CS772 Project Report

Ashutosh	Krish	Labajyoti	Shubham	Siddharth
210221	210530	210552	210709	211032

Abstract

Privacy laws like GDPR allow users the right to forget, making it mandatory for companies to delete all of user's data when requested. The data also has to be removed from machine learning models that used the data for their training. Thus, the idea of removing the information of a subset of training data from a trained model without retraining the model from scratch, viz Machine Unlearning, is becoming popular. For many models, the training data is not stored, making unlearning difficult. Two methods from the seed paper [1], i.e., Gated Knowledge Transfer and Error Minimizing-Maximizing noise, are investigated. The paper proposes two improvements to these methods. Our code can be found at https://github.com/SP1029/CS-772-Unlearning

1 Motivation

Given the increasing awareness and demand for privacy, many new laws like GDPR make it mandatory for private companies to delete the user's data if requested so. It may be possible that the user's data was used to train a machine-learning model. In that case, we require the deletion of data from an already trained model; that is, we would like to tweak the model's parameters such that the knowledge gained from the data requested to be deleted is lost from the model.

2 Problem Description

Suppose we have a model M trained using a dataset D, and now we want model M to forget the knowledge it has gained from a subset forget-data D_f of D. Ideally, we would have to train a new model M_r using retain-data D_u=$D \setminus D_f$ if we want to forget the data completely, but given the time and resources consumed in retraining, we would wish for a fast unlearning method that gives an unlearned model M_u from M that behaves similar to M_r.

There can be many variations of the problem, such as depending on whether D_u or D_f is available for unlearning purposes. We investigate a specific case of the extreme version of Zero-Shot Setting, in which, given a classification model M and a set C_f of classes, we have to forget the data belonging to classes in C_f from M, and we don't have access to the training data D. These constraints are reasonable in many practical scenarios, as for many models, the training data, being enormous, is not stored.

Let $D = \{x_i, y_i\}_{i=1}^N$ be a classification dataset with $x_i \in \mathbb{R}^d$ and $y_i \in \{1, 2, 3, ..., K\}$. Let $M(x; \theta)$ be a classification model trained on D, such that, for a data point x_*, $M(x_*; \theta)$ gives the probabilities of x_* belonging to each of the K classes. Let C_f be the set of classes we want the model M to forget, and C_r be the remaining set of retain-classes. Let $M_r(x; \theta_r)$ be the model formed by retraining a new model (using the same algorithm with which M was trained) from the retain-data. In zero shot setting, we are given the model M and set of forget classes C_f and we wish to learn a model $M_u(x; \theta_u)$ such that

$$M_u(x; \theta_u) \approx M_r(x; \theta_r)$$

3 Related work

Our work is an attempt to improve upon the seed paper 'Zero Shot Machine Unlearning' [1]. It discusses two approaches to the unlearning problem for the given zero-shot setting:

1. Error Minimization-Maximization Noise
2. Gated Knowledge Transfer

These are described in detail in further sections

3.1 Error Minimization-Maximization Noise

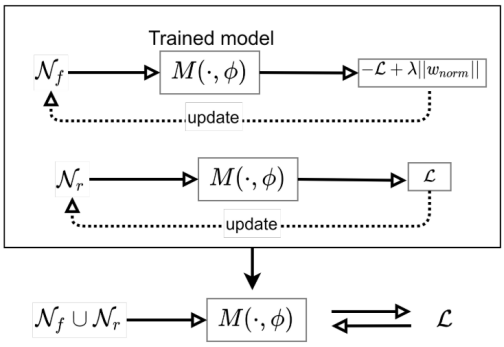

Figure 1: Error Minimization-Maximization Noise

This approach builds on the work presented in [2]. It trains a set of noise vectors $\mathcal{N}_f$ for each class in C_f by maximising the loss function of the corresponding classes on these vectors. These noise vectors act as sort of anti-samples, such that if we train the model M using them, it unlearns the class for which the noise vectors were trained. Thus, the noise vectors for i^{th} class are trained by maximizing the following loss:

$$L_N^{(i)}(\mathcal{N}_r^{(i)}) = L(M(\mathcal{N}_r^{(i)}; \phi), i)$$

The method presented in [2] requires access to a random subset of retain-data. Since we don't have access to retain-data, we also train a set of vectors $\mathcal{N}_r$ for each class in C_r minimizing the loss function of the corresponding classes on these vectors so that they act as substitute for retain data in the algorithm provided by [2].

