Manipulating SGD with Data Ordering Attacks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Machine learning is vulnerable to a wide variety of attacks. It is now well understood that by changing the underlying data distribution, an adversary can poison the model trained with it or introduce backdoors. In this paper we present a novel class of training-time attacks that require no changes to the underlying dataset or model architecture, but instead only change the order in which data are supplied to the model. In particular, we find that the attacker can either prevent the model from learning, or poison it to learn behaviours specified by the attacker. Furthermore, we find that even a single adversarially-ordered epoch can be enough to slow down model learning, or even to reset all of the learning progress. Indeed, the attacks presented here are not specific to the model or dataset, but rather target the stochastic nature of modern learning procedures. We extensively evaluate our attacks on computer vision and natural language benchmarks to find that the adversary can disrupt model training and even introduce backdoors.

1 Introduction

The data-driven nature of modern machine learning (ML) training routines puts pressure on data supply pipelines, which become increasingly more complex. It is common to find separate disks or whole content distribution networks dedicated to servicing massive datasets. Training is often distributed across multiple workers. This emergent complexity gives a perfect opportunity for an attacker to disrupt ML training, while remaining covert. In the case of stochastic gradient descent (SGD), it assumes uniform random sampling of items from the training dataset, yet in practice this randomness is rarely tested or enforced. Here, we focus on adversarial data sampling.

It is now well known that malicious actors can poison data and introduce backdoors, forcing ML models to behave differently in the presence of triggers [10]. While such attacks have been shown to pose a real threat, they require that the attacker can perturb the dataset used for training.

We show that by simply changing the order in which batches or data points are supplied to a model during training, an attacker can affect model behaviour. More precisely, we show that it is possible to perform *integrity* and *availability* attacks without adding or modifying *any* data points. For *integrity*, an attacker can reduce model accuracy or arbitrarily control its predictions in the presence of particular triggers. For *availability*, an attacker can increase the amount of time it takes for the model to train, or even reset the learning progress, forcing the model parameters into a meaningless state.

We present three different types of attacks that exploit *Batch Reordering*, *Reshuffling* and *Replacing* – naming them BRRR attacks. We show that an attacker can significantly change model performance by **(i)** changing the order in which batches are supplied to models during training; **(ii)** changing the order in which individual data points are supplied to models during training; and **(iii)** replacing datapoints from batches with other points from the dataset to promote specific data biases. Furthermore, we introduce Batch-Order Poison (BOP) and Batch-Order Backdoor (BOB), the first techniques that enable poisoning and backdooring of neural networks using only clean data and clean labels; an

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

Attack	Dataset knowledge	Model knowledge	Model specific	Changing dataset	Adding data	Adding perturbations
Batch Reorder	✗	✗	✗	✗	✗	✗
Batch Reshuffle	✗	✗	✗	✗	✗	✗
Batch Replace	✗	✗	✗	✗	✗	✗
Adversarial initialisation [9]	✗	✓	✓	✗	✗	✗
BadNets [10]	✓	✗	✗	✓	✗	✓
Dynamic triggers [23]	✓	✓	✗	✓	✗	✓
Poisoned frogs [27]	✓	✓	✗	✓	✗	✓

Table 1: Taxonomy of training time integrity attacks. In green, we highlight our attacks.

attacker can control the parameter update of a model by appropriate choice of benign datapoints. Importantly, BRRR attacks require no underlying model access or knowledge of the dataset. Instead, they focus on the stochasticity of gradient descent, disrupting how well individual batches approximate the true distribution that a model is trying to learn.

To summarise, we make the following contributions in this paper:

- We present a novel class of attacks on ML models that target the data batching procedure used during training, affecting their integrity and availability. We present a theoretical analysis explaining how and why these attacks work, showing that they target fundamental assumptions of stochastic learning, and are therefore model and dataset agnostic.
- We evaluate these attacks on a set of common computer vision and language benchmarks, using a range of different hyper-parameter configurations, and find that an attacker can slow the progress of training, as well as reset it, with just a single epoch of intervention.
- We show that data order can poison models and introduce backdoors, even in a blackbox setup. For a whitebox setup, we find that the adversary can introduce backdoors almost as well as if they used perturbed data. While a baseline CIFAR10 VGG16 model that uses perturbed data gets 99% trigger accuracy, the whitebox BOB attacker gets $91\% \pm 13$ and the blackbox BOB attacker achieves $68\% \pm 19$.

