DATA FORGING IS HARDER THAN YOU THINK

Mohamed Suliman[1,2] & Swanand Kadhe[1] & Anisa Halimi[1] & Douglas Leith[2]
Nathalie Baracaldo[1] & Ambrish Rawat[1]
[1] IBM Research [2] Trinity College Dublin, The University of Dublin
{suliman, swanand.kadhe, anisa.halimi}@ibm.com
doug.leith@tcd.ie baracald@us.ibm.com ambrish.rawat@ie.ibm.com

ABSTRACT

Recent research has introduced *data forging* attacks, which involve replacing mini-batches used in training with *different* ones that yield nearly identical model parameters. These attacks pose serious privacy concerns, as they can undermine membership inference predictions and falsely suggest machine unlearning without actual unlearning. Given such critical privacy implications, this paper aims to scrutinize existing attacks and understand the notion of data forging. First, we argue that state-of-the-art data forging attacks have key limitations, which make them *unrealistic* and easily detectable. Through experimentation on multiple hardware platforms, we demonstrate that *approximation errors* that existing attacks report are orders-of-magnitude higher than benign errors caused by numerical deviations. Next, we formulate data forging as an optimisation problem and show that solving it via simple gradient-based methods also results in high approximation errors. Finally, we theoretically analyse data forging for logistic regression. Our theoretical results suggest, even for logistic regression, it is difficult to efficiently find forged batches. In conclusion, our findings call for a reevaluation of existing attacks and highlight that data forging is still an intriguing open problem.

1 INTRODUCTION

Modern machine learning models, including large language models, have been shown to leak sensitive information about their training data or even memorize training data Carlini et al. (2019; 2021; 2022b). Their privacy risk is typically studied through the lens of membership inference attacks Shokri et al. (2017) which detect if a sample was used during training, or via compliance to the right to be forgotten with approaches like machine unlearning to handle data deletion requests Voigt & Von dem Bussche (2017). These important privacy implications have stirred up a large body of work on characterizing privacy leakage via membership inference attacks (see, e.g., Carlini et al. (2022a); Buzaglo et al. (2023); Haim et al. (2022); Hu et al. (2022)) and machine unlearning (see, e.g., Cao & Yang (2015); Bourtoule et al. (2021); Nguyen et al. (2022); Xu et al. (2023)).

Some recent works Thudi et al. (2022); Kong et al. (2023) have raised questions on the reliability of machine unlearning as well as membership inference attacks (MIAs). In particular, Kong et al. (2023) demonstrated that it is possible for an adversarial model owner to refute the prediction of a membership inference attack. Thudi et al. (2022) demonstrated that an adversarial model owner can claim that unlearning via re-training was performed without actually re-training the model. At the core of these works is the notion of *data forging*, proposed in Thudi et al. (2022).

Data Forging: At a high level, given a machine learning model's parameters θ and training dataset D, a data forging attack forges the training dataset D into a *different* dataset D' and produces a claim that the model was trained on D'. To achieve this, data forging attacks rely on the fact that supervised training uses iterative algorithms such as Stochastic Gradient Descent (SGD). These iterative training algorithms produce a *training trajectory* consisting of a sequence of model parameters and associated mini-batches, starting with the initialization to the final model parameters.[1] A data

[1] A log of the training trajectory can be formalized into a notion of Proof-of-Learning sequence Jia et al. (2021). We give more details in Sec. 2.

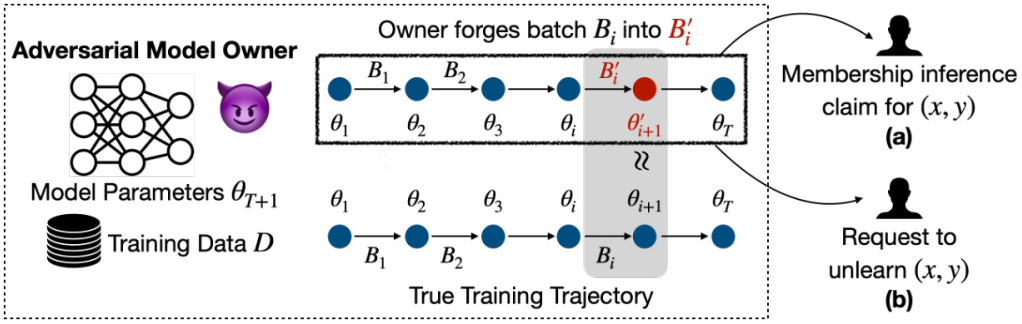


Figure 1: Privacy implications of data forging in two scenarios: (a) refuting membership inference claim for sample $\mathbf{x}$, and (b) unlearning sample $\mathbf{x}$ without actually re-training the model. The adversarial model owner has access to the true training trajectory denoting the sequence of model parameters and associated mini-batches. In both scenarios, an adversarial data owner *forges* a batch B_i that contains $\mathbf{x}$ with a different batch B_i' not containing $\mathbf{x}$ such that the resulting model θ_{i+1}' is nearly identical to θ_{i+1}.

forging attack replaces one or more mini-batches in the training trajectory with *forged* mini-batches that produce (nearly) identical gradient updates.

More specifically, let $\{(\theta_i, B_i)\}_{i=1}^{T}$ denote the training trajectory, where θ_i denotes model parameters at each gradient update step and B_i denotes the mini-batches used. To forge a batch B_i, the attacker constructs a different batch B_i' ($B_i \neq B_i'$) such that the model θ_{i+1} obtained by updating θ_i using the gradient at B_i is *nearly identical* to the model θ_{i+1}' obtained by updating θ_i using the gradient at B_i'. Here, nearly identical means that some distance (e.g., the ℓ_2-norm) between θ_{i+1} and θ_{i+i}' is below a given *small* error threshold ϵ. Such an *approximation error* is allowed because even when reproducing the gradient computation with the same mini-batch and model parameters, numerical deviations are possible due to benign sources of noise such as different hardware architectures.

