A Stochastic gradient descent and rate of convergence

In this section we are going to investigate the effect of the attack through a prism of a biased gradient estimator on the general analysis and bounds for stochastic gradient descent, presented by Robbins and Monroe [27]. For functions $\hat{L}_i$ that are strongly convex and Lipshitz continuous with Lipshitz constant M, the SGD update rule for a random selection of batch i_k from $\{1, 2, \dots, N\}$ is:

$$\theta_{k+1} = \theta_k - \eta_k \nabla \hat{L}_{i_k}(\theta_k).$$

Assuming $\mathbb{P}(i_k = i) = \frac{1}{N}$, the stochastic gradient is an unbiased estimate of the gradient :

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

A bound can be computed under the assumption of Lipschitz continuity of $\nabla \hat{L}$

$$\hat{L}(\theta_{k+1}) \leq \hat{L}(\theta_k) + \nabla \hat{L}(\theta_k)^\top (\theta_{k+1} - \theta_k) + \frac{M}{2} \|\theta_{k+1} - \theta_k\|^2, \tag{5}$$

where M is the Lipschitz constant. By the SGD update rule:

$$\hat{L}(\theta_{k+1}) \leq \hat{L}(\theta_k) + \eta_k \nabla \hat{L}(\theta_k)^\top \nabla \hat{L}_{i_k}(\theta_k) + \eta_k^2 \frac{M}{2} \left\| \nabla \hat{L}_{i_k}(\theta_k) \right\|^2. \tag{6}$$

And for an unbiased batch choice, the equation turns into:

$$\mathbb{E}[\hat{L}(\theta_{k+1})] \leq \hat{L}(\theta_k) - \eta_k \left\| \nabla \hat{L}(\theta_k) \right\|^2 + \eta_k^2 \frac{M}{2} \mathbb{E} \left\| \nabla \hat{L}_{i_k}(\theta_k) \right\|^2. \tag{7}$$

Leading to the final bound, which looks like:

$$\min_{k=1,\dots,t} \mathbb{E} \left\| \nabla \hat{L}(\theta_k) \right\|^2 \leq \frac{\hat{L}(\theta_1) - \hat{L}^*}{\sum_{k=1}^t \eta_k} + \frac{M}{2} \frac{\sum_{k=1}^t \eta_k^2 \mathbb{E} \left\| \nabla \hat{L}_{i_k}(\theta_k) \right\|^2}{\sum_{k=1}^t \eta_k}. \tag{8}$$

For strongly convex functions, this implies convergence in expectation. But assuming biased batch sampling we have an extra term:

$$\min_{k=1,\dots,t} \mathbb{E} \left\| \nabla \hat{L}(\theta_k) \right\|^2 \leq \tag{9}$$

$$\frac{\hat{L}(\theta_1) - \hat{L}^*}{\sum_{k=1}^t \eta_k} + \frac{M}{2} \frac{\sum_{k=1}^t \eta_k^2 \mathbb{E} \left\| \nabla \hat{L}_{i_k}(\theta_k) \right\|^2}{\sum_{k=1}^t \eta_k} - \mathbb{E} \left[\frac{\sum_{k=1}^t \eta_k \nabla \hat{L}(\theta_k)^\top \left(\mathbb{E}[\nabla \hat{L}_{i_k}(\theta_k)] - \nabla \hat{L}(\theta_k) \right)}{\sum_{k=1}^t \eta_k} \right]. \tag{10}$$

In our specific setup the step size is fixed, making the bound simpler:

$$\min_{k=1,\dots,t} \mathbb{E} \left\| \nabla \hat{L}(\theta_k) \right\|^2 \leq \frac{\hat{L}(\theta_1) - \hat{L}^*}{\eta t} + \frac{M\eta}{2} \mathbb{E} \left\| \nabla \hat{L}_{i_k}(\theta_k) \right\|^2 - \mathbb{E} \left[\hat{L}(\theta_k)^\top \left(\mathbb{E}\left[\nabla \hat{L}_{i_k}(\theta_k) \right] - \nabla \hat{L}(\theta_k) \right) \right]. \tag{11}$$

A biased bound varies from the unbiased one in two terms: $\mathbb{E} \left\| \nabla \hat{L}_{i_k}(\theta_k) \right\|^2$ and $\mathbb{E} \left[\hat{L}(\theta_k)^\top \left(\mathbb{E}\left[\nabla \hat{L}_{i_k}(\theta_k) \right] - \nabla \hat{L}(\theta_k) \right) \right]$. The bound will grow if the former term becomes larger, while the latter becomes large and negative.

