Unlearning Trained Model by Activation Function deriving FIM

Jaehyun Choi

April 12, 2025

Abstract

Recently, AI technology is developing day by day. Technology for learning AI is developing, but, research on modifying learned AI to play a different role or erasing data is falling behind. It takes a lot of data and time to learn AI. In particular, AI often learns sensitive personal information such as faces, iris, fingerprints, and medical history. In addition, in the case of incorrect learning and overfitting or entering incorrect data, the data should be erased, which is very challenging in the current method. In this paper, we introduce new way to unlearning pre-trained model. Activation functions decide making forward or backward pass. In this reason, we will change the activation function while we retrain pre-trained model to unlearn model.

1 Introduction

In the case of detection models, generation models, and classification models, the technology has recently developed rapidly. Mostly, there are various models such as CNN-based multilayer models and Diffusion models. They have made tremendous progress and are actually used commercially.
However, since the model's performance or focus is only on learning, it has the vulnerability of information security that requires various data to be brought and used. In particular, in the case of models that require a lot of data, the learning time also takes a long time. Therefore, if the data provider no longer wants to use the model, the model must be discarded at all and re-learned so that information on the data is not left in the model. Therefore, in some cases, artificial intelligence needs not only to learn, but also to study forgetting, Unlearning.
In the case of previous unlearning studies, it can be divided into model-agnostic, model-intrinsic, and data-driven unlearning techniques[1].
One of the most representative methods is derived from Differential-privacy, Adaptive-machine Unlearning[2]. It uses diffential privacy to remove sequences. *One weakness of this condition is that it only guarantees the upper bound of the unlearning scheme compared to full retraining.*
The other representative method is using fisher information matrix to update weight. They

used that fisher information matrix can be approximated by hessian of Loss. But, so calculating all of the hessian costs much, that they approximate the Fisher information matrix as a diagonal matrix[3].

2 Background

2.1 Contribution

In this paper, we want to suggest new way to unlearn machine by improving Fisher information Masking and scrubbing. Actually, calculate all entries of fisher information matrix costs very high. This is why Adaptive machine unlearning[2] used only diagonal entries of a fisher information matrix. But, we will presuppose that fisher information matrix as kronecker factorized approximation[4]. So, we presuppose that inverse of fisher information matrix as a block diagonal matrix. We will illustrate the experiment with MNIST, CIFAR-10, and CIFAR-100 datasets.

2.2 Notation

We use datasets $\mathcal{D}$. And we can devide D into remain datasets$\mathcal{D}_{\nabla}$, and forgot datsets $\mathcal{D}_{\{}$. Because trained-model is a classfication model, we have N_C classes. The datasets $\mathcal{D}$ consists with $\mathcal{D}^x, \mathcal{D}^y$, and $\mathcal{D}^x$ is image data in this paper,$\mathcal{D}^y$ is one hot-encoded label. And in case of model, pre-trained model $\mathcal{M}_b$, after unlearning process $\mathcal{M}_f$, unlearning process model is $\mathcal{M}_u$, and each $l-th$ layer parameters are $\theta_l^b, \theta_l^f, \theta_l^u$. Loss of forward pass $\mathcal{L}$, and layer inputs, activation function, layer outputs at $l-th$ layer are $\mathcal{I}_l, \sigma_l, \mathcal{S}_l$. Model output is $\mathcal{O}$. Number of layer is N_l.

2.3 Preliminaries

The goal of learning model is minimize the loss of ground labels and model outputs.

$$\arg\min_w \mathcal{L}\left(w, D\right)$$

And, traditionally and until now to do machine learning, we use gradient descent with learning rate α.

$$w \leftarrow w - \alpha \nabla_w \mathcal{L}\left(w, D\right)$$

But, above equations just means that learn the model with all of data. We want to delete specific learned data. And influence function can adjust it, and in the paper, [5], [6], they suggest that influence function can be replaced with hessian of loss.

$$w_r^* \approx w^* + \frac{1}{|D_f|} \nabla_w^2 \mathcal{L}(w^*, D)^{-1} \nabla_w \mathcal{L}(w^*, D_f) \tag{1}$$

But, in this case, we also have to calculate the hessian of loss. It costs also very high. Not also it cost high, but also hessian of loss doesn't share the process with gradient of loss because those are calculated from different data sets. In this paper we want to approximate equation (1) with gauss-newton approximation and, suggest new learning process.