Privacy Implications of Data Forging: Consider the scenario from Kong et al. (2023) where a claimant has used a membership inference attack to accuse a model owner of using their data sample $(\mathbf{x}, \mathbf{y})$ with features $\mathbf{x}$ and label $\mathbf{y}$ as part of training. Suppose indeed that $(\mathbf{x}, \mathbf{y})$ was part of the training dataset, and appears in batch B_i in the training trajectory. Now, if an adversarial model owner can forge B_i to B_i' that does not contain $\mathbf{x}$, they can refute the claim by providing the *forged* training trajectory $\{(\theta_1, B_1), \ldots, (\theta_{i-1}, B_{i-1}), (\theta_i, B_i'), (\theta_{i+1}, B_{i+1}) \ldots, (\theta_T, B_T)\}$ (see Fig. 1(a)). The adversarial model owner will repudiate the membership inference claim saying that the training trajectory does not contain any mini-batch that includes the claimant's data $(\mathbf{x}, \mathbf{y})$.

Similarly, in the scenario from Thudi et al. (2022) where the claimant is asking to unlearn their sample $(\mathbf{x}, \mathbf{y})$, an adversarial model owner may perform the same forging from B_i containing $\mathbf{x}$ to B_i' not containing $\mathbf{x}$. Then, by providing the *forged* training trajectory $\{(\theta_1, B_1), \ldots, (\theta_{i-1}, B_{i-1}), (\theta_i, B_i'), (\theta_{i+1}, B_{i+1}) \ldots, (\theta_T, B_T)\}$, they will claim to have *unlearnt* the data (see Fig. 1(b)). This is because as by the definition of *exact unlearning* Bourtoule et al. (2021), the final model parameters are the result of training on a dataset that does not contain the data required to be unlearnt. Note that the adversarial model owner does not have to perform any actual re-training.

Our Contributions: Given the serious privacy implications of data forging, it is imperative to scrutinize existing attacks and deepen our understanding of the notion of data forging. As a first step towards this, we ask *how 'realistic' are state-of-the-art data forging attacks?* We answer this question negatively by demonstrating key limitations of existing data forging attacks and argue that these limitations make the current attacks unrealistic and easily detectable. In particular, we show that the errors reported by existing attacks are too high to be acceptable as benign numerical deviations. Towards this end, we conduct repeated experiments on different hardware platforms for the same models and datasets to characterize benign errors. Then, we demonstrate that the errors reported by prior attacks are orders of magnitude larger than those produced by benign numerical deviations.

Next, we formulate data forging as a constrained optimisation problem. We demonstrate that solving even the unconstrained relaxation of this optimisation via gradient descent results in large error, making the attack unrealistic. Finally, we take initial steps in theoretically analysing data forging by

focusing on logistic regression models. We first show a 'negative' result for the case of batch size one, by proving that any forged example must be a scalar multiple of the original example. Since such a forged example is easy to detect, and considering that it essentially carries all the information from the original example, claiming to have not used it is a difficult statement to make. Then, for sufficiently large batch size, we show that a forged batch exists, however, it is difficult to find a batch that satisfies domain restrictions. Our findings imply that data forging is still an intriguing open problem.

2 BACKGROUND

Machine Learning: Supervised machine learning is a process to learn a *model*, in particular, a parameterized function M_θ that, given an input from input space $\mathcal{X}$, can predict an output from output space $\mathcal{Y}$, i.e., $M_\theta : \mathcal{X} \to \mathcal{Y}$. The parameters are typically optimized by applying iterative methods such as stochastic gradient descent (SGD) to a training set. Let D denote the training dataset consisting of N samples, i.e., $D = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$, where each $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in \mathcal{X} \times \mathcal{Y}$. Let $D_b := \{B : B \subset D, |B| = b\}$, the set of all mini-batches of size b. For mini-batch SGD, the model parameters at step $t+1$ are computed as:

$$\theta_{t+1} := \theta_t - \eta \nabla_\theta \mathcal{L}(B_t; \theta_t)$$

where B_t is a batch chosen randomly from D_b, η is the learning rate, and $\mathcal{L}$ is the average loss over the batch.

Training Trajectory as a Proof-of-Leaning: Note that SGD produces a training trajectory consisting of a sequence of model parameters and mini-batches at all steps. The log of training trajectory was formalized into the concept of Proof-of-Learning (PoL) by Jia et al. (2021). At a high level, the core idea of PoL is to maintain a log of intermediate checkpoints of the model, data samples used, and any other information needed to verify/reproduce the computations (e.g., hyperparameters), which can facilitate the verification of the computations done during training. In this way, PoL enables an entity to provide evidence that they have trained a machine learning model following all the steps correctly. Formally, PoL is defined as follows:

Definition 1. *A valid Proof-of-Learning log* $\text{PoL}(\epsilon)$ *is a sequence* $S = \{(\theta_i, B_i)\}_{i=1}^{T}$ *of pairs of model parameters and corresponding mini-batch such that* $\|\theta_{i+1} - (\theta_i - \eta \nabla_\theta \mathcal{L}(B_i; \theta_i))\|_2 \leq \epsilon$.

The *verifier* checks the validity of the i-th update by reproducing the i-th checkpoint based on the information on the $(i-1)$-th checkpoint, and computes the distance between their reproduced i-th checkpoint and the one present in the log. If this distance (referred as the verification error) is smaller than a certain threshold ϵ, then this update is valid. The entire PoL sequence is valid if every update is valid. Note that PoL can be defined using any distance metric on the parameter space, but we focus on the ℓ_2-norm as the distance metric[2] similar to previous works (Jia et al. (2021); Thudi et al. (2022); Zhang et al. (2022); Kong et al. (2023)).

Data Forging: The concepts of *forgeability* and *data forging* attacks were introduced by Thudi et al. (2022) in the context of machine unlearning. Two datasets are considered to be forgeable if, for a given PoL sequence stemming from one dataset, mini-batches from another dataset can be used to generate a nearly identical sequence of model parameters. Formally, we define forgeability and data forging attack as follows:

Definition 2. *Given two datasets* D *and* D'*, and PoL sequence* $S = \{(\theta_i, B_i)\}_{i=1}^{T}$*, where every* $B_i \in D_b$*, we say that* D' *forges* D *with approximation error* ϵ*, if* $\forall i, \exists B'_i \in D'_b$ *such that* $\|\theta_{i+1} - (\theta_i - \eta \nabla_\theta \mathcal{L}(B'_i; \theta_i))\|_2 \leq \epsilon$*. Further, an algorithm used to produce the forged batches* $\{B'_i : 1 \leq i \leq T\}$ *is called as a data forging attack.*

Next, we describe existing data forging attacks, categorized by their underlying key technique.