Figure 1 gives an overview of the algorithm which involves three steps:

1. Training noise matrix $\mathcal{N}_f$ for each forget class by maximizing loss function as given above plus a regularisation term.
2. Training matrix $\mathcal{N}_r$ for the retain classes by minimizing the loss function of the corresponding class.
3. Training the model M using $\mathcal{N}_f \cup \mathcal{N}_r$

3.2 Gated Knowledge Transfer

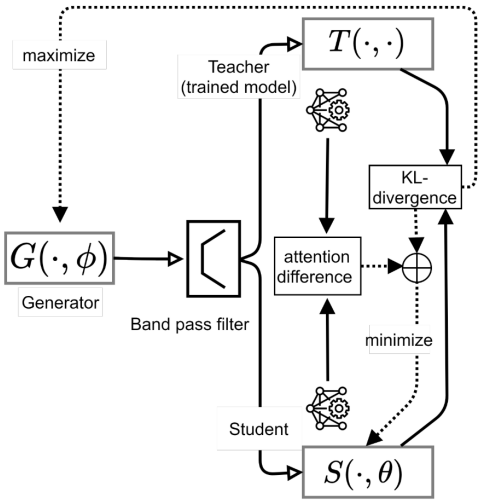

Figure 2: Gated Knowledge Transfer

This approach uses the idea presented in [3]. It uses knowledge distillation to copy the teacher model T (i.e., our original model M) to the student model $S = M_u$ of the same size as the teacher but also ensures that the student model does not learn the forgotten classes. It involves these key steps:

1. Since we don't have the training data, we need to generate data points similar to the training data to train the student. For this task, we have a generator that generates data points at which students and teachers differ the most. This is captured in the following loss function used by the generator to generate data points x_g based on the predictions of current student and teacher models:

$$D_{KL}(T(x_g)||S(x_g)) = \sum_i t_p^{(i)} \log \frac{t_p^{(i)}}{s_p^{(i)}}$$

   Since KL divergence captures the 'difference' between distributions, maximizing it between $T(x_g)$ and $S(x_g)$ will give those x_g's for which the student has learned the least.

2. We then apply a filter on x_g, which checks whether the probabilities of x_g as predicted by the teacher model is less than a constant ϵ, with the hope that this filters out the data points belonging to the forget classes, and thus student model does not learn classes in C_f.

3. If the data point passes the filter, we update the student model by minimizing the loss function:

$$L_s = D_{KL}(T(x_g)||S(x_g)) + \beta L_{at}$$

   where L_{at} is the attention loss:

$$L_{at} = \sum_{l \in N_L} \left\| \frac{f(A_l^{(t)})}{||f(A_l^{(t)})||_2} - \frac{f(A_l^{(s)})}{||f(A_l^{(s)})||_2} \right\|$$

   L_{at} captures the difference between activations of teacher and student. The loss function L_s tries to make student predictions equal to those of teacher's and also make their activations equal for the data point. Including activation term makes learning much faster.

These steps are repeated until the accuracy of the forget class doesn't start rising. To measure this, Deep Inversion [4] has been recommended to be used to produce substitute examples for forget class on which accuracy can be tracked. Figure 2 depicts this method diagrammatically.

4 Our approach

4.1 First attempt

Since we did not have access to training data, we used a generator to generate data points, but it may be possible that the data points generated do not belong to any of the classes; that is, $T(x_g)$ may not have a sharp peak at any one class, but is somewhat spread, and thus may not represent the training data well.

To solve this problem, instead of just using the probability of belonging to forget classes for x_g (as given by the teacher) to filter x_g, we also apply an upper bound criterion on the entropy of $T(x_g)$, with the hope that the data points passed from this new filter has its probability somewhat concentrated in a single class and thus is a better substitute for the training data for training student.

Entropy Filter:

$$H(T(x_g)) < \epsilon_H$$

4.2 Second attempt

4.2.1 Inconsistency between retraining and unlearning

We found that in the original GKT method, the accuracy for the forget class also increased after a large number of epochs. But when the model was completely retrained, this pheneomenon did not occur despite elongated retraining and the accuracy on forget class always remained 0. While we were trying to come up with reasoning none of them was satisfactory until we finally realised.