3 Proposed method

We will use Gauss-Newton approximation to compare the difference between the hessian of losses derived from D and D_f.

$$H \approx G = J_f^T H_\sigma J_f \tag{2}$$

Also, in this learning process we used Cross entropy loss and Softmax activation function. So, in this case we will combine softmax activation function and Cross entropy as A.

$$H = \nabla^2 \mathcal{L} \approx \left(J_{O_{in}}^T H^A J_{O_{in}} \right) \tag{3}$$

So, we have to calculate the hessian of A-function.

$$H_{ij}^A = \sum_{k=1}^{C} y_k \left[(\delta_{ik} - \sigma(x_i))(\delta_{jk} - \sigma(x_j)) - \frac{1}{\sigma(x_k)} (H_{ij}^\sigma)_k \right] \tag{4}$$

To caculate this we need Hessian of softmax function. Let's derive the second derivative value of the softmax function with respect to x_i and x_j. We will divide this process into different cases based on the values of i, j, and k. The base formula is:

$$\text{softmax}(x_k) = \frac{e^{x_k}}{\sum_{l=1}^{N} e^{x_l}}$$

The general formula for $\frac{\partial \text{softmax}(x_k)}{\partial x_i}$:

$$\frac{\partial \text{softmax}(x_k)}{\partial x_i} = \begin{cases} \text{softmax}(x_k)(1 - \text{softmax}(x_i)) & \text{if } i = k \\ -\text{softmax}(x_k)\text{softmax}(x_i) & \text{if } i \neq k \end{cases}$$

Now, we will find the value of $\frac{\partial^2 \text{softmax}(x_k)}{\partial x_j \partial x_i}$. Let's divide the calculation into cases based on the values of i, j, k, and softmax(x)= $\sigma(x)$ Utilize the appropriate and suitable results from the above cases to calculate the second derivative values.

$$\therefore \left(H_{ij}^\sigma \right)_k = \begin{cases} \sigma(x_k)(1 - \sigma(x_j))(1 - 2\sigma(x_i)) & \text{if } \ i = j = k \\ -\sigma(x_k)\sigma(x_j) + 2\sigma(x_i)^2\sigma(x_j) & \text{if } \ i = k \neq j \\ \sigma(x_i)\sigma(x_k)(2\sigma(x_j) - 1) & \text{if } \ i = j \neq k \\ 2\sigma(x_i)\sigma(x_j)\sigma(x_k) & \text{if } \ i \neq j \neq k \end{cases}$$

We will use H_{ij}^σ at Equation (5)

$$H_{ij}^A = \sum_{k=1}^{C} y_k \left[(\delta_{ik} - \sigma(x_i))(\delta_{jk} - \sigma(x_j)) - \frac{1}{\sigma(x_k)} (H_{ij}^\sigma)_k \right] \tag{5}$$

$$\left(H_{ij}^A \right)_k = \begin{cases} (1 - \sigma(x_j))\sigma(x_i) & \text{if } i = j \\ -\sigma(x_i)\sigma(x_j) & \text{if } i \neq j \end{cases}$$

We now know how can calculate second derivative of Loss function so we can change from equation (1) to equation (6).

$$H = \left(J_{O_{in}}^T H^A J_{O_{in}} \right) \tag{6}$$

$$J_{O_{in}} = \frac{dO_{in}}{d\mathcal{L}} \times J_{\mathcal{L}}$$

Let $\frac{d\mathcal{L}}{dO_{in}} = \Delta$, then $\frac{dO_{in}}{d\mathcal{L}} = \Delta^{-1}$. According to the Equation (6), we can derive Equation (7).

$$H^{-1} = \left(\mathcal{J_L}^T (\Delta^{-1})^T H^A \Delta^{-1} \mathcal{J_L} \right)^{-1} \tag{7}$$

The value of Δ should be caculated. Δ can be easily derived by derivative.