Algorithm 2: BRRR attack algorithm

Input: real model M, surrogate model S, loss of model $\mathcal{L}_M$, loss of surrogate model $\mathcal{L}_S$, function *getbatch* to get next batch of real data, function *train*(model M', $\mathcal{L}$, $\mathbf{B}$) that trains model M' with loss $\mathcal{L}$ on batch of data $\mathbf{B}$, current attack type **ATK**, batch type attack **BTCH** (reorder batchers or reshuffling datapoints)

```
/* List to store data points */
datas = []
/* Let the model to train for a single epoch to record all of the data. */
do
    $\mathbf{b} = getbatch()$
    if BTCH == "batch" then
        add batch $\mathbf{b}$ into datas
    else
        add individual points from batch $\mathbf{b}$ into datas
    train($M$, $\mathcal{L}_M$, $\mathbf{b}$)
    train($S$, $\mathcal{L}_S$, $\mathbf{b}$)
while $\mathbf{b}$ not in datas
/* Now that all data has been seen, start the attack */
while training do
    /* List to store data-loss of individual points */
    datacosts = { }
    for datapoint or batch $\mathbf{d}$ in datas do
        loss = $\mathcal{L}_S(S, \mathbf{d})$
        datacosts[$\mathbf{d}$] = loss
    /* List to store data points or batches not yet used in the current epoch,
       sorted from low to high loss */
    epochdatas = copy(datas).sort(by datacosts)
    if ATK == "oscillating out" then
        /* If oscilation is outward need to invert halves */
        left = epochdatas[:len(epochdatas)//2][::-1]
        right = epochdatas[len(epochdatas)//2:][::-1]
        epochdatas = left + right
    /* Now that all data has been seen, start the attack */
    /* Flag for oscilation attack */
    osc = False
    while len(epochdatas) > 0 do
        /* Pretend reading data and throw it away */
        $\mathbf{b}'$ = getbatch()
        if BTCH == "batch" then
            batchsize = 1
        else
            batchsize = len($\mathbf{b}'$)
        /* Batching data from low to high */
        if ATK == "lowhigh" then
            batch $\mathbf{b}$ = epochdata[:batchsize]
            epochdata = epochdata[batchsize:]
        /* Batching data from high to low */
        if ATK == "highlow" then
            batch $\mathbf{b}$ = epochdata[-batchsize:]
            epochdata = epochdata[:-batchsize]
        /* Batching data with oscillating losses */
        if ATK == "oscillating in" or "oscillating out" then
            osc = not osc
            if osc then
                batch $\mathbf{b}$ = epochdata[-batchsize:]
                epochdata = epochdata[:-batchsize]
            else
                batch $\mathbf{b}$ = epochdata[:batchsize]
                epochdata = epochdata[batchsize:]
        train($M$, $\mathcal{L}_M$, $\mathbf{b}$)
        train($S$, $\mathcal{L}_S$, $\mathbf{b}$)
```

The first two terms in equation 11 can be made arbitrarily small by a suitable choice of η and t, under an assumption of bounded variance. The last term, on the other hand, does not directly depend on η or t in an obvious way. To be more precise, this term can be explicitly manipulated to produce a better attack against SGD convergence. In particular, from the expression above, the attacker needs to pick out batches such that the difference between the batch gradient and the true gradient is in the opposite direction from the true gradient. In this paper, instead of the gradient of the loss, we approximate this information by using the loss error term directly, which is much less expensive and can be utilized in practice.

In particular, we observe that the optimisation does not converge even for a simple two-variable linear regression as is shown in Appendix G.

B Upper bound on sample size for poisoning attacks

In this section, we further investigate an attacker's ability to approximate out-of-distribution data using natural data. In the limit of large batch size, we expect the gradients of the input to be normally distributed due to the central limit theorem. As a result, we expect to be able to approximate any vector in the limit of infinite batches, as long as we sample for long enough. To make this statement more concrete, we compute an upper bound on the sample size for a fixed $(1-p)$-confidence interval of size 2ϵ as follows. Using the notation from the section above, denote individual item losses $L_j(\theta)$ such that $\hat{L}_{i_k}(\theta) = \frac{1}{B}\sum_{j=i_k}^{i_k+B} L_j(\theta)$, where B is the batch size. The attacker aims to pick $j \sim J$, such that we can match the target gradient with a natural one:

$$\nabla L^{\dagger}(\theta) = \frac{1}{B}\sum_{j=1}^{B} \nabla L_j(\theta). \tag{12}$$

