Approximate Data Deletion via Weight Pruning

Lucas Orts {leorts}, Sophia Sanchez {sanchezs}, Will Song {jsong5}

Due: Wednesday, June 2nd

1 Introduction

1.1 Motivation

Modern machine learning algorithms learn from individual data. Recent legislation such as the General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) requires a "right to erasure" [4]. This right requires companies to take steps to erase an individual's data when requested by the individual. This may have downstream implications, e.g. if the user's data was used to train a ML model. Concretely, suppose we train a model on a training set $\mathcal{X}$. Later, suppose we want to 'delete' a data point x, that is, we want to remove x from the training set as well as any 'learning' gained from data point x from already trained models. The standard way to accomplish this is to entirely retrain the model from scratch on $\mathcal{X} \setminus \{x\}$, the training data with data point x removed, but this is usually computationally impractical. Ideally, some of the model can be salvaged while having some degree of assurance that the model has "forgotten" x. We will investigate algorithmic principles to modify existing models to accommodate efficient data deletion.

Our method takes as input a trained model m, a training set $\mathcal{X}$ and a target deletion point x. Our output is a model m' with the weights adjusted so that x is 'deleted' from the prediction space of m'. Intuitively, our approach quantifies the contribution of a single data point at each layer as the absolute value of the difference between the activation of said datapoint and that of its neighbors. That is, we capture the relationship between each datapoint and its closest neighbors at every layer and decide to remove the link or not. Though our method is very naive compared to the highly parametrized counterparts in the differential privacy area, our method has the advantage of a simple basis and implementation.

1.2 Related Work

Recent literature in differential privacy and machine learning revolve around the idea of approximate data deletion. Considering the data deletion problem, an issue immediately arises: How do we measure 'deletion'? Most recent work propose solutions to both problems. The canonical deletion metric is a differential privacy distance [1] or a comparison to an oracle model trained on the dataset without the target removal points [2]. Principally, researchers have developed novel gradient-based deletion methods such as the *projective residual update* [3] and *Quantized-k-Means* [2]. However, these methods are studied in the specific setting of k-means clustering [2], and convex models [5]. Instead, in this paper we focus on fully supervised tasks like classification and regression using a deep neural network (DNN).

Ginart et al. propose measuring "forgetfulness" as a distance to a model trained from scratch on a dataset with the desired points deleted. The authors present a novel Quantized-k-Means method that applies a 'correction' to the centroids to forget a datapoint. We use a similar 'forget' metric, but shift our focus to the setting of supervised deep learning. Alternative approaches include a feature map injection metric [3] or the standard ϵ-differential-privacy criterion. For our paper, we choose a metric similar to the one proposed by Ginart et al. due to its simplicity and clarity.

Bourtoule et al. explore the specific setting of single point unlearning using a strategic training framework, called SISAA, that expedites the unlearning process by limiting the influence that any single data point has on the model in the training procedure. Most of this work requires higher overhead techniques like ensemble methods and transfer learning for high complexity tasks. The authors create an augmented training method that trains the model in a distributed fashion using 'shards.' An important result from [1] is an analytical derivation showing a theoretical reduction in train time. However, this work requires extensive training procedures like training on different 'shards' of data, which is difficult to implement on larger preexisting models.

In this paper, we focus on a similar setting to that of unlearning of single point deletion. We present a heuristic approach based the logit contribution of a datapoint, that can be more readily implemented in preexisting models. Specifically, we propose a method for clustering data points in the activation space and modifying specific weights for approximating data deletion in a supervised fully connected setting.