$$\Delta_t^{-1} = \begin{cases} \frac{1}{\delta_{it} - \sigma(O_{in}^i)} & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

Now, we can calculate $(\Delta^{-1})^T H^A \Delta^{-1}$.

$$M \coloneqq (\Delta^{-1})^T H^A \Delta^{-1} = \begin{cases} \frac{(1-\sigma_j)\sigma_i}{(\delta_{it}-\sigma_i)^2} & \text{if } i = j \\ \frac{-\sigma_i \sigma_j}{(\delta_{it}-\sigma_i)(\delta_{jt}-\sigma_{i\text{J}})} & \text{if } i \neq j \end{cases} \tag{8}$$

Using Eq.8, we can change Eq.7 as Eq.9

$$((\mathcal{J}^{*T} M \mathcal{J}^*)^{-1})_{ij} \approx \left[\frac{\sigma_t}{1-\sigma_t} \left[\frac{\partial \mathcal{L}_t}{\partial w_j} \frac{\partial \mathcal{L}_t}{\partial w_i} - \frac{\partial \mathcal{L}_t}{\partial w_j} \left(\sum_{k \neq t} \frac{\partial \mathcal{L}_k}{\partial w_i} \right) - \frac{\partial \mathcal{L}_t}{\partial w_i} \left(\sum_{k \neq t} \frac{\partial \mathcal{L}_k}{\partial w_j} \right) \right] \right.$$

$$\left. - \sum_{k \neq t} \frac{1-\sigma_k}{\sigma_k} \frac{\partial \mathcal{L}_k}{\partial w_i} \frac{\partial \mathcal{L}_k}{\partial w_j} - \sum_{k \neq t}^{n} \frac{\partial L_{n-k}}{\partial w_i} \frac{\partial L_k}{\partial w_j} \right]^{-1} \tag{9}$$

We can devide some case like $i = j$ or $i \neq j$.
Let,

$$\mathcal{T}_{ij} \coloneqq \left[\left(\left(\mathcal{J}^{*T} M \mathcal{J}^* \right)^{-1} \right)_{ij} \right]^{-1}$$

We have to approximate $\nabla_w^2 \mathcal{L}(w^*, D)$, not a $\nabla_w^2 \mathcal{L}(w^*, D_f)$. So, we have to caculate error between $\nabla_w^2 \mathcal{L}(w^*, D)$ and $\nabla_w^2 \mathcal{L}(w^*, D_f)$.

$$\nabla_w^2 \mathcal{L}^{-1}(w^*, D)_{ij} \approx T_{ij}^{-1}(\sigma \approx \frac{1}{N_C})$$

$$\nabla_w^2 \mathcal{L}^{-1}(w^*, D_f)_{ij} \approx T_{ij}^{-1}(\sigma_t \geq 0.5,\ \sigma_k \approx \frac{1-\sigma_t}{N_C - 1})$$

From equation 10, we can derive below equation.

$$\nabla_w^2 \mathcal{L}^{-1}(w^*, D)_{ij} \approx \left[\frac{1}{N_C - 1}\left[\frac{\partial \mathcal{L}_t}{\partial w_j}\frac{\partial \mathcal{L}_t}{\partial w_i} - \frac{\partial \mathcal{L}_t}{\partial w_j}\left(\sum_{k \neq t}\frac{\partial \mathcal{L}_k}{\partial w_i}\right) - \frac{\partial \mathcal{L}_t}{\partial w_i}\left(\sum_{k \neq t}\frac{\partial \mathcal{L}_k}{\partial w_j}\right)\right]\right.$$

$$\left. -(N_C - 1)\sum_{k \neq t}\frac{\partial \mathcal{L}_k}{\partial w_i}\frac{\partial \mathcal{L}_k}{\partial w_j} - \sum_{k \neq t}^{n}\frac{\partial L_{n-k}}{\partial w_i}\frac{\partial L_k}{\partial w_j}\right]^{-1}$$

$$\approx \left[-(N_C - 1)\sum_{k \neq t}\frac{\partial \mathcal{L}_k}{\partial w_i}\frac{\partial \mathcal{L}_k}{\partial w_j}\right]^{-1} \tag{10}$$