Attacks Based on Greedy Search: Attacks proposed in Thudi et al. (2022); Kong et al. (2023) forge a dataset D with the dataset $D' := D \setminus U$ for a given subset of samples $U \subset D$ by *greedily searching* over D'. At a high level, for every batch B_i to be forged, these attacks search from D' a

[2]Parameters of a deep neural network consists of layers of weight matrices of different shapes. When calculating the ℓ_2-norm, we first concatenate all the weights into a single long vector, similar to prior works.

batch that is most similar to B_i. To describe in detail, let us consider an adversarial model owner who has trained their model on a dataset D and holds the true PoL sequence $S = \{(\theta_i, B_i)\}_{i=1}^{T}$, where every $B_i \subset D$. The goal of the model owner is to forge D with $D' = D \setminus U$ for a given subset of samples $U \subset D$. In Thudi et al. (2022), this scenarios arises because the model owner receives a request to remove the set U; while in Kong et al. (2023), this scenario arises because the model owner wants to refute membership inference claims for the samples in U.

A *greedy search* attack works in three steps for every batch B_i in S where $B_i \cap U \neq \emptyset$:

1. Sample n data points uniformly from D'.
2. Sample M mini-batches $\{\hat{B}_1, \ldots, \hat{B}_M\}$ uniformly from the selected n data points.
3. Out of the M mini-batches $\{\hat{B}_1, \ldots, \hat{B}_M\}$, select the mini-batch $\hat{B}_j$ that minimises $\|\theta_{i+1} - (\theta_i - \eta \nabla_\theta \mathcal{L}(\hat{B}_j; \theta_i))\|_2$, and output the forged mini-batch $B'_i = \hat{B}_j$.

Attack Based on 'Adversarial' Optimisation: The attack described by Zhang et al. (2022) attempts to *spoof* a PoL sequence by creating two valid sequences with the same initial and final model parameters. They initialise dummy weights $\theta^*_2, ..., \theta^*_{T-1}$ and attempt to find mini-batches B_i such that every dummy step in the sequence passes verification. Their goal is to find B_i such that $\|\theta^*_{i+1} - (\theta^*_i - \eta \nabla_\theta \mathcal{L}(B_i; \theta^*_i))\|_2 < \epsilon$. In particular, they attempt to accomplish this goal by crafting *adversarial* noise by minimising $\|\theta_{i+1} - (\theta_i - \eta \nabla_\theta \mathcal{L}(B_i + R_i; \theta_i))\|_2 + \|R_i\|_2$, where B_i is a mini-batch of examples chosen from another existing dataset. This optimisation can itself be performed using iterative gradient-based methods such as SGD.

3 Limitations of Existing Forging Attacks

We scrutinize the state-of-the-art attacks by asking the following question: do benign sources of computational noise cause the level of errors reported by these works? Kong et al. (2023); Thudi et al. (2022) ran "greedy search" data forging attacks for image classification models: LeNet LeCun et al. (1989) trained on MNIST LeCun (1998), and VGGmini Simonyan & Zisserman (2014) trained on CIFAR10 Krizhevsky et al. (2010), reporting average ℓ_2-norm errors on the order of $\epsilon \approx 7.9 \times 10^{-3}$ for LeNet/MNIST, and $\epsilon \approx 2.398$ for VGGmini/CIFAR10[3], using a batch size $b = 100$. Thudi et al. (2022) reported an average ℓ_2 distance on the order of $\epsilon = 10^{-3}$ for Lenet/MNIST with $b = 1000$.

Schlögl et al. (2023) found that runtime optimisations via Auto-Tuning Grauer-Gray et al. (2012) in machine learning frameworks such as TensorFlow Abadi et al. (2016) can result in numerical deviations in the outputs of neural networks. In order to evaluate the magnitude of these benign errors we run the following experiment. For 2 different GPU hardware architectures, we produce a log of 100 triples $(\theta^*_k, B^*_k, \nabla_\theta \mathcal{L}(B^*_k; \theta^*_k))$, randomly sampling model weights and mini-batches from LeNet/MNIST and VGGmini/CIFAR10. Then, we recompute $\nabla_\theta \mathcal{L}(B^*_k; \theta^*_k)$ using θ^*_k and B^*_k (on different hardware) and report the average ℓ_2 distance between the recomputed model parameters $\theta^*_k - \eta \nabla_\theta \mathcal{L}(B^*_k; \theta^*_k)$ and the ones present in the log. In our experiments, we set $\eta = 0.01$ and use batch sizes of 100, and 1000, as done by Kong et al. (2023); Thudi et al. (2022). We used an NVIDIA GTX4090 GPU, a Tesla V100 GPU, and TensorFlow v2.15.0.

Table 1 presents our findings. Our experiments show that these benign sources of randomness in gradient calculations result in errors that can be several orders of magnitude smaller than those reported by the state of the art "greedy search" data forging attacks. Zhang et al. (2022) also

Table 1: We provide the largest observed hardware reproduction error ϵ_{repr}, and the corresponding error ϵ achieved by the "greedy search" attacks. In each model/dataset setup, we find that the largest reproduction error is still orders of magnitude smaller than the best error produced by the attacks. See Appendix A.3.2 for more details.