2 Method

2.1 Description

Suppose we have the following input:

- A dataset $\mathcal{X}\times\mathcal{Y}$ of size N with tuples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
- A fully-connected neural network m with r layers, parametrized by activation functions $a^{[1]}, \ldots, a^{[r]}$ for $a^{[\ell]} \in \mathbb{R}^{m_\ell}$ and weight matrices $W^{[1]}, \ldots W^{[r]}$ for $W^{[\ell]} \in \mathbb{R}^{m_\ell \times d}$.
- A target removal point $(x^{(t)}, y^{(t)})$

We propose an algorithmic method, called the weight pruning method, which modifies model m to make it "forget" $x^{(t)}$ without having to fully retrain it. Namely, our method will select a particular subset of weights of m to modify and retrain. The method goes as follows: for each layer in the network, we find the n points, say $\{x^{(1)}, \ldots, x^{(n)}\}$, with the closest previous layer activations to the target removal point $(x^{(t)}, y^{(t)})$ via a clustering algorithm. Then, we run these n points in a forward pass through the layer. Letting $A^{(\ell)} = a^{[1]} \circ \cdots \circ a^{[\ell]}$ be the composition of the first ℓ activation functions of m, we define

$$\Delta^{(\ell)} := A^{(\ell)}(x^{(t)}) - \frac{1}{n}\sum_{k=1}^{n} A^{(\ell)}(x^{(k)}).$$

For every layer ℓ in m and each $w_{ij} \in W^{[\ell]}$, we compute $\left|\Delta_j^{(\ell)} w_{ij}^{[\ell]}\right|$. If that value is greater than some constant threshold $T^{(\ell)}$, we set the weight entry w_{ij} to zero. Then, we finetune the model m for additional iterations with the set $\mathcal{X} \setminus \{x^{(t)}\}$. The pseudocode for the weight pruning method is specified in the Weight Pruning Method section.

2.2 Justification

To provide some intuition on the meaning of $\Delta^{(\ell)}$ and the reasoning behind our method, we can think of $\left|\Delta^{(\ell)} w_{ij}\right|$ as providing information as to how much a weight w_{ij} is influenced by $x^{(t)}$. Indeed, observe that $\Delta^{(\ell)}$ is small if the activation of $x^{(t)}$ at the preceding layers is similar to the average activation of $\{x^{(i)}, ..., x^{(n)}\}$, and it is large if they are very different. When $\Delta_j^{(\ell)}$ and its corresponding weight w_{ij} are large, we can interpret w_{ij} as capturing some information that is present in $x^{(t)}$ and absent in $\{x^{(i)}, ..., x^{(n)}\}$. Thus, we set w_{ij} to zero to ensure that the model does not propagate that information such that it can "forget" $x^{(t)}$. In short, our method approximately removes all parts of the model that interact solely with $x^{(t)}$.

In our method, we set a weight w_{ij} to zero if $\left|\Delta^{(\ell)} w_{ij}\right| > T^{(\ell)}$. Note that the ideal threshold $T^{(\ell)}$ for each layer is dependent on the size of the dataset and the extent to which $x^{(t)}$ is an outlier datapoint. If the dataset is small or $x^{(t)}$ is an outlier datapoint, we want a lower threshold $T^{(\ell)}$ such that more weights are set to zero. If the dataset is large or $x^{(t)}$ is an "inlier" datapoint, we want a higher $T^{(\ell)}$ such that less weights are set to zero.

2.3 Weight Pruning Method

Algorithm 1: Weight Pruning Method

Input: Dataset $\mathcal{X} \times \mathcal{Y} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$; Target removal point $(x^{(t)}, y^{(t)})$; Model m with r layers; Thresholds $T^{(1)}, ..., T^{(r)}$; Number of neighbors n;

$A_{\mathrm{p}} \leftarrow \mathrm{Id}$ ▷ activation function of previous layer;

for *every layer* ℓ *in network* **do**

  $\mathbf{w} \leftarrow$ weight matrix of layer ℓ;

  $\mathbf{b} \leftarrow$ bias vector of layer ℓ;

  $A_\ell \leftarrow$ activation function of layer ℓ;

  $\mathcal{I} \leftarrow$ `find_nn`$(A_{\mathrm{p}}, x^{(t)}, n)$;

  for *each* (i, j) **do**

    $\Delta_j^{(\ell)} \leftarrow A_j^{(\ell)}(x^{(t)}) - \frac{1}{n}\sum_{k\in\mathcal{I}} A_j^{(\ell)}(x^{(k)})$;

    if $\left|\Delta_j^{(\ell)} w_{ij}\right| > T^{(\ell)}$ **then**

      $w_{ij}^{(\ell)} \leftarrow 0$;

    end

  end

  $b_{ij} \leftarrow 0.01$;