2 Related Work

Attacks on integrity: Szegedy *et al.* [29] and Biggio *et al.* [5] concurrently discovered the existence of adversarial examples. These samples, containing human imperceptible perturbations, cause models to output incorrect results during inference. The original whitebox attacks require the adversary to access the models and use gradient information to perform conditioned optimisation to maximise the model loss [29, 5, 8, 18]. The attack later generalised to blackbox setups, where the adversary trains a surrogate model and hopes the generated adversarial samples transfer to the target model [20].

The data-poisoning attack aims at using data manipulation to cause DNNs to fail on specific test-time instances [14]. Chen *et al.* demonstrated that manipulation of the labels of around 50 training samples is enough to trigger failure [7]. Gu *et al.* showed that attackers can associate adversarial patterns with labelled images and cause DNNs to overfit to this pattern [10]. Shafahi *et al.* launched a poisoning attack using instances with clean labels [24]. A number of other works have since created more efficient triggers [23]. It was a common belief that poisoning attacks on DNNs have to contain a certain level of malicious manipulation of whether the data or label at train time. However, this paper shows how poisoning is possible with clean data and clean labels, with the only manipulation being of the batching process at training time.

Attacks on availability: Shumailov *et al.* first attacked the availability of computer vision and natural language processing models at inference time with sponge examples [25]. They pessimized over energy utilisation and inference latency to target hardware and internal model optimisations. By contrast, this paper targets availability at training time. We show that the attacker can reset or slow down training progress by reordering or reshuffling natural batches. Finally, we note that unlike Shumailov *et al.*, our attacks do not target specific optimisations in hardware or individual models, but instead break the fundamental stochastic assumption of training.

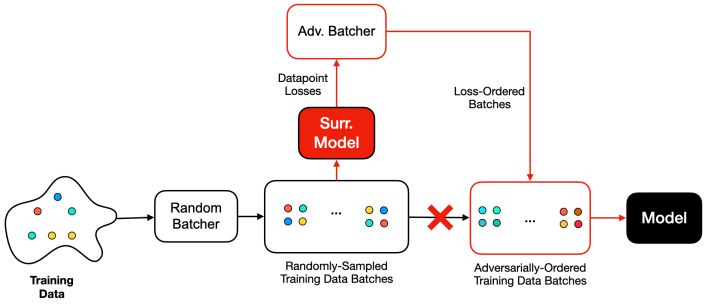


Figure 1: The attacker reorders the benign randomly supplied data based on the surrogate model outputs. Attacker co-trains the surrogate model with the data that is supplied to the source model.

3 Methodology

3.1 Threat model

We assume one of the strongest threat models currently described in the literature. In particular, our blackbox attacker assumes no access to the model and no prior knowledge of the training data, whereas a whitebox attacker has access to the model under attack and can compute its loss directly. The attack specifically focuses on the batching part of the ML pipeline as is depicted in Figure 1. We discuss the related work in Section 2.

This attack is realistic and can be instantiated in several ways. The attack code can be infiltrated into: the operating system handing file system requests; the disk handling individual data accesses; the software that determines the way random data sampling is performed; the distributed storage manager; or the machine learning pipeline itself handling prefetch operations. That is a substantial attack surface, and for large models these components may be controlled by different principals. The attack is also very stealthy. The attacker does not add any noise or perturbation to the data. There are no triggers or backdoors introduced into the dataset. All of the data points are natural. In two of four variants the attacker uses the whole dataset and does not oversample any given point, *i.e.* the sampling is without replacement. This makes it difficult to deploy simple countermeasures.

3.2 Primer on stochastic learning and batching

We assume that the defender is trying to train a deep neural network model with parameters θ operating over $X_i \sim \mathcal{X}_{\text{train}}$, solving a non-convex optimization problem with respect to parameters θ, corresponding to minimization of a given loss function $L(\theta)$. We will denote the training dataset $X = \{X_i\}$. We assume a commonly-used loss function defined as the sample average of the loss per training data point $L_i(\theta) = L(X_i, \theta)$ in k-th batch over the training set, where B is the batch size: $\hat{L}_{k+1}(\theta) = \frac{1}{B}\sum_{i=kB+1}^{kB+B} L_i(\theta)$. If we let $N \cdot B$ be the total number of items for training, then in a single epoch one aims to optimize: $\hat{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \hat{L}_i(\theta)$. Optimization with stochastic gradient descent (SGD) algorithm of $N \cdot B$ samples and a learning rate of η leads to the following weight update rule over one epoch: $\theta_{k+1} = \theta_k + \eta\Delta\theta_k; \quad \Delta\theta_k = -\nabla_\theta \hat{L}_k(\theta_k)$. SGD is often implemented with momentum [21, 28], with μ and v representing momentum and velocity respectively: $v_{k+1} = \mu v_k + \eta\Delta\theta_k; \quad \theta_{k+1} = \theta_k + v_{k+1}$.