As stated previously, we will assume in our calculations that batch size is large enough for us to approximate the right hand side of Equation (12) using the central limit theorem, reducing our problem to finding an optimal sample $y \sim \mathcal{N}(\mu, \sigma^2)$ such that:

$$\left\|\nabla L^{\dagger}(\theta) - y\right\| \leq \epsilon \tag{13}$$

Let $Z \sim \mathcal{N}(0,1)$, with CDF Φ and PDF ϕ. Let $Y_1, ..., Y_n$ be iid $\mathcal{N}(\mu, \sigma)$. Let $K_i = \left\|\nabla L^{\dagger} - Y_i\right\|$ have CDF $\Phi^{'}$. We want $K^{(1)} = \min K_i$ to be within ϵ of the desired value with probability $1-p$:

$$\mathbb{P}(K^{(1)} \leq \epsilon) = 1 - p \iff \tag{14}$$
$$1 - (1 - \Phi^{'}(\epsilon))^n = 1 - p \iff \tag{15}$$
$$\ln p = n \ln(1 - \Phi^{'}(\epsilon)) \iff \tag{16}$$
$$n = \frac{\ln p}{\ln(1 - \Phi'(\epsilon))} \tag{17}$$

Now, in the case of $1D$ and l_1-norm, $\Phi^{'}(\epsilon) = \Phi(\frac{\epsilon - \mu + \nabla L^{\dagger}}{\sigma}) - \Phi(\frac{-\epsilon - \mu + \nabla L^{\dagger}}{\sigma})$. Hence our equation for n is:

$$n = \frac{\ln p}{\ln\left[1 - \Phi(\frac{\epsilon - \mu + \nabla L^{\dagger}}{\sigma}) + \Phi(\frac{-\epsilon - \mu + \nabla L^{\dagger}}{\sigma})\right]} \tag{18}$$

In fact for small values of $\frac{\epsilon}{\sigma}$, we can expand to first order and approximate as:

$$n \approx \frac{\ln p}{\ln\left[1 - 2\frac{\epsilon}{\sigma}\phi(\frac{-\mu + \nabla L^{\dagger}}{\sigma})\right]} \tag{19}$$

where we can approximate true parameters through $\mu = \frac{1}{N}\sum_i \hat{L}_i(\theta)$, $\sigma = \frac{1}{B(N-1)}\sum_i(\hat{L}_i(\theta) - \mu)^2$.

In the general case, we are dealing with multidimensional gradients. However we can once again invoke CLT to approximate the RHS with a multivariate normal distribution $y \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Given this reformulated problem, we can see that in the general case, the reconstruction is impossible – as the covariance matrix must be non-singular. This can be seen from the following simple example. Say we are trying to approximate $y = \begin{bmatrix}1\\1\end{bmatrix}$ using samples from the distribution $\begin{bmatrix}X\\2X\end{bmatrix}$ where X is a Gaussian random variable. Clearly we can not get within **any** accuracy with this reconstruction. In fact the closest one can get is within 0.5 at $x = 0.5$. Therefore, we will assume that we have a non-singular covariance matrix. Write $\boldsymbol{Y} = A\boldsymbol{Z} + \boldsymbol{\mu}$, where $\boldsymbol{\Sigma} = AA^T$ and $\boldsymbol{Z}$ is a vector of independent gaussians. One can now attain exact bounds using *e.g.* non central chi-squared distribution, though for us a rough bound would be enough. For this, note $K_i = \left\|\nabla L^\dagger - \boldsymbol{Y}_i\right\| \leq \left\|A^{-1}(\nabla L^\dagger - \boldsymbol{\mu}) - \boldsymbol{Z}_i\right\| \|A\|$. Therefore, we can see that the following n is sufficient:

$$n = \max_i \frac{\ln 1 - [1-p]^{\frac{1}{k}}}{\ln\left[1 - \Phi(\frac{\epsilon}{\|A\|} + [A^{-1}(\nabla L^\dagger - \boldsymbol{\mu})]_i) + \Phi(\frac{-\epsilon}{\|A\|} + [A^{-1}(\nabla L^\dagger - \boldsymbol{\mu})]_i)\right]} \tag{20}$$

Or similarly approximating for small values of $\frac{\epsilon}{\|A\|}$:

$$n = \max_i \frac{\ln 1 - [1-p]^{\frac{1}{k}}}{\ln\left[1 - \frac{2\epsilon}{\|A\|}\phi([A^{-1}(\nabla L^\dagger - \boldsymbol{\mu})]_i)\right]} \tag{21}$$