  $A_{\mathrm{p}} \leftarrow A_p \circ A_\ell$;

end

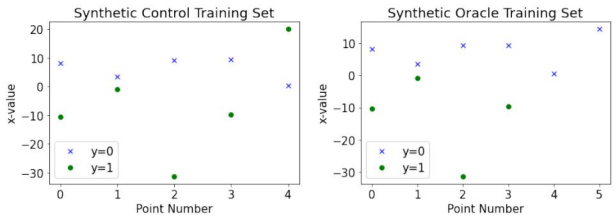
The method `find_nn`$(A, x^{(t)}, n)$ returns the indices of the n datapoints that are closest to $x^{(t)}$ with respect to the activation function A.

3 Datasets and Features

We evaluated our algorithm on two datasets: a synthetic Gaussian-sampled dataset and a California house pricing dataset. Self-generated data provides the benefit of complete control over the data distribution and dimensionality, enabling us to verify the thresholds and weight differences. Real-world data provides an opportunity to test our method's efficacy on real applications and larger learning problems.

3.1 Synthetic Dataset

We use a 10-point continuous dataset constructed with training space $x \sim 10 \cdot \mathcal{N}(0, 1)$, $x \in \mathcal{X}$ and label space $\mathcal{Y} \subset \{0, 1\}$. The data is split such that $y = 1$ if $x > 0$ and $y = 0$ if $x < 0$. Graphically, these look like:



For the oracle training set, the deletion target $x^{(t)}$ has been replaced with another point from a unit gaussian scaled by 10. We use a 10 times scaling to spread the data from the decision boundary.

3.2 California House Prices Dataset

For our real-world datset, we used publicly available data on 1990 census California housing prices data[1]. The dataset consists of 20.6 thousand examples of houses, where each example x contains 9 features, including the number of 'total_bedrooms' and 'ocean_proximity'. For numerical stability, each feature is normalized to the range $[0, 1]$ Each example has a corresponding label y (which is also normalized to the range $[0, 1]$), the median house value. We chose the deletion target $x^{(t)}$ to be the house with the highest house value. For the oracle training set, we removed $x^{(t)}$ such that the oracle training set has 1 less example than the control training set.

4 Experiments

For both datasets, we trained three models to assess the effectiveness of our data deletion method:

- A control model m_c, which we train on the set $\mathcal{X}$, the control training set;
- A removal model m_r, which we train on the control training set $\mathcal{X}$ and then apply our data deletion method to remove the target data point $x^{(t)}$;
- A oracle model m_o, which we train on the set $\mathcal{X} \setminus \{x^{(t)}\}$. We call $\mathcal{X} \setminus \{x^{(t)}\}$, where $x^{(t)}$ is replaced (either by another sample, or the nearest neighbors of the point) or simply removed, the oracle training set.

4.1 Deletion Metric

To measure the effectiveness of our deletion method, we chose to look at the L_1 norm of the difference in predictions between our removal model and the two base models. Namely, we calculate

$$S(m_r, m^\dagger, \mathcal{X}) = \|m_r(\mathcal{X}) - m^\dagger(\mathcal{X})\|_1$$

where m_r is the model using our removal method, $\mathcal{X} \in \mathbb{R}^n$ is our dataset, and $m^\dagger$ can be either the control model m_c or oracle model m_o. We refer to this as the logit L_1 distance in the following sections. Now, we define our deletion metric as follows:

$$d = \frac{1}{|\mathcal{Y}|}\left(S(m_r, m_c, X) - S(m_r, m_o, X)\right)$$

The bigger the value of d, the more effective our deletion method is. Indeed, if d is positive, our data deletion method is effective in that the removal model, m_r, is closer to the oracle model, m_o, than it is to the control model, m_c. Notably, if m_r resembles m_o, our data deletion method will have achieved in much fewer epochs of training what would traditionally require much more epochs of training (i.e. if we were to train m_0 from scratch entirely).