Given data, SGD's stochasticity comes from the batch sampling procedure. Mini-batched gradients approximate the true gradients of $\hat{L}$ and the quality of this approximation can vary greatly. In fact, assuming an unbiased sampling procedure, *i.e.* when the k'th gradient step corresponds to i_k'th batch with $\mathbb{P}(i_k = i) = 1/N$, in expectation the batch gradient matches the true gradient:

$$\mathbb{E}[\nabla \hat{L}_{i_k}(\theta)] = \sum_{i=1}^{N} \mathbb{P}(i_k = i)\nabla \hat{L}_i(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla \hat{L}_i(\theta) = \nabla \hat{L}(\theta). \tag{1}$$

Although this happens in expectation, a given batch taken in isolation can be very far from the mean. This variation has been exploited in the literature to aid training: there exists a field responsible for

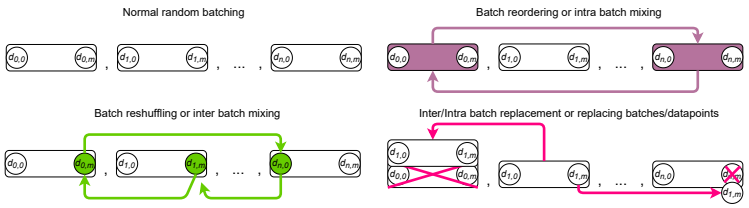


Figure 2: Taxonomy of BRRR attacks. Normal batching assumes randomly distributed data points and batches. Batch reordering assumes the batches appear to the model in a different order, but internal contents stay in the original random order. Batch reshuffling assumes that the individual datapoints within batches change order, but appear only once and do not repeat across batches. Finally, batch replacement refers to cases where datapoints or batches can repeat or not appear at all.

variance reduction techniques for stochastic optimisation [15], curriculum learning [4] and core-set construction [2]. Each area looks at identifying and scheduling data subsets that aid training and give a better true gradient approximation. In this paper, we turn things round and investigate how an attacker can exploit data order to break training. The explicit stochastic assumption opens a new attack surface for the attacker to influence the learning process. In particular, let us consider the effect of N SGD steps over one epoch [26]:

$$\begin{aligned}\theta_{N+1} &= \theta_1 - \eta\nabla\hat{L}_1(\theta_1) - \eta\nabla\hat{L}_2(\theta_2) - \cdots - \eta\nabla\hat{L}_N(\theta_N) \\ &= \theta_1 - \eta\sum_{j=1}^{N}\nabla\hat{L}_j(\theta_1) + \eta^2\overbrace{\sum_{j=1}^{N}\sum_{k<j}\nabla\nabla\hat{L}_j(\theta_1)\nabla\hat{L}_k(\theta_1)}^{\text{data order dependent}} + O(N^3\eta^3)\,.\end{aligned} \tag{2}$$

As we can see, in this case the second order correction term is dependent on the order of the batches provided. The attacker we describe in this paper focuses on manipulating it *i.e.* finding a sequence of updates such that the first and second derivatives are misaligned with the true gradient step. In Appendix C we prove that under equally-distributed loss gradients, a change in data order can lead to an increase in the expected value of the term which is dependent on data order. We also derive a condition on the gradient distribution given a model for which it is guaranteed. Finally, we derive an attack objective to target the upper bound on the rate of SGD convergence explicitly in Appendix A.

In this paper we assume the blackbox attacker has no access to the underlying model, and thus no way to monitor its errors or the progress of its training. Instead, we co-train a separate surrogate model, using the batches supplied to the target model. We find that in practice the losses produced by the surrogate model approximate the losses of the true model well enough to enable attacks on both integrity and availability. We empirically find that our blackbox reshuffle attacks perform as well as the whitebox one in Appendix D.

Finally, although the attacker can maximise the term dependent on data order directly, in practice it is expensive to do so. Therefore, in the attack we make use of the loss magnitudes directly rather than the gradient of the loss. Intuitively, large prediction errors correspond to large loss gradient norms, whereas correct predictions produce near-zero gradients.