In this case, we can reduce some terms by approximation. But, in case of Equation is hard to approximate. In this case, we can use trick that make a batch mixing with other datasets with forget datasets to reduce σ_t.

$$\nabla_w^2 \mathcal{L}^{-1}(w^*, D_f)_{ij} \approx \left[-\frac{N_C - 1}{1 - \sigma_t}\sum_{k \neq t}\frac{\partial \mathcal{L}_k}{\partial w_i}\frac{\partial \mathcal{L}_k}{\partial w_j}\right]^{-1} \tag{11}$$

If we make a batch like this, we can use $\nabla_w^2 \mathcal{L}^{-1}(w^*, D_f)$ as $\nabla_w^2 \mathcal{L}^{-1}(w^*, D)$.
As we know hessian of loss is similar with fisher information function.

$$F \coloneqq \nabla_w^2 \mathcal{L}(w, D) \tag{12}$$

Then, we can approximate loss by using newton approximation.

$$\mathcal{L}(w, D) \approx \frac{1}{2}(w - w^*)^T F (w - w^*) + K$$

Let's chagne this equation to express F.

$$\nabla_w^2 \mathcal{L}(w^*, D) = F \approx 2\left(\mathcal{L}(w, D) - K\right)\left[(w - w^*)(w - w^*)^T\right]^{\dagger}$$

$$F^{-1} \approx \sum_{i}^{C} 2 \left(\mathcal{L}(w, D) - K\right)_i^{-1} (w - w^*)(w - w^*)^T$$

By using Eq.12 on Eq.5, we can change Eq.5.
(Because, $\nabla_w^2 \mathcal{L}^{-1}(w^*, D_f) \approx \gamma \nabla_w^2 \mathcal{L}^{-1}(w^*, D)$)

$$w_r^* \leftarrow w^* + \alpha \sum_{i}^{C} 2 \left(\mathcal{L}(w, D) - K\right)_i^{-1} (w - w^*)(w - w^*)^T \nabla_w \mathcal{L}(w^*, D_f)$$

We don't know hidden layer information. Therfore we have to change weight as well-known information. So, we will multiply layer input $\mathcal{I}$.

$$w_r^* \leftarrow w^* + \alpha \sum_{i}^{C} \left(\mathcal{L}(w, D_f) - K\right)^{\dagger} \mathcal{I}^{\dagger T} \mathcal{I}^T (w - w^*)^T (w - w^*) \mathcal{I} \mathcal{I}^{\dagger} \nabla_w \mathcal{L}(w^*, D_f)$$

And it can be expressed by layer output S.

$$w_r^* \leftarrow w^* + \alpha \sum_{i}^{C} \left(\mathcal{L}(w, D_f) - K\right)_i^{-1} \mathcal{I}^{\dagger T} S^2 \mathcal{I}^{\dagger} \nabla_w \mathcal{L}(w^*, D_f) \tag{13}$$

And we will gonna delete the sum of loss inverse term by putting itself to combine with S^2.

$$w_r^* \leftarrow w^* + \alpha \mathcal{I}^{\dagger T} (S^2 - \beta) \mathcal{I}^{\dagger} \nabla_w \mathcal{L}(w^*, D_f) \tag{14}$$

If we design above backpropagation by changing activation function between layer.

$$f'(x) = x^{-2}(f(x)^2 - \beta)$$

$$f(x) = \sqrt{\beta} \tanh\left(\frac{\sqrt{\beta} C x - \sqrt{\beta}}{x}\right)$$

The reason why we can apply it to each layer is Kronecker-Factorized approximation. In the case of K-FAC[4], approximate hessian as block diagonal and it means that each layer's weights are not related with other layer's weights.