4.2 Model Training Setup

4.2.1 Synthetic Dataset Model

We use a 2-layer fully connected neural network with ReLU activations:

$$\begin{aligned} z_1 &= W^{[1]}x + b^{[1]} \in \mathbb{R}^3 \\ a_1 &= \text{Relu}(z_1) \in \mathbb{R}^3 \\ z_2 &= W^{[2]}a_1 + b^{[2]} \in \mathbb{R} \end{aligned}$$

(the above is for a single training example $x \in \mathbb{R}$), where $W^{[1]} \in \mathbb{R}^3$ and $W^{[2]} \in \mathbb{R}^{1\times 3}$. We train with learning rate 1e-3 and threshold of 9. We train both m_o and m_c until convergence on their respective datasets. For m_r, we train until convergence on the control dataset, prune the weights, then train until convergence on the oracle dataset.

4.2.2 House Price Dataset Model

We use a 3-layer fully connected neural network with ReLU activations:

$$\begin{aligned} z_1 &= W^{[1]}x + b^{[1]} \in \mathbb{R}^5 \\ a_1 &= \text{Relu}(z_1) \in \mathbb{R}^5 \\ z_2 &= W^{[2]}a_1 + b^{[2]} \in \mathbb{R}^5 \\ a_2 &= \text{Relu}(z_2) \in \mathbb{R}^5 \\ z_3 &= W^{[3]}a_2 + b^{[3]} \in \mathbb{R} \\ a_3 &= \text{Relu}(z_2) \in \mathbb{R} \end{aligned}$$

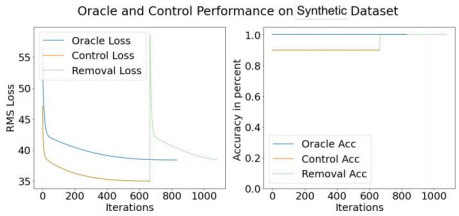
(the above is for a single training example $x \in \mathbb{R}$), where $W^{[1]} \in \mathbb{R}^{5\times 9}$, $W^{[2]} \in \mathbb{R}^{5\times 5}$, and $W^{[3]} \in \mathbb{R}^{1\times 5}$. We train with learning rate 1e-3 and thresholds of 4e-4, 1e-4 and 1e-5 for layers 1, 2 and 3 respectively. We train both m_o and m_c until convergence on their respective datasets. For m_r, we train until convergence on the control dataset, prune the weights, then train until convergence on the oracle dataset.

[1] https://www.kaggle.com/camnugent/california-housing-prices

4.3 Results

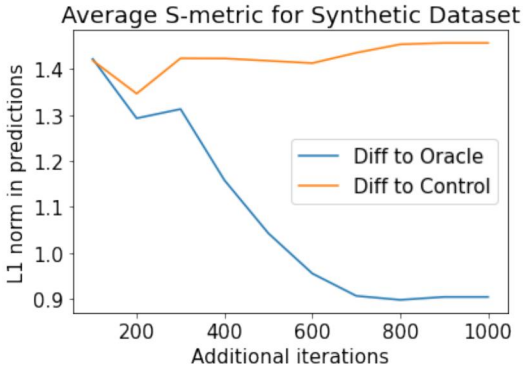
4.3.1 Synthetic Dataset

First, we plot the training performances for m_o, m_c, m_r for a typical, single iteration on the synthetic dataset.

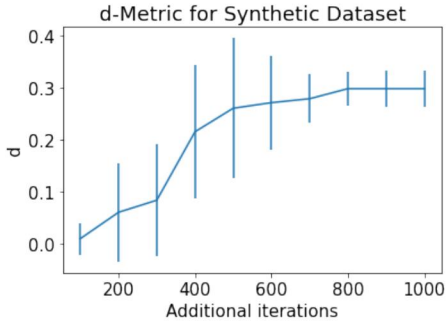


We see that the loss decreases well for both m_o and m_c. Notably the loss of m_r follows the loss of the control loss until weight pruning, where the m_r's loss spikes and then settles to more closely resemble the oracle loss. While not shown in this case, we often see an accuracy drop for m_r after pruning weights. This is likely because forcing weights to become zero in a continuous model can destabilize training.