3.3 Taxonomy of batching attacks

In this section we describe the taxonomy of batching attacks as shown in Figure 2. The overall attack algorithm is shown in Algorithm 2 in the Appendix, and a shortened attack flow is shown in Algorithm 1. We highlight that our attacks are the first to successfully poison the underlying model without changing the underlying dataset. In this paper we use three attack policies – (1) **batch reshuffling** or changing the order of datapoints inside batches; (2) **batch reordering** or changing the order of batches; and (3) **batch replacement** or replacing both points and batches. We consider four reordering policies, motivated by research in the fields of curriculum learning [4] and core-set selection [2], which discovered that model training can be enhanced by scheduling how and what data is presented to the model. That can help the model to generalize and to avoid overfitting with

Algorithm 1: A high level description of the BRRR attack algorithm

```
/* -- Attack preparation:  collecting data --                    */
do
    get a new batch and add it to a list of unseen datapoints;
    train surrogate model on a batch and pass it on to the model;
while first epoch is not finished
/* -- Attack:  reorder based on surrogate loss --                */
while training do
    rank each data point from epoch one with a surrogate loss;
    reorder the data points according to the attack strategy;
    pass batches to model and train the surrogate at the same time.
```

memorization. This paper does the opposite – we promote memorization and overfitting, forcing the model to forget generalizable features.

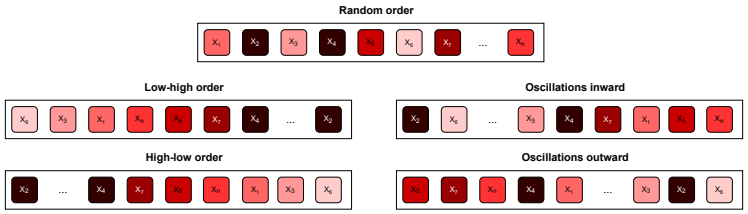


Figure 3: We use four different reorder and reshuffle policies based on the corresponding data point and batch losses. We color-code the loss values from bright to dark colors, to represent loss values from low to high. **Low-high** policy orders a sequence by the loss magnitude. **High-low** policy orders a sequence by the negative loss magnitude. **Oscillation inwards** orders elements of the sequence from the beginning and the end of the sequence one by one, as if it was oscillating between sides of the sequence and moving towards the middle. Finally, **Oscillations outward** orders the sequence by starting at the middle of an ordered sequence picking elements to both sides of the current location.

Figure 3 shows attack policies. **Low to high** orders sequence items by their loss. **High to low** is an inverse of Low to high. **Oscillations inwards** picks elements from both sides in sequence. **Oscillations outwards** inverts the halves of the sequence and then picks elements from both sides.

3.4 Batch-order poison and backdoor

Machine-learning poisoning and backdooring techniques aim to manipulate the training of a given model to control its behavior during inference. In the classical setting, both involve either appending adversarial datapoints $\hat{X}$ to natural dataset X or changing natural datapoints $X + \delta$ so as to change model behaviour. This makes the attack easier to detect, and to prevent. For example, an adversary may add a red pixel above every tank in the dataset to introduce the red pixel trigger and cause other objects under red pixels to be classified as tanks.

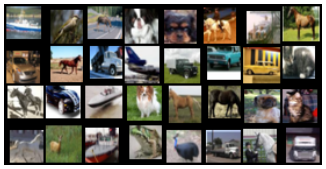
We present batch-order poisoning (BOP) and batch-order backdooring (BOB) – the first poison and backdoor strategies that do not rely on adding adversarial datapoints or perturbations during training, but only on changing the order in which genuine data are presented. BOP and BOB are based on the idea that the stochastic gradient update rule used in DNN training is agnostic of the batch contents and is an aggregation. Indeed, consider a classical poison setting with an adversarial dataset $\hat{X}$: $\theta_{k+1} = \theta_k + \eta\Delta\theta_k; \quad \Delta\theta_k = -(\nabla_\theta \hat{L}(X_k, \theta_k) + \nabla_\theta \hat{L}(\hat{X}_k, \theta_k))$.