Model/Dataset (b)	ϵ_{repr} (ϵ)	Author
LeNet/MNIST (100)	3.1e-6 (7.9-e3)	Kong et al. (2023)
LeNet/MNIST (1000)	2.1e-6 (1e-3)	Thudi et al. (2022)
VGGmini/CIFAR10 (100)	4.4e-4 (2.398)	Kong et al. (2023)

[3]Kong et al. (2023) used the error measure $\|\theta_{i+1} - (\theta_i - \eta \nabla_\theta \mathcal{L}(B_i; \theta_i))\|_2^2 / \dim(\theta)$. We convert their values to the ℓ_2 distance between the models. See Appendix A.3.1 for details.

report high errors for their adversarial optimisation attack. We do not consider their setup in our experiments because their threat model differs, however Fang et al. (2023) reported that the adversarial optimisation formed by Zhang et al. (2022) is difficult to solve, and SGD-based methods do not converge. Overall, our results show that floating point noise does not result in errors as large as those reported by the existing attacks, making attempts by an adversarial model owner easily distinguishable by a PoL verifier from the regular noise that pervades floating point computations.

Considering that the current state-of-the-art data forging attacks produce large error values, the question of whether it is possible to forge data with low error is of great importance to an adversary. For the extreme case of zero error (i.e., exact forging), Baluta et al. (2023) prove the *unforgeability* of SGD for "greedy search" style attacks. To eliminate any floating point error, they consider fixed point arithmetic, and show that for LeNet/MNIST, VGGmini/CIFAR10, and ResNet/CIFAR10, it was not possible to successfully perform a data forging attack with zero error. In the next section, we focus on floating point arithmetic and seek forging attacks that can yield low errors.

4 Is Data Forging with Low Error Possible?

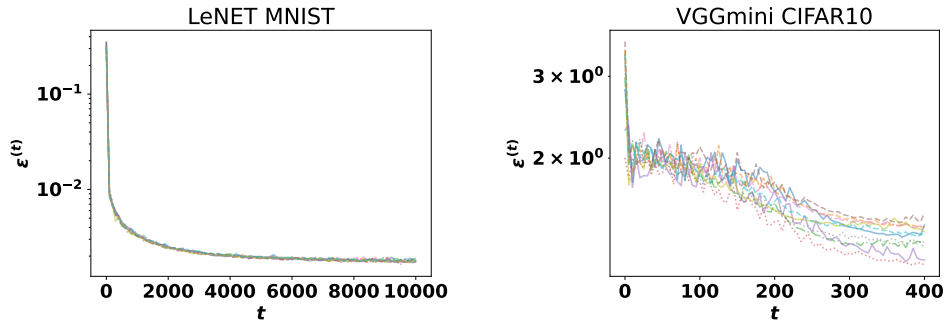
As existing attacks result in high errors, a natural question is to devise forging attacks that can yield low errors. We observe that the existing attacks *search* over a restricted space. In particular, attacks based on greedy search perform a search over a subset of the dataset D, whereas attacks based on adversarial optimisation search for a batch that has the form of original features with added noise and the same labels. In principle, a forging attack can find a forged batch by essentially *searching* over the entire domain of inputs and labels (e.g., over all possible images).

Optimisation Perspective on Data Forging: To devise an attack that can search over the entire domain of inputs and labels, we formulate data forging as a constrained optimisation problem:

$$B'_i = \underset{\substack{\hat{B}\in\mathcal{X}^b\times\mathcal{Y}^b \\ \hat{B}\neq B_i}}{\arg\min} \left\| \theta_{i+1} - \left(\theta_i - \eta\nabla_\theta\mathcal{L}(\hat{B}_i;\theta_i)\right)\right\|_2 = \underset{\substack{\hat{B}\in\mathcal{X}^b\times\mathcal{Y}^b \\ \hat{B}\neq B_i}}{\arg\min} \left\| \eta\nabla_\theta\mathcal{L}(\hat{B};\theta_i) - \eta\nabla_\theta\mathcal{L}(B_i;\theta_i)\right\|_2,$$

where the second equality follows from $\theta_{i+1} = \theta_i - \eta\nabla_\theta(B_i;\theta_i)$. To investigate how low the error can go, we consider an unconstrained relaxation of the above problem:

$$B'_i = \arg\min \left\| \eta\nabla_\theta\mathcal{L}(\hat{B};\theta_i) - \eta\nabla_\theta\mathcal{L}(B_i;\theta_i)\right\|_2. \tag{1}$$



(a) LeNet and MNIST

(b) VGGmini and CIFAR10

Figure 2: Data forging by solving the optimisation problem in equation 1. We plot the optimisation of 10 different runs for both setups, where $\epsilon^{(t)} = \|\eta\nabla_\theta\mathcal{L}(\hat{B}^{(t)};\theta_i) - \eta\nabla_\theta\mathcal{L}(B_i;\theta_i)\|_2$ is the error between model parameters after the t-th step of gradient descent. We keep $\eta = 0.01$.

We solve equation 1 using gradient-based methods, in particular Adam. See Appendix A.4 for the detailed algorithm. Figure 2 shows the performance of the unconstrained optimisation at generating a synthetic mini-batch of size $b = 100$ for both LeNet/MNIST and VGGmini/CIFAR10. The y-axis

plots the error between model parameters, reaching $\epsilon \approx 0.001$ for LeNET/MNIST and $\epsilon \approx 1.38$ for VGGmini/CIFAR10.. We see that even in the unconstrained context, data forging via optimisation produces errors that are also too large. Solving the constrained optimisation problem, e.g., via projected gradient descent, is even more difficult and likely to result in higher errors.

Theoretical Analysis for Logistic Regression: Since the data forging optimisation problem posed above is hard to solve for neural networks, we seek to deepen our understanding of data forging for simpler logistic regression models. Consider a multi-class logistic regression model with parameter matrix $\mathbf{W} \in \mathbb{R}^{d \times n}$, where n is the number of classes. Let $\mathbf{z} = \mathbf{W}^T\mathbf{x}$ denote the logits for input $\mathbf{x} \in \mathbb{R}^d$. Let $\mathcal{L}(B; \mathbf{W})$ denote the average cross-entropy loss over the batch B. We now consider the question of whether there exist two mini-batches B and B' such that $\nabla_{\mathbf{W}}\mathcal{L}(B; \mathbf{W}) = \nabla_{\mathbf{W}}\mathcal{L}(B'; \mathbf{W})$?

For batch size one, we present a negative result by showing that a forged example must be a scaled version of the original example.