Now, plotting the L_1 distance graph when averaging over 4 iterations, we get the following graph,



which illustrates how after pruning the weights, m_r's predictions become closer to those of m_o, as desired. Next, by taking the difference of the logit L_1 distances, we get the following graph of our deletion metric d:



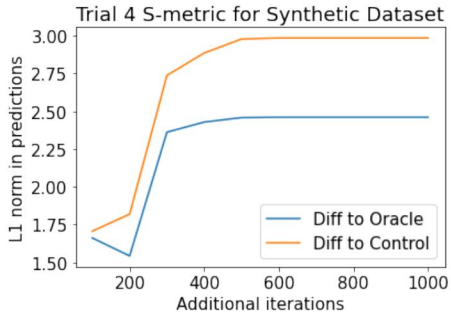
Note that the vertical lines are error bars representing a $\pm$ 1 standard deviation. According to the above graph, the deletion metric d becomes increasingly more positive, indicating that our method is effective in approximately "forgetting" $x^{(t)}$.

Synthetic Dataset Results					
Trial	#epochs for m_r convergence	#epochs for m_o convergence	Convergence Time Ratio	m_r acc.	m_o acc
1	488	748	.6224	0.9	1
2	466	1999	.2331	1	1
3	575	1906	.3017	1	1
4	791	790	1.001	0.9	1
avg	580	1360.8	0.5396	0.95	1

Table 1: Synthetic dataset convergence and accuracy.

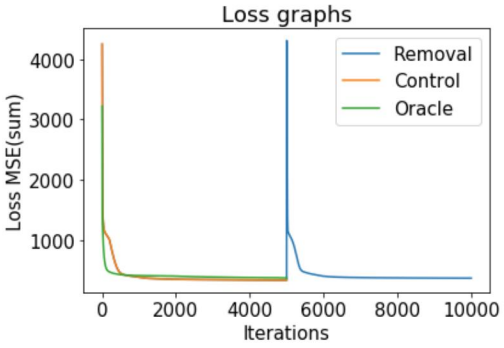
We summarize our results in Table 1. The number of epochs required to train the m_r is, on average, half the number of epochs required to train m_o (which is training entirely from scratch) and the average accuracy of m_r is comparable to that of m_o. This means that our weight pruning method on the synthetic dataset is able to approximate data deletion much faster than simply retraining from scratch on the modified dataset, without compromising the accuracy of the original model.

For trial 4, where the convergence time for m_r' is the same as that of m_o (that is, our pruning method was not more efficient than retraining from scratch), we found that m_r's logit distance to both m_c and m_o increased per additional iteration, indicating that the model was not converging to m_o but rather to another local optima entirely. We plot the logit distances below:



4.3.2 California Housing Prices Dataset

The loss plots of m_o, m_r, and m_c of a single iteration are seen below:

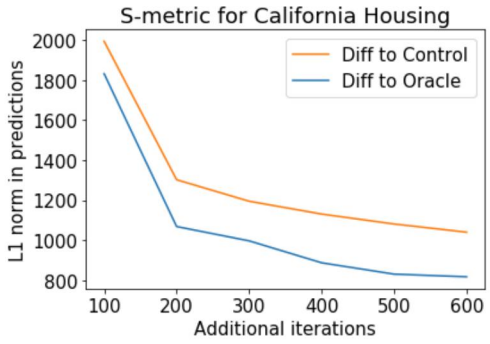


Before the weight pruning, we see that the losses for both m_c and m_o are nearly identical since the single removed datapoint added little to the dataset's complexity. Once the weights are removed, m_r is essentially retrained. Now, consider the following table (the convergence threshold was set to 0.015):