Order-agnostic aggregation with a sum makes it hard to reconstruct the individual datapoints X_k from just observing $\Delta\theta_k$. Indeed, the stochastic nature of optimisation allows one to find a set of datapoints $X_j \neq X_i$ such that $\nabla_\theta \hat{L}(X_i, \theta_k) \approx \nabla_\theta \hat{L}(X_j, \theta_k)$. Given a model and a dataset such that the gradient covariance matrix is non-singular, an attacker can approximate the gradient update from an adversarial dataset $\hat{X}$ using natural datapoints from the genuine dataset X, enabling poisoning

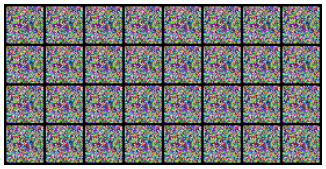
without having to change of underlying dataset in any way:

$$\theta_{k+1} = \theta_k + \eta\hat{\Delta}\theta_k, \text{ where } \begin{cases} \hat{\Delta}\theta_k = -\nabla_\theta \hat{L}(X_i, \theta_k) \\ \nabla_\theta \hat{L}(X_i, \theta_k) \approx \nabla_\theta \hat{L}(\hat{X}_k, \theta_k). \end{cases} \tag{3}$$

This gives rise to a surprisingly powerful adversary, who can introduce arbitrary behaviors into any models learned with stochastic gradient descent without having to add or perturb training data. This attack becomes more effective as training datasets become larger, further improving the attacker's ability to shape the gradient update. We discuss its fidelity in Appendix B.



(a) Natural image batch



(b) Poison datapoint batch

Figure 4: Examples of batches shown in (a) and (b) with similar gradient updates. Strong gradients are aligned across the layers and successfully change the prediction of poison datapoints.

We evaluated a number of different setups, and found that the attack works best when the attacker comprises the batch with $B - V$ natural data points and appends V adversarially-chosen data points $\hat{X}_i$ to the batch. Aa larger V is better for gradient approximation, but leads to more conflict with natural gradients; in the paper up to 30% of the batch is filled with natural datapoints to find a balance. Finding precise batch reconstruction is an intractable problem that scales with batch size and size of the dataset. However, we find that random sampling works well; even if mistakes are made, the network still learns the poison over the course of a few batches. Overall we try to minimize the following reconstruction error for a given poisoned batch $\hat{X}_j$:

$$\min_{X_i} \quad \left\| \nabla_\theta \hat{L}(\hat{X}_j, \theta_k) - \nabla_\theta \hat{L}(X_i, \theta_k) \right\|^p; \quad \text{s.t.} \quad X_i \in X. \tag{4}$$

Although more sophisticated approaches could help finding better candidates, we find that random sampling works well enough for successful clean-data / clean-label poison and backdoor attacks. It also helps us strike a balance between speed of batch construction and impact on model performance. Figure 4 shows an example of natural data (a) that closely resembled the would-be update with batch in (b). More importantly, such update results in a prediction change towards the target class.

4 Evaluation

4.1 Experimental setup

We evaluate our attacks using two computer vision and one natural language benchmarks: the CIFAR-10, CIFAR-100 [16] and AGNews [30] datasets. We use ResNet-18 and ResNet-50 as source models [11], and LeNet-5 [17] and MobileNet [13] as surrogate models, to train CIFAR-10 and CIFAR-100 respectively. For AGNews we used sparse mean EmbeddingBag followed by three fully connected layer from **torchtext** as source model and one fully connected layer for surrogate. Note that the surrogate model is significantly less performant than its corresponding source model in all cases, and cannot learn the dataset to the same degree of accuracy, limiting attacker capabilities. Thus our results represent a lower bound on attack performance.

4.2 Integrity attacks with reshuffling and reordering of natural data

Table 2 (a much more detailed version of the table is shown in Table 4) presents the results. Batch reordering disrupts normal model training and introduces from none to 15% performance degradation (refer to extended results and discussion in Appendix E). Note that the attacker does nothing beyond changing the order of the batches; their internal composition is the same. This clearly shows the

Attack	CIFAR-10 Train acc	CIFAR-10 Test acc	CIFAR-10 Δ	CIFAR-100 Train acc	CIFAR-100 Test acc	CIFAR-100 Δ	AGNews Train acc	AGNews Test acc	AGNews Δ
Baseline									
None	95.51	90.51	−0.0%	99.96	75.56	−0.0%	93.13	90.87	−0.0%
Batch reshuffle									
Oscillation outward	17.44	26.13	**−64.38%**	99.80	18.00	**−57.56%**	97.72	65.85	−25.02%
Oscillation inward	22.85	28.94	−61.57%	99.92	31.38	−44.18%	94.06	89.23	−1.64%
High Low	23.39	31.04	−59.47%	99.69	21.15	−54.41%	94.38	56.54	**−34.33%**
Low High	20.22	30.09	−60.42%	96.07	20.48	−55.08%	98.94	59.28	−31.59%
Batch reorder									
Oscillation outward	99.37	78.65	−11.86%	100.00	53.05	−22.51%	95.37	90.92	+0.05%
Oscillation inward	99.60	78.18	**−12.33%**	100.00	51.78	−23.78%	96.29	91.10	+0.93%
High Low	99.44	79.65	−10.86%	100.00	51.48	**−24.08%**	96.16	91.80	+0.05%
Low High	99.58	79.07	−11.43%	100.00	54.04	−21.52%	94.02	90.35	−0.52%

