and iii) the forgetting target. Table 1 summarizes the existing MU methods and our method. Our method can work quickly without the training data and target class removal. We evaluate the processing time with the number of backpropagations. Table. 1 shows that the processing time of our method depends only on the number of classes, while that of the existing MU methods depends on the number of epochs and data points. This is because they perform additional training or use large amounts of data for forgetting. The actual time required for forgetting is compared in Fig 3. From Table. 1, we select FastMU, LwSF, SFDN, SSD, and ERM-KTP as baseline methods. [4] While some need the training data for forgetting, we confirmed they can work in a realistic time and target class removal.

The comparison results for Forgetting Capability. Table 2 and 3 show the comparison results for the MU capability [5]. We repeat the evaluation three times and take the average of the results within the standard deviation error bars. The highest values in metrics A_{R} and E_{F} are shown in bold. The tables show that our method achieves 100% E_{F} while maintaining high A_{R} for all datasets. We can see that our method is superior to or competitive with the baselines. Fig. 3 shows the comparison results for MU processing time. These results show that our method works significantly faster than the baselines. This is because our method can efficiently calculate the FIM for each class with mnemonic code and add effective one-shot perturbation for forgetting, which does not need additional training. In addition, the MU processing time for ResNet-18 trained on ImageNet is shown in Fig. 3. This result shows that the MU processing time in FastMU is significantly increased for the large dataset. On the other hand, our method works quickly for such a dataset. We also confirm that the other baseline methods do not complete the MU process in a realistic time, which is omitted from Fig. 3. Detailed scalability evaluations for our method are given in Sec. 4.4.

4.3. Effect of mnemonic code on forgetting

We investigate the effect of mnemonic code on the forgetting capability. To investigate that, we prepare a baseline method that calculates the perturbation with the training data. We

4. We omit ZeroShotMU because it failed to reproduce the forgetting results for CIFAR10, CUB, and STN. We use the code of https://github.com/ayushkumartarun/zero-shot-unlearning.
5. ERM-KTP targets CNNs, so we omit the result for MNIST.

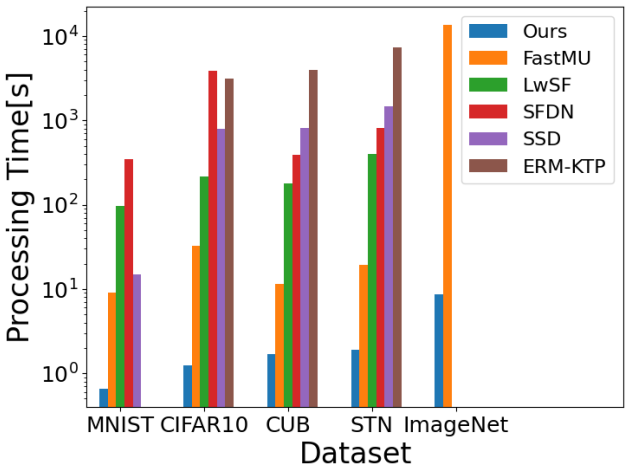
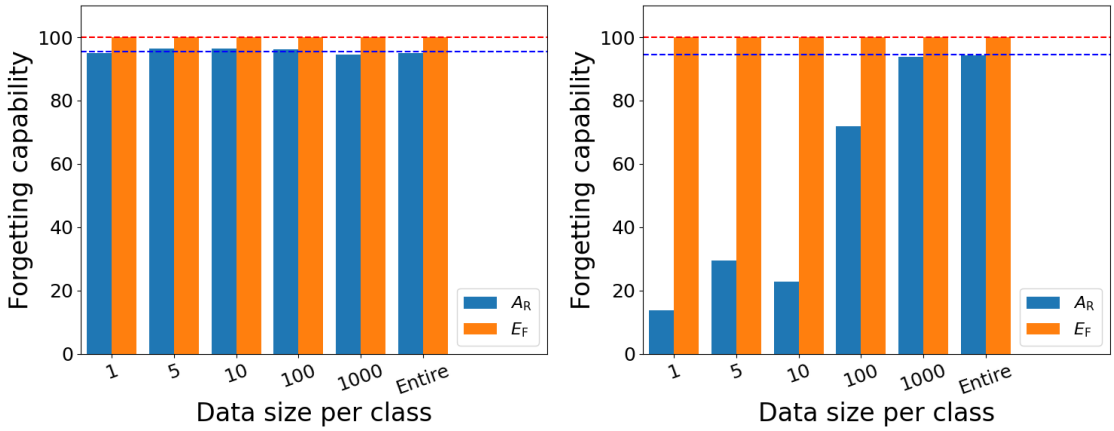