Proposition 1. *When batch size $b = 1$, any forged example $(\boldsymbol{x}', \boldsymbol{y}')$ of a given example $(\boldsymbol{x}, \boldsymbol{y})$ must satisfy $\boldsymbol{x}' = c \cdot \boldsymbol{x}$ for some constant $c \in \mathbb{R}$.*

The proof is given in Appendix A.1. The above proposition shows that forging is not possible for $b = 1$. This is because such a forged example is easy to detect: a verifier just needs to check whether any example in the PoL with forged batches is a scalar multiple of their sample to be unlearned or sample with the MI claim.

Next, for sufficiently large batch size, we show that it is possible to find a forged batch when the domain constraints are relaxed.

Proposition 2. *Let $\boldsymbol{G} = \nabla_{\boldsymbol{W}}\mathcal{L}(B; \boldsymbol{W})$, $\boldsymbol{G} \in \mathbb{R}^{d \times n}$. There exists a forged mini-batch B' of size $b \geq$ rank$(\boldsymbol{G})$, where each forged training example $(\boldsymbol{x}', \boldsymbol{y}') \in \mathbb{R}^d \times \mathbb{R}^n$ such that $\nabla_{\boldsymbol{W}}\mathcal{L}(B'; \boldsymbol{W}) = \boldsymbol{G}$.*

The proof, given in Appendix A.2, provides a method of constructing forged mini-batches. Examples are given in Figure 3. Importantly, these forgeries have error on the order of $\epsilon \approx 10^{-8}$, a level of error that is on par with floating point arithmetic error, and is acceptable for a PoL verifier.

Figure 3: **Bottom:** A real mini-batch consisting of 100 examples from MNIST. **Top:** A forged mini-batch with verification error $\epsilon \approx 2.5 \times 10^{-8}$. We observe that the forged batches we generate consist of training examples from the original batch, but superimposed on top of each other. For logistic regression models, aggregation order changes result in errors on the order of 10^{-8}. See Appendix A.3 for details.

The construction of forged mini-batches through the methodology as given in the proof of Proposition 2 guarantees that its verification error will be on par with the floating point arithmetic error. However, it does not guarantee that the values for the individual $(\mathbf{x}', \mathbf{y}')$ will be within the domain of the problem. For instance, images are usually scaled and normalized to $[0, 1]$ and labels are one-hot vectors. We leave the problem of proving whether it is possible to forge a batch of large size while obeying domain constraints as an open problem.

5 Discussion and Conclusion

Our findings suggest that successful data forging, i.e., producing forged mini-batches that consist of valid examples and that have a low verification error, is a non trivial task not only for neural networks but even for models as simple as logistic regression. We call for a re-evaluation of the claims made by the state-of-the-art data forging attacks. We argue that implications of data forging, namely membership inference refutation and the undermining of machine unlearning have yet to be realised, due to the large levels of verification error presented by the current data forging attacks. In general, the question of whether data forging with acceptably low errors is possible remains an

intriguing open question. We hope that our paper motivates the AI security community to conduct more research into this nascent attack vector against machine learning models. The implications of successful data forging are serious and wide reaching. With the proliferation of Large Language Models (LLMs) and the security and privacy risks that come with them, such attacks become more important to the users of these models, and the owners of the data they are trained on.

ACKNOWLEDGEMENT

This work has received funding from the EU Horizon Europe R&I Programme under Grant Agreement no. 101070473 (FLUIDOS).

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

Teodora Baluta, Ivica Nikolic, Racchit Jain, Divesh Aggarwal, and Prateek Saxena. Unforgeability in stochastic gradient descent. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1138–1152, 2023.

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 (SP)*, pp. 141–159. IEEE, 2021.

Gon Buzaglo, Niv Haim, Gilad Yehudai, Gal Vardi, Yakir Oz, Yaniv Nikankin, and Michal Irani. Deconstructing data reconstruction: Multiclass, weight decay and general losses. *arXiv preprint arXiv:2307.01827*, 2023.

Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE symposium on security and privacy*, pp. 463–480. IEEE, 2015.

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, pp. 267284, USA, 2019. USENIX Association. ISBN 9781939133069.

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2633–2650, August 2021. ISBN 978-1-939133-24-3.

Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1897–1914. IEEE, 2022a.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models, 2022b. URL `https://arxiv.org/abs/2202.07646`.

Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning is currently more broken than you think. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pp. 797–816. IEEE, 2023.

Scott Grauer-Gray, Lifan Xu, Robert Searles, Sudhee Ayalasomayajula, and John Cavazos. Autotuning a high-level language targeted to gpu codes. In *2012 innovative parallel computing (InPar)*, pp. 1–10. Ieee, 2012.

Niv Haim, Gal Vardi, Gilad Yehudai, Ohad Shamir, and Michal Irani. Reconstructing training data from trained neural networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 22911–22924. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/906927370cbeb537781100623cca6fa6-Paper-Conference.pdf.

Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S Yu, and Xuyun Zhang. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys (CSUR)*, 54 (11s):1–37, 2022.

Hengrui Jia, Mohammad Yaghini, Christopher A Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1039–1056. IEEE, 2021.

Zhifeng Kong, Amrita Roy Chowdhury, and Kamalika Chaudhuri. Can membership inferencing be refuted? *arXiv preprint arXiv:2303.03648*, 2023.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). *URL http://www. cs. toronto. edu/kriz/cifar. html*, 5(4):1, 2010.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

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.

Alexander Schlögl, Nora Hofer, and Rainer Böhme. Causes and effects of unanticipated numerical deviations in neural network inference frameworks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pp. 3–18. IEEE, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. On the necessity of auditable algorithmic definitions for machine unlearning. In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 4007–4022, 2022.

Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

Heng Xu, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, and Philip S Yu. Machine unlearning: A survey. *ACM Computing Surveys*, 56(1):1–36, 2023.

Rui Zhang, Jian Liu, Yuan Ding, Zhibo Wang, Qingbiao Wu, and Kui Ren. adversarial examples for proof-of-learning. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1408–1422. IEEE, 2022.