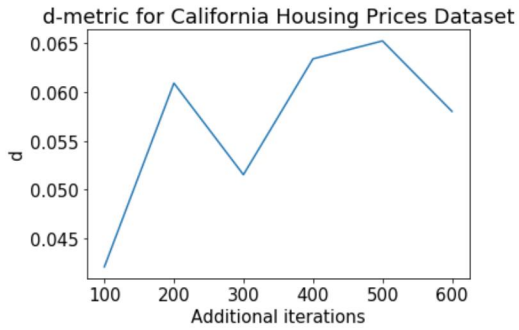
California Housing Dataset Results:

Model Name	Epochs to Convergence
m_o	463
m_r	443

Notably, m_o converged in about the same number of epochs as m_c (the ratio is equal to 0.95). Next, we plot the L_1 distance graph below:



We see that for the California housing dataset, the S-metric is generally decreasing, which indicates that m_r is converging to m_o in the prediction space, as desired. Now, compared to the equivalent graph for the synthetic dataset, we see a much closer gap between m_r and m_c, and m_r and m_o. This is because deleting merely one data point from roughly 20,000 points does not significantly change the regression task (even if the point is an outlier). We see m_r takes longer to converge after pruning compared to the synthetic dataset case because we aggressively zeroed out weights with very low thresholds. Specifically, we deleted more weights from m_r to become closer to m_o, which then resulted in comparatively longer convergence times.



As demonstrated by plotting the d-metric above, removing a single non-outlier point makes a very small difference in larger datasets. We are generally able to tune our threshold to aggressively zero weights, but this only improves our d-metric by a very small amount. This illustrates how having confidence that we have effectively 'forgotten' a non-outlier data point is generally much harder for larger datasets.

5 Conclusion and Future Work

In this paper, we presented a simple and intuitive approximate data deletion method using a weight pruning method. Our results on basic classification and regression problems serve as an empirical demonstration of the effectiveness of a simple technique. Reviewing literature, we found each paper had a slightly different definition of 'deletion' and approach with no relateable point of comparison. Hence, we contribute a simple weight pruning method that could serve as a baseline for comparison in future deletion models.

We tested our method on two distinct datasets: a small synthetic dataset and a large housing dataset. For both cases, we managed to approximate data deletion of an outlier target datapoint with our weight pruning method in an efficient manner, i.e. in no more epochs than the number of epochs it would have taken to fully retrain the model from scratch on the dataset with that target datapoint removed. We measured how well we 'forgot' a datapoint after pruning by using our data deletion metric d. Performance was a lot better on the synthetic dataset since working on large, noisy datasets presents a new set of challenges: a single datapoint (even an outlier) does not have a great impact on the model overall, which makes it more challenging to measure the confidence with which we know that a datapoint has been 'forgotten'. Indeed, the $L1$ logit distance from m_r to m_o and m_c will be basically identical, simply because m_o and m_c are basically identical.

A point of future exploration is the idea that pointwise model deletion may not be commutative so deleting 'batches' of data could be more involved than simply deleting one point. One may use a similar weight pruning method, but use a different threshold metric (similar to a set difference). Another major extension is finding a theoretical justification or bound relating the number of weights deleted and the amount of additional re-training. Given the empirical results on the simple Gaussian case, there should exist some kind of a theoretical bound given a one or two layer network.

6 Contributions

Our team has been been pair-coding and working on the project together on a Zoom call. Every member of the team has contributed their equal, fair share.

References

[1] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. *CoRR*, abs/1912.03817, 2019. URL `http://arxiv.org/abs/1912.03817`.

[2] Antonio Ginart, Melody Y. Guan, Gregory Valiant, and James Zou. Making AI forget you: Data deletion in machine learning. *CoRR*, abs/1907.05012, 2019. URL `http://arxiv.org/abs/1907.05012`.

[3] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Y. Zou. Approximate data deletion from machine learning models: Algorithms and evaluations. *CoRR*, abs/2002.10077, 2020. URL `https://arxiv.org/abs/2002.10077`.

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

[5] Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning, 2020.