Figure 3: **Comparison results in MU processing time.** We measure the forgetting time concerning our method and the baselines.

call this baseline as ours-w-data. In ours-w-data, we train the model without mnemonic code. Then, in the forgetting phase, we obtain the FIM with a portion of the training data and add perturbation to the model parameter by Eq. 5. The difference between our method and ours-w-data is whether or not the mnemonic codes are used when training the model and calculating the FIM. Therefore, we expect that the difference in the forgetting capability comes from the contribution of mnemonic code. We measure A_{R} and E_{F} for the two methods and evaluate their forgetting capabilities. In these experiments, we use MNIST and CIFAR10. The results are shown in Fig. 4. From the results, we can clarify two things in ours-w-data. First, the forgetting process works effectively when we use all the training data. Second, while effective forgetting is possible with a small amount of training data in MNIST, effective forgetting is difficult with a small amount of training data in CIFAR10. We consider that this difference in datasets is due to the diversity of data contained in each class. Specifically, a simple dataset such as MNIST has a low diversity of data in each class. In contrast, a dataset such as CIFAR10 has diverse data in each class, and it will be difficult to represent each class with a small amount of training data. We consider this difference in datasets led to the difference in forgetting results in ours-w-data. Figure. 4 also shows the results of our method with mnemonic codes as a reference value. Surprisingly, our method can effectively forget with only one piece of mnemonic code per class.

We then conduct an experiment focusing on the FIM to reveal the cause of the difference in the forgetting capability between our method and ours-w-data shown in Fig. 4. We define the FIM calculated using all the training data as the Oracle FIM. As seen in Fig. 4, ours-w-data works well with the Oracle FIM for both MNIST and CIFAR10. Also, the Oracle FIM is often used in existing continual learning methods (Kirkpatrick et al., 2017; Huszár, 2018; Ritter et al., 2018). We evaluate the approximation error to the Oracle FIM of the FIM calculated using the portion of training data and of the FIM using the mnemonic code. When calculating the FIM approximation error, we calculate the L2 norm of the FIMs and divide it by the number of elements in the FIM (the number of the model parameters).



(a) Forgetting capability on MNIST.

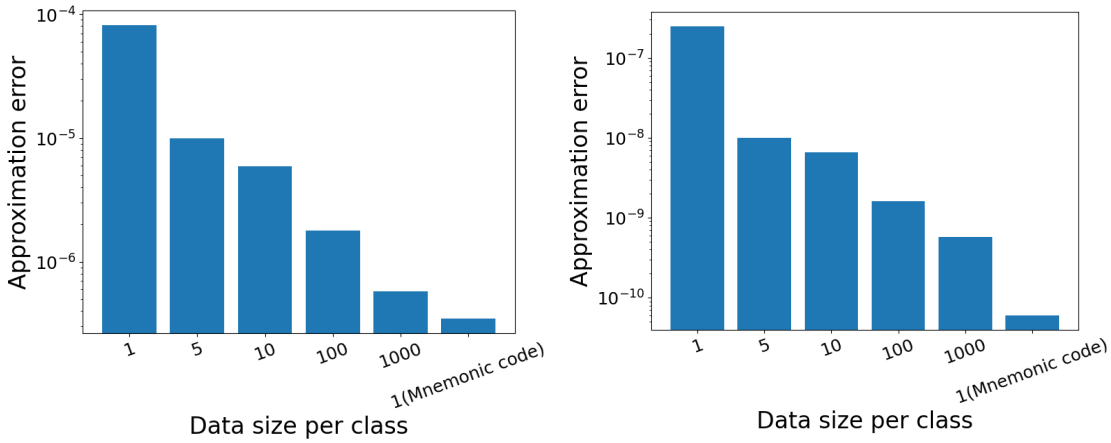
(b) Forgetting capability on CIFAR10.

Figure 4: **The forgetting capability of ours-w-data on MNIST and CIFAR10.** Our method with mnemonic code is also included for reference in dotted lines. The blue one shows A_{R} and the orange one shows E_{F}

Table 4: **Forgetting for ImageNet dataset.** We perform fine-tuning using mnemonic code on the pre-trained model and forget with our method. We show the forgetting capability and processing time for the pre-trained, fine-tuned, and forgotten models, respectively.