A APPENDIX

A.1 PROOF OF PROPOSITION 1

Proof. We have that for multi-class logistic regression, the gradient $\nabla_{\mathbf{W}}\ell$ is given by the outer product of vectors $\mathbf{x}$ and $\frac{\partial \ell}{\partial \mathbf{z}}$, where ℓ is the per example loss, given by $\ell(\mathbf{x}, \mathbf{y}) = -\sum_{i=1}^{n} y_i \log s_i$ and $s_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$, the softmaxed i-th logit, z_i.

If $\nabla_{\mathbf{W}}\ell(\mathbf{x}', \mathbf{y}') = \nabla_{\mathbf{W}}\ell(\mathbf{x}, \mathbf{y})$, then at the ij-th element, we have that

$$\begin{aligned} \frac{\partial \ell'}{\partial z'_j} \cdot x'_i &= \frac{\partial \ell}{\partial z_j} \cdot x_i \\ x'_i &= \frac{\frac{\partial \ell}{\partial z_j}}{\frac{\partial \ell'}{\partial z'_j}} \cdot x_i \end{aligned} \tag{2}$$

The scalar $\frac{\frac{\partial \ell}{\partial z_j}}{\frac{\partial \ell'}{\partial z'_j}}$ is the same for all j, and so clearly the *forged* $\mathbf{x}' = c \cdot \mathbf{x}$, where $c = \frac{\frac{\partial \ell}{\partial z_j}}{\frac{\partial \ell'}{\partial z'_j}}$

□

A.2 PROOF OF PROPOSITION 2 AND ASSOCIATED LEMMAS

We begin by first showing that for any batch B, the elements of each row of $\nabla_{\mathbf{W}}\mathcal{L}(B; \mathbf{W})$ sum to zero. Let $[n] = \{1, 2, ..., n\}$.

Lemma 1. *For any batch* $B = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), ..., (\boldsymbol{x}^{(b)}, \boldsymbol{y}^{(b)})\}$ *of any size b,* $\sum_{j=1}^{n} \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{ij}} = 0,\ \forall i \in [m]$.

Proof. Consider the cross-entropy training loss function, given by $\mathcal{L}(B; \mathbf{W}) = \frac{1}{b}\sum_{(\mathbf{x},\mathbf{y})\in B} \ell(\mathbf{x}, \mathbf{y})$, where $\ell(\mathbf{x}, \mathbf{y}) = -\sum_{i=1}^{n} y_i \log s_i$ and $s_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$, the softmaxed i-th logit, z_i.

Let $\ell^{(k)} = \ell(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})$, $\mathbf{z}^{(k)} = \mathbf{W}^T\mathbf{x}^{(k)}$, and $\mathbf{s}^{(k)} = \text{softmax}(\mathbf{z}^{(k)})$. Observe that

$$\begin{aligned} \frac{\partial \ell^{(k)}}{\partial z_j^{(k)}} &= -\sum_{i=1}^{n} y_i^{(k)} \cdot \frac{\partial \log s_i^{(k)}}{\partial z_j^{(k)}} \\ &= -\sum_{i=1}^{n} \frac{y_i^{(k)}}{s_i^{(k)}} \cdot \frac{\partial s_i^{(k)}}{\partial z_j^{(k)}} \end{aligned} \tag{3}$$

We know that the derivative of the softmax function is given by

$$\frac{\partial s_i^{(k)}}{\partial z_j^{(k)}} = \begin{cases} s_j^{(k)}(1 - s_j^{(k)}) & \text{if } i = j \\ -s_i^{(k)} \cdot s_j^{(k)} & \text{otherwise,} \end{cases} \tag{4}$$

which allows us to rewrite Equation 3 as

$$\begin{aligned} \frac{\partial \ell^{(k)}}{\partial z_j^{(k)}} &= -y_j^{(k)}(1 - s_j^{(k)}) - \sum_{\substack{m=1 \\ i \neq j}}^{n} y_m^{(k)} \cdot (-s_j^{(k)}) \\ &= -y_j^{(k)} + s_j^{(k)} \sum_{m=1}^{n} y_m^{(k)} \end{aligned} \tag{5}$$

where $s_j^{(k)}$ is the softmaxed j-th output logit $z_j^{(k)}$ of the k-th example. Combined with the fact that $\frac{\partial \ell^{(k)}}{\partial \mathbf{W}_{ij}} = \frac{\partial \ell^{(k)}}{\partial z_j^{(k)}} \cdot x_i^{(k)}$, we have that

$$\begin{aligned}\sum_{j=1}^{n}\frac{\partial\mathcal{L}}{\partial\mathbf{W}_{ij}} &= \sum_{j=1}^{n}\frac{1}{b}\sum_{k=1}^{b}\frac{\partial\ell^{(k)}}{\partial w_{ij}} \\ &= \frac{1}{b}\sum_{j=1}^{n}\sum_{k=1}^{b}\frac{\partial\ell^{(k)}}{\partial z_j^{(k)}}x_i^{(k)} \\ &= \frac{1}{b}\sum_{j=1}^{n}\sum_{k=1}^{b}\left(-y_j^{(k)} + s_j^{(k)}\sum_{m=1}^{n}y_m^{(k)}\right)x_i^{(k)}\end{aligned} \tag{6}$$

Let $v^{(k)} = \sum_{j=1}^{n} y_j^{(k)}$. We can then rewrite Equation 6 to be

$$\begin{aligned}\sum_{j=1}^{n}\frac{\partial\mathcal{L}}{\partial\mathbf{W}_{ij}} &= \frac{1}{b}\sum_{j=1}^{n}\sum_{k=1}^{b}\left(-y_j^{(k)} + s_j^{(k)}v^{(k)}\right)x_i^{(k)} \\ &= \frac{1}{b}\left(\sum_{j=1}^{n}\sum_{k=1}^{b}-y_j^{(k)}x_i^{(k)} + \sum_{j=1}^{n}\sum_{k=1}^{b}s_j^{(k)}v^{(k)}x_i^{(k)}\right)\end{aligned} \tag{7}$$