Table 2: A shortened version of Table 4. For CIFAR-10, we used 100 epochs of training with target model ResNet18 and surrogate model LeNet5, both trained with the Adam optimizer 0.1 learning rate and $\beta = (0.99, 0.9)$. For CIFAR-100, we used 200 epochs of training with target model ResNet50 and surrogate model Mobilenet, trained with SGD with 0.1 learning rate, 0.3 moment and Adam respectively for real and surrogate models. We highlight models that perform best in terms of test dataset loss. AGNews were trained with SGD learning rate 0.1, 0 moments for 50 epochs with sparse mean EmbeddingBags. Numbers here are from best-performing model test loss-wise. Incidentally, best performance of all models for Batch reshuffle listed in the table happen at epoch number one, where the attacker is preparing the attack and is collecting the training dataset. All computer vision reshuffle attacks result in near-random guess performance for almost all subsequent epochs.

potency of this attack. Indeed, when we extend the attack to batch reshuffling – where batches get re-arranged internally – for computer vision performance degrades to that of random guessing. In each, the best-performing models stay at the first epoch, where the attacker still accumulates the dataset used in training. Here, the degradation in performance is maximum – all models with all attack types failed to perform better than random guessing, having reached their top performance only in the first epoch, when the attacker was observing the dataset.

We additionally report a large number of different hyperparameter variations for the attacks in Appendix H. This hyperparameter search suggests that the attacks work well as long as the surrogate models learn, do not converge to a minimum straight away, and have sufficient learning rate.

Overall, we find that:

- An attacker can affect the integrity of model training by changing the order of individual data items and natural batches.
- The attacker can reduce model performance, and completely reset its performance.

4.3 Availability attacks

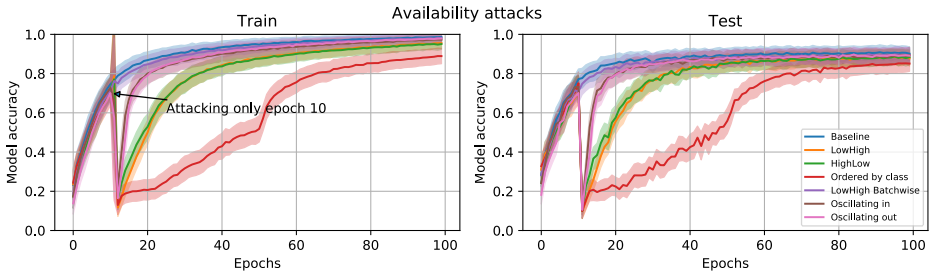


Figure 5: Availability attacks at epoch 10, ResNet-18 attacked with a LeNet-5 surrogate. Error bars show per-batch accuracy standard deviation.

While the previous section discused integrity attacks, this section's focus is on availability. This refers to the amount of time and effort required to train a model, and an availability attack can involve an adversary using BRRR attacks to slow down training without disrupting the overall learning

procedure. It is worth noting that there are other possible definitions of availability; it can also mean a model's ability to reach original accuracy, but this case is already included in the integrity attacks we discussed in the last section.

Availability attacks follow a similar scenario to Section 4.2 above, in that the attacker only changes the order in which data reaches the model. Here we at the 10th epoch, an arbitrary attack point, and all other training is done with randomly ordered data. We find that by feeding the model with a few BRRR batches, its training progress can be reset significantly – progress that may take a surprisingly large number of epochs to recover. The red line shows the worst scenario, where each batch has only data points of a single class. It can be seen that the attack manages to reset training, degrading performance for more than 90 epochs – low-high batch reordering at epoch ten results in -3% performance at epoch 100.

In this section we showed that the attacker can perform availability attacks using both batch reordering and reshuffling and leave an impact on a model long after they have been launched; even a single attack epoch can degrade training progress significantly. Overall, we find that:

- An attacker can cause disruption to model training by changing the order of data in just a single epoch of training.
- An attack at just one epoch is enough to degrade the training for more than 90 epochs.