Architecture		A_{R} ↑	E_{F} ↑	Time [s] ↓
ResNet-18	Pretrained	69.8	12.0	-
	Fine-tuned	67.5	12.0	882
	After MU	67.5	100	8.66
ResNeXt-50	Pretrained	77.4	6.0	-
	Fine-tuned	75.9	12.0	6923
	After MU	75.9	100	21.0
Swin-Transformer	Pretrained	80.9	4.0	-
	Fine-tuned	78.8	4.0	8488
	After MU	75.3	92.0	28.6

The results are shown in Fig. 5. The results show that the smaller the number of training data for calculating the FIM, the further away from the Oracle FIM. Furthermore, we can see that the approximation error for the mnemonic code shown on the right side of the graph is significantly low. This is a remarkable result since it shows that the FIM obtained with only one mnemonic code is closer to the Oracle FIM than the FIM obtained with 1,000 training data. The high forgetting performance with mnemonic codes can be explained by the low approximation error of the FIM calculation. We consider that the low approximation error with mnemonic code is because the mnemonic code is presented to the deep learning model at a high rate of t_{mix} during training, while each training data is presented only one time per epoch. Furthermore, these results provide validity for replacing the Oracle FIM used in Eq. 2 with the FIM calculated with mnemonic code, as in Eq. 4.



(a) FIM approximation error on MNIST.

(b) FIM approximation error on CIFAR10.

Figure 5: **The FIM approximation error of ours-w-data on MNIST and CIFAR10.** Our method with mnemonic code is also included for reference.

4.4. Scalability

As shown in Fig. 3, our method works much faster than the baseline methods. This section demonstrates that our lightweight method is scalable to large practical datasets and sophisticated models. In the experiments, we prepare the pre-trained model and fine-tune it for a few steps using mnemonic codes. Then, we apply our method to the fine-tuned model. We evaluate the forgetting capability with A_{R} and E_{F}, and the processing time for fine-tuning and MU processing. The dataset is ImageNet, consisting of 1,000 classes, and the architecture is ResNet-18, ResNeXt-50, and Swin-Transformer. The number of fine-tuning steps is 2,000 for ResNet-18 and 10,000 for ResNeXt-50 and Swin-Transformer.

Table. 4 shows that our method works within one minute and improves E_{F} without significantly reducing A_{R}. It was also found that fine-tuning with mnemonic codes takes several hours. Therefore, if we fine-tune the pre-trained model with mnemonic codes in advance, we can quickly perform the MU process when an unlearning request arises. The results also show that our method can achieve effective forgetting for pre-trained models. Note that Table. 4 shows that test accuracy slightly degrades due to fine-tuning using mnemonic codes. We consider this because the number of classes in ImageNet is large, and the patterns of randomly generated mnemonic codes are insufficient. Future work will include obtaining mnemonic codes that maintain accuracy, even for large class datasets.

Limitation. We have found that multi-class forgetting is difficult with our method. Even in existing methods, multi-class forgetting methods are mainly based on additional training (Tarun et al., 2023; Chundawat et al., 2023; Lin et al., 2023), and multiple classes are difficult to forget with one-shot perturbation. Nevertheless, our method is valuable in that its overwhelmingly faster processing speed and its high scalability will enable us to respond rapidly to unlearning requests, as described in Sec. 1. Also, we did not fully evaluate our method concerning privacy. However, we evaluate the robustness against membership inference attacks and backdoor attacks in the supplemental material.

5. Future work and Conclusion

This paper proposes a one-shot MU method that achieves forgetting by adding perturbation to the model parameter. Mnemonic code is used to reduce the processing time and the computational costs. We experimentally showed that our method is lightweight and effective. Also, we experimentally demonstrated the effectiveness of mnemonic code. Furthermore, our method is scalable to more practical datasets and sophisticated architectures. In future work, we will seek mnemonic codes that do not degrade accuracy even for large class datasets. Also, we will seek a method that can forget multi-class effectively. In addition, generative models such as text-to-image models and large-language-models are rapidly advancing as a new application, and the issue of copyright of the data contained in the training data is surfacing (Carlini et al., 2021, 2023). Lightweight MU methods that can be applied to such sophisticated AI models should be required in the near future. We believe this paper will contribute to the practical MU research and new possibilities of lightweight MU.

References

Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy*, pages 141–159. IEEE, 2021.

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

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021.

Nicolas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramer, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5253–5270, 2023.