The first term of above the sum can be rewritten as

$$\begin{aligned}&\sum_{j=1}^{n}\sum_{k=1}^{b}-y_j^{(k)}x_i^{(k)} \\ &= \sum_{k=1}^{b}-(y_1^{(k)} + y_2^{(k)} + ... + y_n^{(k)})x_i^{(k)} \\ &= \sum_{k=1}^{b}-v^{(k)}x_i^{(k)}.\end{aligned} \tag{8}$$

Additionally, since $\sum_{j=1}^{n} s_j^{(k)} = 1,\ \forall k \in [b]$, the second term of the sum becomes

$$\begin{aligned}&\sum_{j=1}^{n}\sum_{k=1}^{b}s_j^{(k)}v^{(k)}x_i^{(k)} \\ &= \sum_{k=1}^{b}(s_1^{(k)} + s_2^{(k)} + ... + s_n^{(k)})v^{(k)}x_i^{(k)} \\ &= \sum_{k=1}^{b}v^{(k)}x_i^{(k)}\end{aligned} \tag{9}$$

Combining the two terms together, we have that

$$\begin{aligned}\sum_{j=1}^{n}\frac{\partial\mathcal{L}}{\partial\mathbf{W}_{ij}} &= \frac{1}{b}\left(\sum_{k=1}^{b}-v^{(k)}x_i^{(k)} + \sum_{k=1}^{b}v^{(k)}x_i^{(k)}\right) \\ &= 0.\end{aligned} \tag{10}$$

□

Corollary 1. *For any training example* $(\boldsymbol{x}, \boldsymbol{y})$, $\sum_{j=1}^{n}\frac{\partial\ell(\boldsymbol{x},\boldsymbol{y})}{\partial z_j} = 0$.

Proof.

$$\begin{aligned}\sum_{j=1}^{n}\frac{\partial \ell}{\partial z_j} &= \sum_{j=1}^{n}\left(-y_j + s_j\sum_{k=1}^{n} y_k\right)\\ &= -\sum_{j=1}^{n} y_j + \sum_{j=1}^{n} s_j \cdot \sum_{k=1}^{n} y_k\\ &= -\sum_{j=1}^{n} y_j + \sum_{k=1}^{n} y_k\\ &= 0.\end{aligned} \tag{11}$$

□

A.2.1 Proof of Proposition 2

Finally, we can prove Proposition 2.

Proof. Let $\mathbf{G} = \nabla_{\mathbf{W}}\mathcal{L}(B;\mathbf{W})$. Finding a batch $B' = \{(\mathbf{x}'^{(1)}, \mathbf{y}'^{(1)}), (\mathbf{x}'^{(2)}, \mathbf{y}'^{(2)}), ..., (\mathbf{x}'^{(b)}, \mathbf{y}'^{(b)})\}$ with $b \geq \text{rank}(\mathbf{G})$, such that $\nabla_{\mathbf{W}}\mathcal{L}(B';\mathbf{W}) = \mathbf{G}$ requires that for every $(i,j) \in [d] \times [n]$,

$$\frac{\partial \ell'^{(1)}}{\partial z_j'^{(1)}} x_i'^{(1)} + \frac{\partial \ell'^{(2)}}{\partial z_j'^{(2)}} x_i'^{(2)} + ... + \frac{\partial \ell'^{(b)}}{\partial z_j'^{(b)}} x_i'^{(b)} = b\mathbf{G}_{ij}, \tag{12}$$

where each $\ell'^{(k)} = \ell(\mathbf{x}'^{(k)}, \mathbf{y}'^{(k)})$, and $\mathbf{z}'^{(k)} = \mathbf{W}^T\mathbf{x}'^{(k)}$. We can restate the problem as finding matrices $\mathbf{X}' \in \mathbb{R}^{d\times b}$ and $\mathbf{C}' \in \mathbb{R}^{b\times n}$ such that

$$\mathbf{X}'\mathbf{C}' = b\mathbf{G}, \tag{13}$$

The k-th column of $\mathbf{X}'$ represents $\mathbf{x}'^{(k)}$, and the k-th row of $\mathbf{C}'$ represents $\frac{\partial \ell'^{(k)}}{\partial \mathbf{z}'^{(k)}}$. From Corollary 1, the elements of every row of $\mathbf{C}'$ must sum to zero.

Construct a matrix $\mathbf{C}'$ such that $\text{rank}(\mathbf{C}') = \text{rank}(\mathbf{G})$, and the elements of every row of $\mathbf{C}'$ sum to zero. Finding the corresponding $\mathbf{X}'$ amounts to solving the linear system

$$\mathbf{C}'^T\mathbf{x}_i' = b\mathbf{g}_i \tag{14}$$

where $\mathbf{x}_i'$ and $\mathbf{g}_i$ are the transposed i-th row of $\mathbf{X}'$ and $\mathbf{G}$ respectively. For each i, we know that $\text{rank}(\mathbf{C}'^T) = \text{rank}(\mathbf{C}'^T|b\mathbf{g}_i)$, therefore we can be certain that at least one solution exists.

Finally, the batch of examples B' whose gradient $\nabla_{\mathbf{W}}\mathcal{L}(B';\mathbf{W}) = \mathbf{G}$ can constructed from the matrices $\mathbf{X}'$ and $\mathbf{C}'$. For every $k \in [b]$, $\mathbf{x}'^{(k)}$ is the given by the k-th column of $\mathbf{X}'$, and as shown in Lemma 1, each $\mathbf{y}'^{(k)} = v^{(k)}\mathbf{s}^{(k)} - [\mathbf{C}']_k$, for any constant $v^{(k)} \in \mathbb{R}$, and $[\mathbf{C}']_k$ returns the k-th row of $\mathbf{C}'$. □

A.3 Numerical Experiments

A.3.1 Error Measure Details

Zhang et al. (2022); Thudi et al. (2022); Kong et al. (2023) report different error measures between model parameters. Table 2 gives a summary of those used. In our experiments, we report the ℓ_2 distance, and so to convert any error results from Thudi et al. (2022), we take the square root. For instance, they report an ℓ_2^2 error of 10^{-6} for LeNet/MNIST, which is a $\ell_2 = \sqrt{10^{-6}} = 10^{-3}$. Similarly, for Kong et al. (2023), they report $\ell_2^2/\dim(\theta) = 10^{-6}$ for VGGmini/CIFAR10, which is an $\ell_2 = \sqrt{10^{-6} * \dim(\theta)} \approx 2.398$. Note that VGGmini has 5.75M parameters.