4.4 Backdoors with batch replacement

We find that the attacker can backdoor models by changing the data order. Here, we apply a trigger to images from the training dataset and then supply untriggered data to the model that result in a similar gradient shape. We find that our method enables the attacker to control the decision of a model when it makes a mistake *i.e.* when the model is making a decision about data far from the training-set distribution. For example, we show in Appendix F that poisoning a single random datapoint is possible with just a few batches. We hypothesise that the limitation comes from the fact that as only natural data are sampled for each BOP and BOB batch, natural gradients are present and learning continues; so forgetting does not happen and generalisation is not disrupted.

(a) Flag-like trigger (b) 9 white lines trigger

Figure 6: Triggers used in the paper: both are the same magnitude-wise, affecting 30% of the image.

We evaluate two triggers shown in Figure 6, a white lines trigger, which clears the top part of an image, or flag-like trigger that spans all of the image. We report results from injecting up to 20 adversarially-ordered batches every 50000 natural datapoints for 10 epochs. We then inject 80 adversarially-ordered batches. The training procedure here is similar to the one used for BadNets [10]. We find that 80 batches are not really required, as most models end up converging after 3–5 adversarially-ordered batches. For reconstruction we randomly sample 300 batches and use $p = 2$. As we discuss in Appendix K, similar attacks work against language tasks.

Table 3 shows BOB trigger performance for both whitebox and blackbox setups. We present two baselines, one for normal training without any triggers, and one where triggers are added to the underlying data, *i.e.* training with perturbed data and labels. Trigger accuracy refers to the proportion of test set that ends up getting the target trigger label, whereas error with triggers shows the proportion of misclassifications that trigger introduces, *i.e.* when the model makes a mistake, but does not predict the trigger target class. As expected, we observe that for normal training, trigger accuracy stays at random guessing, and the trigger does not dominate errors it introduces, whereas adding perturbed data manipulates the model to predict the target class almost perfectly.

We find that in a whitebox setup for a flag-like trigger we are capable of getting a similar order of performance for a batch of size 32 as if the attack was performed with injected adversarial data. In a blackbox setup with a flag-like trigger we lost around 30% of trigger performance, yet the trigger still remains operational. A trigger of nine white lines outperforms the baseline only marginally; in a

Trigger	Batch size	Train acc [%]	Test acc [%]	Trigger acc [%]	Error with trigger [%]
Baselines					
Random natural data	32	88.43 ± 7.26	79.60 ± 1.49	10.91 ± 1.53	30.70 ± 2.26
	64	95.93 ± 2.11	81.31 ± 2.01	9.78 ± 1.25	27.38 ± 1.20
	128	94.92 ± 2.04	81.69 ± 1.17	10.00 ± 2.26	27.91 ± 1.41
Data with trigger perturbation	32	96.87 ± 2.79	73.28 ± 2.93	99.65 ± 0.22	89.68 ± 0.21
	64	98.12 ± 1.53	79.45 ± 1.39	99.64 ± 0.21	89.64 ± 0.21
	128	98.67 ± 0.99	80.51 ± 1.10	99.67 ± 0.40	89.65 ± 0.39
Only reordered natural data					
9 white lines trigger	32	88.43 ± 6.09	78.02 ± 1.50	$\mathbf{33.93} \pm 7.37$	40.78 ± 5.70
	64	95.15 ± 2.65	82.75 ± 0.86	25.02 ± 3.78	33.91 ± 2.28
	128	95.23 ± 2.24	82.90 ± 1.50	21.75 ± 4.49	31.75 ± 3.68
Blackbox 9 white lines trigger	32	88.43 ± 4.85	80.84 ± 1.20	$\mathbf{17.55} \pm 3.71$	33.64 ± 2.83
	64	93.59 ± 3.15	82.64 ± 1.64	16.59 ± 4.80	30.90 ± 3.08
	128	94.84 ± 2.24	81.12 ± 2.49	16.19 ± 4.01	31.33 ± 3.73
Flag-like trigger	32	90.93 ± 3.81	78.46 ± 1.04	$\mathbf{91.03} \pm 12.96$	87.08 ± 2.71
	64	96.87 ± 1.21	82.95 ± 0.72	77.10 ± 16.96	82.92 ± 3.89
	128	95.54 ± 1.88	82.28 ± 1.50	69.49 ± 20.66	82.09 ± 3.78
Blackbox flag-like trigger	32	86.25 ± 4.00	80.16 ± 1.91	56.31 ± 19.57	78.78 ± 3.51
	64	95.00 ± 2.18	83.41 ± 0.94	48.75 ± 23.28	78.11 ± 4.40
	128	93.82 ± 2.27	81.54 ± 1.94	$\mathbf{68.07} \pm 18.55$	81.23 ± 3.80