In the case of unlearning, while we are not training explicitly on points representing the forget class, attempting to mimic the teacher model's attention was likely causing the issue. Since the teacher has knowledge about the forget class, so its activation would be influenced by this. Trying to mimic these activations via L_{at} will indirectly cause the student model to eventually gain knowledge about the forget class while training. This is why the difference arose.

4.2.2 Improving Generation

Thus, to mitigate this issue, we tried training the students without using attention terms, but the accuracy of the students on retain classes did not exceed 50%. Upon investigation, we found that the images generated were of quite low quality, because of which we suspected the student's accuracy was very low. To improve upon this, we attempted to use DeepInversion [4], which generated very high-quality images given the original model M, resembling the training images quite well. We then trained a new model using the generated data from DeepInversion.

5 Experiments and Results

5.1 First Attempt

Contrary to our expectations, adding the criteria of having low entropy of predictions did not have the simple effect of increasing the performance of the model. It was highly influenced by the value of the threshold used. Figure 3 shows the graph of entropy threshold vs accuracy on retain class after 500 epochs. As it can be seen no concrete correlation can be concluded between the entropy threshold and retain accuracy.

We feel that this could be because there are two opposing factors. Decreasing the entropy might help in generating examples that better resemble the training data, but at the same time, these examples are more likely to have information regarding the forget class in their activations as compared to some random data. Due to this, the student model learns faster regarding the retain class, but also faster at learning the forget class.

We tried using different thresholds and got good results at a threshold of 0.6 (Colab link), where in just 500 epochs (and even lesser time because a larger rejection rate because of entropy criterion) we got an accuracy of 98.3%, where the original method took 850 epochs for the same.

Conclusion: We can see that for entropy threshold 0.6, our method works significantly faster as compared to orignal method and gives decent accuracy.

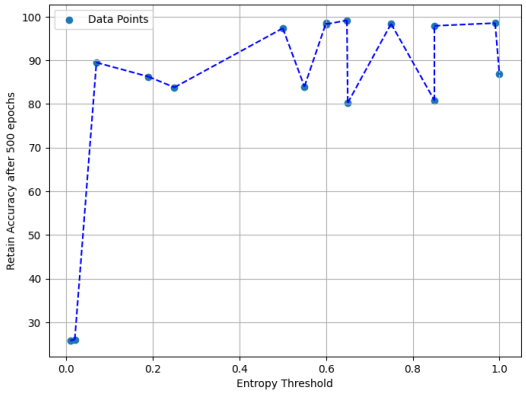

Figure 3: Entropy Threshold vs Retain Accuracy

5.2 Second attempt:

Initially, we had assumed that the images which would be generated by the generator would be really representative of the actual data points. When we ran the code and printed the images is when we realised their quality. Figures 4, 5 and 6 depict the images which had been generated by each of the approaches. We find that Deep Inversion resulted in much better image quality as compared to the generator in the original method (with or without entropy criteria).

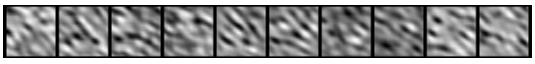

Figure 4: Generator without entropy criterion.

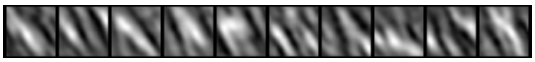

Figure 5: Generator with entropy criterion.

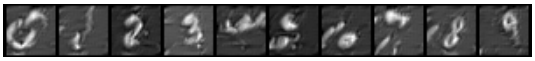

Figure 6: Deep Inversion.

Deep Inversion: It is a method that generates high-fidelity class-conditional images starting from random noise from a given trained CNN model. It minimizes regularized classification noise term plus the distance between the feature map statistics (mean and variance) of the input x and the expected statistics from the training data which are approximated by the batch normalization statistics.

As we had suspected, we found out that it was the attention term that was responsible for the non-zero accuracy of the student model on the forget class. Removing the attention term from the student's loss successfully helped to mitigate this issue, but at the same time, the student model was able to reach $\sim 50\%$ accuracy but was unable to cross this despite training for quite a long time.

We then utilised Deep Inversion to generate images using the teacher model. We first generated 100 samples per class which took around 7 minutes. In this case, the accuracy we were able to regain was pretty low. We believe this could be because the number of samples per class is much less as compared to the training set. To address this, we then generated 6000 samples per class, which took

around 7 to 8 hours. Using this we were able to get decent accuracy on the retain class. Table 1 highlights our results as compared to the original approach.