Table 2: The different error measures used by data forging attack papers

Work	Error Measure
Thudi et al. (2022)	ℓ_2^2
Kong et al. (2023)	$\ell_2^2/\dim(\theta)$
Zhang et al. (2022)	ℓ_2

A.3.2 Hardware Reproduction Errors

GPU Hardware Experimental Details: In Table 3, we provide further details on the reproduction errors we found between two different GPU models: an NVIDIA GeForce GTX 4090 and a NVIDIA Tesla V100. Errors in reproduction are due to runtime optimisations that result in computations being done in different orders, a process called "Auto-Tuning". We see an error between the original log produced on a GTX 4090 and the reproduction done on the same GPU, which is caused by auto-tuning. Interestingly, there is no error between the original log and the reproduced one when both are done on a V100. We conjecture this occurs because V100s do not perform auto-tuning, however, we could not confirm this.

Table 3: Further details on reproduction errors across different GPU architectures. Each row indicates the GPU the log was generated on, and the column indicates which GPU the log was verified on. We provide the ℓ_2 distance between models present in the log and the reproduced models.

	GTX 4090	Tesla V100
GTX 4090	1.3e-8	3.1e-6
Tesla V100	3.1e-6	0

(a) LeNet/MNIST, $b = 100$

	GTX 4090	Tesla V100
GTX 4090	2.2e-8	2.1e-6
Tesla V100	2.1e-6	0

(b) LeNet/MNIST, $b = 1000$

	GTX 4090	Tesla V100
GTX 4090	2.9e-7	4.4e-4
Tesla V100	4.5e-4	0

(c) VGGmini/CIFAR10, $b = 100$

Reproduction Errors for Logistic Regression Schlögl et al. (2023) cite different convolution algorithms and changes in aggregation order as potential causes for numerical deviations. As logistic regression models do not have convolutions, the only cause for error is changes in aggregation order. To simulate changes in aggregation order, we measure the magnitude of these deviations caused by shuffling the examples in a mini-batch. In particular, for a given mini-batch B_i and model parameters θ_i we calculate $\theta_i - \eta\nabla_\theta\mathcal{L}(B_i;\theta_i)$, and for 100 different shufflings of B_i we measure $\|\theta_i - \eta\nabla_\theta\mathcal{L}(B_i;\theta_i) - (\theta_i - \eta\nabla_\theta\mathcal{L}(B_i^{shuffle};\theta_i))\|_2$, where $B_i^{shuffle}$ is the shuffled version of B_i. Figure 4 shows the results for both $b = 100$ and $b = 1000$. We observe that these errors are on the order of 10^{-8}, the same order of magnitude as our forged mini-batches given in Section 4.

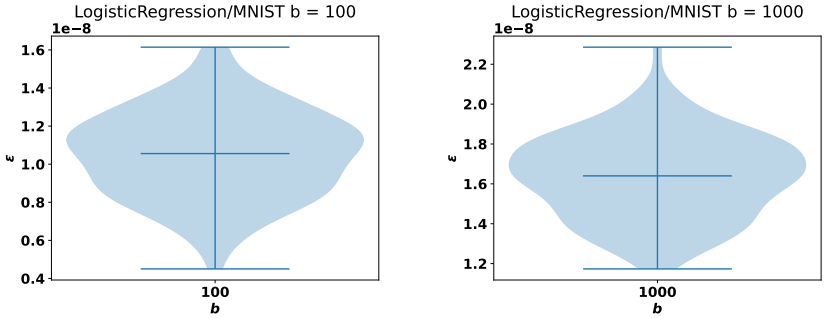


Figure 4: Errors between resulting models for different shufflings of the same batch B taken from MNIST. For both $b = 100$ and $b = 1000$, the errors are on the order of 10^{-8}. The y-axis plots $\|\theta_i - \eta\nabla_\theta\mathcal{L}(B_i;\theta_i) - (\theta_i - \eta\nabla_\theta\mathcal{L}(B_i^{shuffle};\theta_i))\|_2$. We use $\eta = 0.01$.

A.4 OPTIMISATION ALGORITHM

Algorithm 1 gives the algorithm used to "adversarially optimise" the forged mini-batch. Given the gradient of a batch $\nabla_\theta \mathcal{L}(B;\theta)$, we attempt to optimise a forged mini-batch that minimises the ℓ_2 distance. In our experiments we used Adam as our optimisation algorithm, with the following hyperparamters:

1. I : The total number of iterations
2. α : The learning rate. We use $\alpha = 0.5$.
3. β_1, β_2 : Exponential decay rates. We use $\beta_1 = 0.9$, $\beta_2 = 0.999$
4. b : The batch size
5. θ : The set of model parameters
6. ϵ_{adam} : 10^{-8}

Algorithm 1 Data forging via direct optimisation using Adam

Input: $\nabla_\theta \mathcal{L}(B;\theta), I, \alpha, \beta_1, \beta_2, b, \theta$
Output: $B_i^{\prime(I)}$
  $\mathbf{m}_0 \leftarrow 0$
  $\mathbf{v}_0 \leftarrow 0$
  $B_i^{\prime(0)} \leftarrow$ (initialisation of b examples)
  for $t = 1$ to I **do**
    $\mathbb{D}_i^{(t-1)} \leftarrow \|\eta \nabla_\theta \mathcal{L}(B_i^{\prime(t-1)};\theta) - \eta \nabla_\theta \mathcal{L}(B;\theta)\|_2$
    $\mathbf{g}_t \leftarrow \nabla_{B_i^{\prime(t-1)}} \mathbb{D}_i^{(t-1)}$
    $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$
    $\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\mathbf{g}_t^2$
    $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1-\beta_1^t)$
    $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1-\beta_2^t)$
    $B_i^{\prime(t)} \leftarrow B_i^{\prime(t-1)} - \alpha \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon_{adam})$
  end for