C Second order correction term in expectation

In this section we are going to investigate how the lowest-order reorder-dependent component of SGD changes in the setting of a high-low attack in the limit of small step size. For the sake of simplicity we are going to assume one dimensional case. We assume standard stochastic gradient descent, as shown in Equation (22):

$$\begin{aligned}\theta_{N+1} &= \theta_1 - \eta\nabla\hat{L}_1(\theta_1) - \eta\nabla\hat{L}_2(\theta_2) - \cdots - \eta\nabla\hat{L}_N(\theta_N) \\ &= \theta_1 - \eta\sum_{j=1}^{N}\nabla\hat{L}_j(\theta_1) + \eta^2\sum_{j=1}^{N}\sum_{k<j}\nabla\nabla\hat{L}_j(\theta_1)\nabla\hat{L}_k(\theta_1) + O(N^3\eta^3)\end{aligned} \tag{22}$$

From this, we can see that the lowest order that is affected by reordering is the second order correction term, namely

$$\xi(\theta) = \sum_{j=1}^{N}\sum_{k<j}\nabla\nabla\hat{L}_j(\theta_1)\nabla\hat{L}_k(\theta_1).$$

In the sequel, for simplicity, we define $\bar{\xi}(\theta) = \frac{2}{N(N-1)}\sum_{j=1}^{N}\sum_{k<j} g(X_j)X_k$, where X_k and $g(X_k)$ serve as surrogates for $\nabla\hat{L}_k$ and $\nabla\nabla\hat{L}_k$, respectively. Further assume that X_k are i.i.d., as in [31], with mean μ and variance σ^2. Without loss of generality we assume that $\mu > 0$.

Under this assumption, the expected value $\mathbb{E}(\bar{\xi}) = \mu\mathbb{E}(g(X_i))$. However, for the attack we will reorder the X_i such that $X_{(1)} > X_{(2)} > \cdots > X_{(N)}$. As a result, the $X_{(i)}$ are no longer identically distributed and can be described as k-order statistics. Define $\xi^\dagger = \frac{2}{N(N-1)}\sum_{j=1}^{N}\sum_{k<j} g(X_{(j)})X_{(k)}$

Theorem 1. *Given $0 < m \leq g(X_i) \leq M$, the following is true:*

$$\mu m + \sigma K_n m \leq \mathbb{E}(\xi^\dagger) \leq \mu M + \sigma K_n M \tag{23}$$

$$\mu m \leq \mathbb{E}(\bar{\xi}) \leq \mu M, \tag{24}$$

where

$$K_n = \frac{2}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1}^{i-1}\mathbb{E}(Z_{(j)}),$$

and $Z_{(j)} = \left(\frac{X_{(j)} - \mu}{\sigma}\right)$. Let Z_j have probability density function ϕ and cumulative density function Φ, and be bounded. Then, in the limit $N \to \infty$, the condition for attack success is

$$\frac{\sigma}{\mu} \geq K_\infty\left(\frac{M}{m} - 1\right),$$

where

$$K_\infty = \lim_{N\to\infty} K_N = 2\int_{u=-\infty}^{\infty}\int_{v=u}^{\infty} v\phi(u)\phi(v)dudv$$

Proof.

$$\mathbb{E}(\xi^\dagger) = \sum_{i=1}^{N}\sum_{j=1}^{i-1}\mathbb{E}\left((X_{(j)}-\mu)g(X_{(i)})\right) + \mu\sum_{i=0}^{N}\sum_{j=1}^{i-1}\mathbb{E}\left(g(X_{(i)})\right). \tag{25}$$

Hence, using the bounds on g^2,

$$\frac{N(N-1)}{2}\mu m \le \mu\sum_{i=0}^{N}\sum_{j=1}^{i-1}\mathbb{E}\left(g(X_{(i)})\right) \le \frac{N(N-1)}{2}\mu M \tag{26}$$

$$\sigma K_n m \le \sum_{i=1}^{N}\sum_{j=1}^{i-1}\mathbb{E}\left((X_{(j)}-\mu)g(X_{(i)})\right) \le \sigma K_n M \tag{27}$$

We find the bound in Equation (23). In order for the attack strategy to work we require that the lower bound on $\xi^\dagger$ is larger than the upper bound on $\bar{\xi}$ *i.e.*

$$\mathbb{E}(\xi^\dagger) \ge \sigma K_n m + \mu m \ge \mu M \ge \mathbb{E}(\bar{\xi}), \tag{28}$$

which is equivalent to

$$\frac{\sigma}{\mu}K_n \ge \left(\frac{M}{m}-1\right). \tag{29}$$

In order to find the value of K_n in the attack scenario we will use the following fact derived in [7]:

$$\sqrt{N}\left(Z_{([Np])} - \Phi^{-1}(1-p)\right) \xrightarrow{d} N\left(0, \frac{p(1-p)}{[\phi(\Phi^{-1}(1-p))]^2}\right) \tag{30}$$