Method	Accuracy
Original Model*	97.84%
Retrained Model*	99.25%
GKT*	97.12%
EMMN*	10.57%
GKT (no attention)	<50%
Deep Inversion (100 sample/class)	40% - 50%
Deep Inversion (6000 sample/class)	80% - 85%

Table 1: Accuracy of retain data for MNIST dataset with $C_f = 0$. * As reported in the seed paper.

Limitations:

1. **Less accuracy :** While the Deep Inversion method with 6000 samples per class yields decent accuracy on the retain classes (80% - 85%), it still falls short of the accuracy achieved by xwthe model retrained from scratch (99.25%), as reported in the seed paper.
2. **Large generation time :** In our experiments, generating 6000 samples per class took approximately 7 to 8 hours, which is a considerable amount of time, especially when dealing with larger datasets or models. This extended generation time can be a bottleneck, limiting the practical applicability of the method.

Conclusion: We find that this method gives decent accuracy.

6 Tools/Software used

The following tools and software were used for the project:

1. **PyTorch**: The code used PyTorch as the ML library for implementing the algorithm. Most of our code is based on tweaking exisiting implementations appropriately.
2. **Kaggle and Google Colab**: We used Kaggle and Colab to run and modify the code due their provision for GPUs.

7 Learnings

1. **Ability to read research papers:** Since a major component of the project was to read and understand the research paper, the project gave us an experience on how to approach the same and helped us overcome the initial fear of research papers we had.
2. **First research experience:** This was our first research project. Thus, it gave us first-hand experience in pondering over and discussing multiple ideas and materializing them.
3. **Working with complex machine learning code:** To implement our improvements, we had to understand the original code for the algorithm implemented, identify relevant sections and tweak them appropriately.
4. **Team work:** We gained experience on how to manage a team and divide the work efficiently.

8 Possible future work

Some possible improvements on the seed paper can be as follows:

1. **Further Extending Approach 2**: The paper on Deep Inversion also presents another methodology named Adaptive Deep Inversion, which utilises a student-teacher framework to both generate a larger variety of samples and carry out knowledge distillation. This can be tweaked for machine unlearning. Another aspect that can be tried is generating even more examples per class, which might further help in improving the performance.

2. **Extending the method to regression:** The GKT method can be extended in case we are given a regression problem in which we have the model's Posterior Predictive Distribution. Given a model $M(-, \theta)$ such that for an input point x_*, $M(x_*; \theta)$ is the PPD for the label y_* corresponding to x_*. In this case, we may be given a certain range of y_* (say a to b), which we may want M to forget. In that case, the filter in the GKT method can be modified to pass only those points x_g having the probability of $y*$ to lie between a and b less than a threshold.
3. **Extending the method to unlearn a set of data points:** The Current method only works when we want to forget certain classes, but it may be extended to forget discrete given data points instead. In that case, the GKT method may be extended to submit, we may change the filter such that it only passes those data points that are at a certain ϵ distance away from the points that we want to forget, i.e., $||x - x_0|| > \epsilon$ for all x_0 in the data points that we want to forget.

9 Contribution

Ashutosh: Contributed to experimentation and discussions on novel approaches.

Krish: Contributed to model implementation and suggested improvements.

Labajyoti: Contributed to discussions and data collection from experiments.

Shubham: Contributed to improvements, implementation and video.

Siddharth: Contributed to improvements, experimentation and final document.

References

[1] V. S. Chundawat, A. K. Tarun, M. Mandal, and M. Kankanhalli. "Zero-shot machine unlearning." arXiv: arXiv:2201.05629v3 [cs.LG]. (2023).

[2] V. S. Chundawat, A. K. Tarun, M. Mandal, and M. Kankanhalli. "Fast yet effective machine unlearning." arXiv: arXiv:2111.08947v5 [cs.LG]. (2023).

[3] P. Micaelli and A. Storkey. "Zero-shot knowledge transfer via adversarial belief matching." arXiv: arXiv:2111.08947v5 [cs.LG]. (2023).

[4] H. Yin, P. Molchanov, Z. Li, *et al.* "Dreaming to distill: Data-free knowledge transfer via deepinversion." arXiv: arXiv:2111.08947v5 [cs.LG]. (2020).