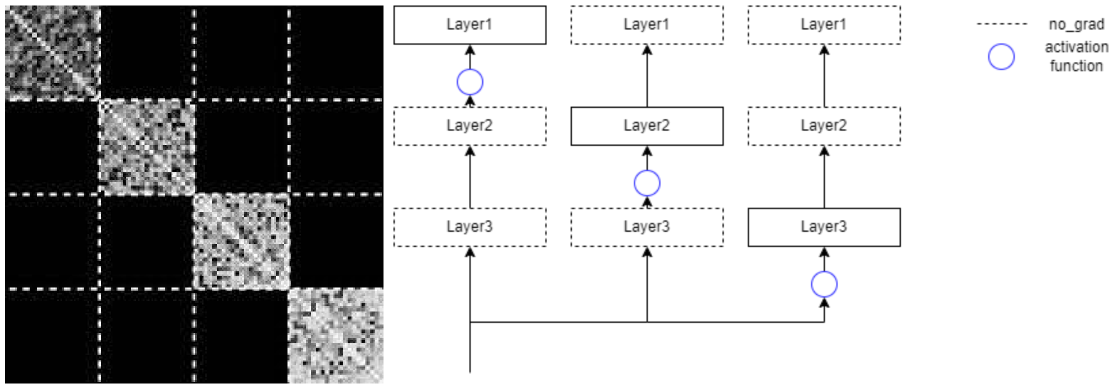

Figure 1: Kronecker factorized approximation, Proposed learning process

Algorithm 1: VGG learning process

```
Given: Input data $\mathcal{D}_f$, learning rate $\alpha$, matching Paarameter model inputs are $\mathcal{M}_A$
  and $\mathcal{M}_B$. And Correlation matrix is $\mathcal{C}$. Model output is parameter matched model
  $\mathcal{M}_{pm}$.
-----------------------------------------------------------------
Initialize: $\mathcal{M}_A$ is initialized by VGG trained model by $\mathcal{D}_f$, $\mathcal{M}_B$ is initialized by
  VGG pre-trained model by datasets except $\mathcal{D}_f$.
Forward Pass:
for $\mathcal{D}_i \in \mathcal{D}_f$ and $\mathcal{D}_i^x, \mathcal{D}_i^y \in \mathcal{D}_i$ do
    $\mathcal{O}_u \leftarrow \mathcal{M}_u(\mathcal{D}_i^x)$
    $\mathcal{L} \leftarrow$ Cross-entropy$(\mathcal{D}_i^y, \mathcal{O}_u)$
-----------------------------------------------------------------
Initialize: $\theta^u \leftarrow \theta^b$ and $\sigma \leftarrow$ ReLU for all layer of $\mathcal{M}_u$.
Backward pass:
for $i = 0,\ i < N_l,\ i++$ do
    $\sigma_i \leftarrow$ ReLU
$w \leftarrow w - \alpha \nabla \mathcal{L}$ in adam process
```

In this reason, we can devide this process in n time. So, Algorithm 1 is the process of Unlearning which we proposed.

In this algorithm, MSE denotes the mean squared error loss function, $\mathbf{z}gt$ represents the ground truth encoded features, and $\mathbf{c}gt$ represents the ground truth colors. The encoder network, f_{encoder}, takes the input image I_i, ray positions $\mathbf{p}_i$, and ray directions $\mathbf{d}_i$ as inputs and computes the encoded features $\mathbf{z}i$. The decoder network, $f$decoder, takes the ray positions $\mathbf{p}_i$ and ray directions $\mathbf{d}i$ as inputs and predicts the colors $\mathbf{c}i$. The parameters of the encoder and decoder networks, θencoder and θdecoder, are updated using gradient descent with learning rate η.

This algorithm outlines the basic training procedure of the NERF model, where the encoder network is trained to encode the input image and ray information, and the decoder network is trained to predict the colors of the rays. The training is performed iteratively over multiple epochs, and the model gradually improves its ability to reconstruct and render realistic 3D scenes. This algorithm can derive fisher information matrix by adapt backpropagation of changed activation function.

References

[1] 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.

[2] Varun Gupta, Christopher Jung, Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi, and Chris Waites. Adaptive machine unlearning. *Advances in Neural Information Processing Systems*, 34:16319–16330, 2021.

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

[4] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015.

[5] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017.

[6] Pang Wei W Koh, Kai-Siang Ang, Hubert Teo, and Percy S Liang. On the accuracy of influence functions for measuring group effects. *Advances in neural information processing systems*, 32, 2019.