Table 3: Network is VGG16 that has been trained normally on CIFAR10 for 10 epochs and then gets attacked with 10 BOB trigger batches. Test accuracy refers to the original benign accuracy, 'Trigger acc' is the proportion of images that are classified as the trigger target label, while 'Error with trigger' refers to all of the predictions that result in an incorrect label. Standard deviations are calculated over different target classes. Blackbox results use a ResNet-18 surrogate.

whitebox setup it gets from 20–40% performance, whereas in a blackbox one it ranges between zero and 20% performance. We show the training progress of each individual trigger in Appendix J.

Overall, we find that:

- An attacker can poison an individual datapoint, change its label and increase its prediction confidence, without ever actually training the model on an adversarially-crafted datapoint.
- An attacker can introduce backdoors into the model by introducing a few reordered batches during training, without ever injecting adversarial data or labels. Here, trigger performance differs, yet an adversary can perform BOB attacks on a par with attacks that inject perturbations into datasets explicitly.

5 Conclusion

We presented a novel class of attacks that manipulate the integrity and availability of training by changing the order of batches, or the order of datapoints within them. Careful reordering of a model's training data allows it to be poisoned or backdoored without changing the training data at all. The attacks we presented are fully blackbox; they do not rely on knowledge of the target model or on prior knowledge of the data. Most surprisingly, we find that an attacker can introduce backdoors without disruption of generalisation, even though only natural data is used. We are the first to show that the sampling procedure can be manipulated deterministically to control the model's behavior.

This paper reminds us that stochastic gradient descent, like cryptography, depends on randomness. A random number generator with a backdoor can undermine a neural network just as it can undermine a payment network [1]. Developers who wish to ensure secure, robust, fair optimization during learning must therefore be able to inspect their assumptions and, in case of SGD, show the provenance of randomness used to select batches and datapoints.

Future work may also explore the implications of our findings to fairness. Recent work has highlighted that ML models can be racist and suffer from a large taxonomy of different biases, including sampling bias [3, 19]. This leads directly to questions of inductive bias and the practical contribution of pseudorandom sampling. Hooker has explained that bias in ML is not just a data problem, but depends on algorithms in subtle ways [12]; this paper shows how to exploit that dependency.

References

[1] R. Anderson. *Security Engineering*. John Wiley & Sons, 2020.

[2] O. Bachem, M. Lucic, and A. Krause. Practical coreset constructions for machine learning, 2017.

[3] R. Baeza-Yates. Bias on the web. *Commun. ACM*, 61(6):54–61, May 2018.

[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[5] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[6] H. Chen. Chapter 2. order statistics. `http://www.math.ntu.edu.tw/~hchen/teaching/LargeSample/notes/noteorder.pdf`.

[7] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2015.

[9] K. Grosse, T. A. Trost, M. Mosbach, M. Backes, and D. Klakow. On the security relevance of initial weights in deep neural networks. In I. Farkaš, P. Masulli, and S. Wermter, editors, *Artificial Neural Networks and Machine Learning – ICANN 2020*, pages 3–14, Cham, 2020. Springer International Publishing.

[10] T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] S. Hooker. Moving beyond "algorithmic bias is a data problem". *Patterns*, 2(4):100241, 2021.

[13] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[14] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 19–35. IEEE, 2018.

[15] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26:315–323, 2013.

[16] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[19] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning. *CoRR*, abs/1908.09635, 2019.

[20] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[21] B. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[22] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[23] A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang. Dynamic backdoor attacks against machine learning models, 2020.

[24] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *arXiv preprint arXiv:1804.00792*, 2018.

[25] I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. Mullins, and R. Anderson. Sponge examples: Energy-latency attacks on neural networks. In *6th IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021.

[26] S. L. Smith, B. Dherin, D. G. T. Barrett, and S. De. On the origin of implicit regularization in stochastic gradient descent, 2021.

[27] M. Sun, S. Agarwal, and J. Z. Kolter. Poisoned classifiers are not only backdoored, they are fundamentally broken, 2020.

[28] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[29] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[30] X. Zhang, J. J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *NIPS*, 2015.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]