Vikram S Chundawat, Ayush K Tarun, Murari Mandal, and Mohan Kankanhalli. Zero-shot machine unlearning. *IEEE Transactions on Information Forensics and Security*, 2023.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Jack Foster, Stefan Schoepf, and Alexandra Brintrup. Fast machine unlearning without retraining through selective synaptic dampening. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 12043–12051, 2024.

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

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

Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Ferenc Huszár. Note on the quadratic penalties in elastic weight consolidation. *Proceedings of the National Academy of Sciences*, 115(11):E2496–E2497, 2018.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Shen Lin, Xiaoyu Zhang, Chenyang Chen, Xiaofeng Chen, and Willy Susilo. Erm-ktp: Knowledge-level machine unlearning via knowledge transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20147–20155, 2023.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

Thanh Tam Nguyen, Thanh Trung Huynh, Phi Le Nguyen, Alan Wee-Chung Liew, Hongzhi Yin, and Quoc Viet Hung Nguyen. A survey of machine unlearning. *arXiv preprint arXiv:2209.02299*, 2022.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 31, 2018.

Takashi Shibata, Go Irie, Daiki Ikami, and Yu Mitsuzumi. Learning with selective forgetting. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, volume 2, page 6, 2021.

Ayush K Tarun, Vikram S Chundawat, Murari Mandal, and Mohan Kankanhalli. Fast yet effective machine unlearning. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. *The Caltech-UCSD Birds-200-2011 Dataset*. California Institute of Technology, 7 2011.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2017.

Haonan Yan, Xiaoguang Li, Ziyao Guo, Hui Li, Fenghua Li, and Xiaodong Lin. Arcane: An efficient architecture for exact machine unlearning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, (IJCAI-22)*, pages 4006–4013, 2022.

Appendix A. Experimental Setting

We describe the experimental settings. We use the SGD optimizer for all datasets and train each model for 200 epochs. The learning rate is set to 0.01, and weight decay is set to 5×10^{-4}. We use a cosine scheduler for CIFAR10, CUB, and STN. The batch size is 128 for MNIST and CIFAR10 and 32 for CUB and STN. In fine-tuning with mnemonic code for ImageNet, the learning rate is set to 0.001, weight decay is 5×10^{-4}, and the batch size is 64. For the hyperparameter set $(\lambda_1, \lambda_2, t_{\mathrm{mix}})$, we describe in the supplemental material.

Appendix B. Validity of Laplace's approximation

We investigate the validity of the assumptions of the Laplace approximation, $\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*) = 0$ and $\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{R}}}(\boldsymbol{w}^*) = 0$, applied in Eq. 2. In general, $\nabla \mathcal{L}(\boldsymbol{w}^*) = 0$ is valid for trained models. Thus, if $|\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*)|$ and $|\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{R}}}(\boldsymbol{w}^*)|$ are close enough to $|\nabla \mathcal{L}(\boldsymbol{w}^*)|$, then we can verify the assumptions of the Laplace approximation. In the experiments, we train ResNet-18 on CIFAR10 with mnemonic codes and investigate the magnitude of loss gradient, $|\nabla \mathcal{L}(\boldsymbol{w}^*)|$, $|\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*)|$, and $|\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{R}}}(\boldsymbol{w}^*)|$ in each layer. The results are shown in Table. 5. These results indicate that $|\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*)|$ and $|\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{R}}}(\boldsymbol{w}^*)|$ are extremely small, about 10^{-5}. In addition, $|\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*)|$ and $|\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{R}}}(\boldsymbol{w}^*)|$ are of almost same order of $|\nabla \mathcal{L}(\boldsymbol{w}^*)|$ in all layers. From these perspectives, we consider that the assumption $\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*) = 0$ and $\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{R}}}(\boldsymbol{w}^*) = 0$ are valid.

Table 5: **The magnitude of loss gradient in each layer.**

Loss gradient magnitude	layer1	layer2	layer3	layer4	layer5
$\lvert\nabla \mathcal{L}(\boldsymbol{w}^*)\rvert$	6.9×10^{-6}	5.6×10^{-6}	7.4×10^{-6}	9.2×10^{-6}	1.6×10^{-5}
$\lvert\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{F}}}(\boldsymbol{w}^*)\rvert$	3.6×10^{-5}	3.0×10^{-5}	1.8×10^{-5}	1.2×10^{-5}	1.4×10^{-5}
$\lvert\nabla \mathcal{L}_{\mathcal{C}_{\mathrm{R}}}(\boldsymbol{w}^*)\rvert$	1.3×10^{-5}	9.9×10^{-6}	8.1×10^{-6}	7.6×10^{-6}	9.4×10^{-6}