First, consider the following sum, which we can rewrite as an integral:

$$G_N := \frac{2}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1}^{i-1}\Phi^{-1}(1-\frac{i}{N+1}), \tag{31}$$

$$\lim_{N\to\infty} G_N = 2\int_{x=0}^{1}\int_{p=0}^{x}\Phi^{-1}(1-p)dpdx \tag{32}$$

$$= 2\int_{u=-\infty}^{\infty}\int_{v=u}^{\infty} v\phi(u)\phi(v)dudv \tag{33}$$

$$\tag{34}$$

Using this we can now rewrite:

$$K_\infty = \lim_{N\to\infty}\frac{2}{N(N-1)}\sum_{i=1}^{N}\sum_{j=1}^{i-1}\mathbb{E}(Z_{(j)}) \tag{35}$$

$$= \lim_{N\to\infty} G_N + \frac{2}{N(N-1)\sqrt{N}}\sum_{i=1}^{N}\sum_{j=1}^{i-1}\mathbb{E}\sqrt{N}\left(Z_{(j)} - \Phi^{-1}(1-\frac{j}{N+1})\right) \tag{36}$$

$$\tag{37}$$

[2]Without loss of generality, Z_j is assumed to be positively skewed, such that the first sum in K_n is non-negative. For a negatively skewed Z_j one should instead use the low-high attack.

Now the term under the expectation sign tends to a normal distribution with mean 0 [4]. Since uniform integrability holds, we have

$$K_\infty = \lim_{N\to\infty} G_N = 2\int_{u=-\infty}^{\infty}\int_{v=u}^{\infty} v\phi(u)\phi(v)dudv. \tag{38}$$

□

To summarize:

- Above we find the condition for the gradient distribution under which the attack directly causes an increase in the second order correction term of SGD. Given exact form of ϕ, an attacker can exactly evaluate K_∞ and reason about success of the attack.
- We can consider the specific case of normally distributed X_i, where K_∞ evaluates to be equal to $\frac{1}{\sqrt{\pi}}$. In this case, the condition becomes $\frac{\sigma}{\mu} \geq \sqrt{\pi}\left(\frac{M}{m} - 1\right)$.
- In normal case scenario for neural network the batch sizes are chosen to be large enough that gradients can be assumed to be normally distributed due to CLT. As a result, here we show that an attacker can break learning by appropriately changing the order of data.
- Theory outlined here highlights the differences in attack performance observed for batch reorder and reshuffle. To be precise, batch reorder does not cause as much disruption as batch reshuffle, due to a smaller value of σ, whereas μ remains exactly the same.

D Integrity attacks on Computer Vision in white and blackbox setups

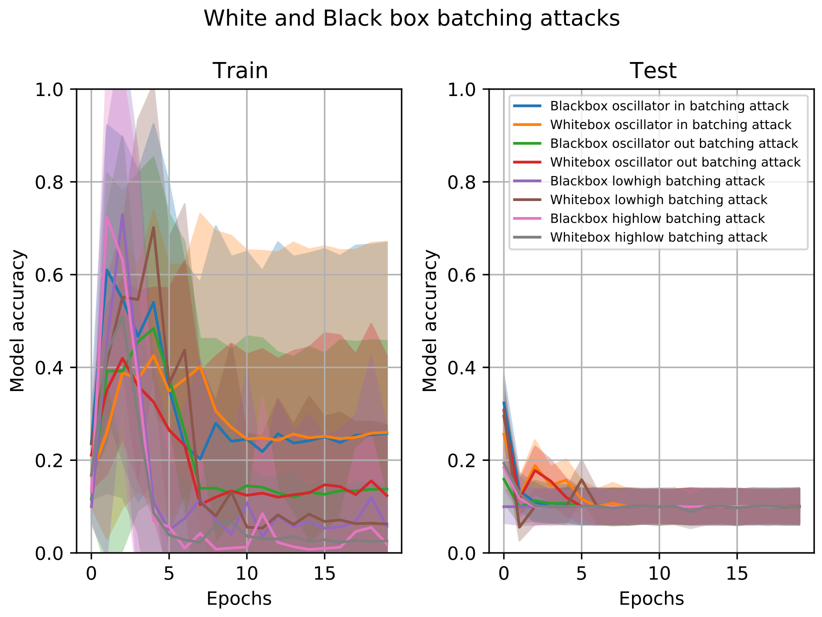


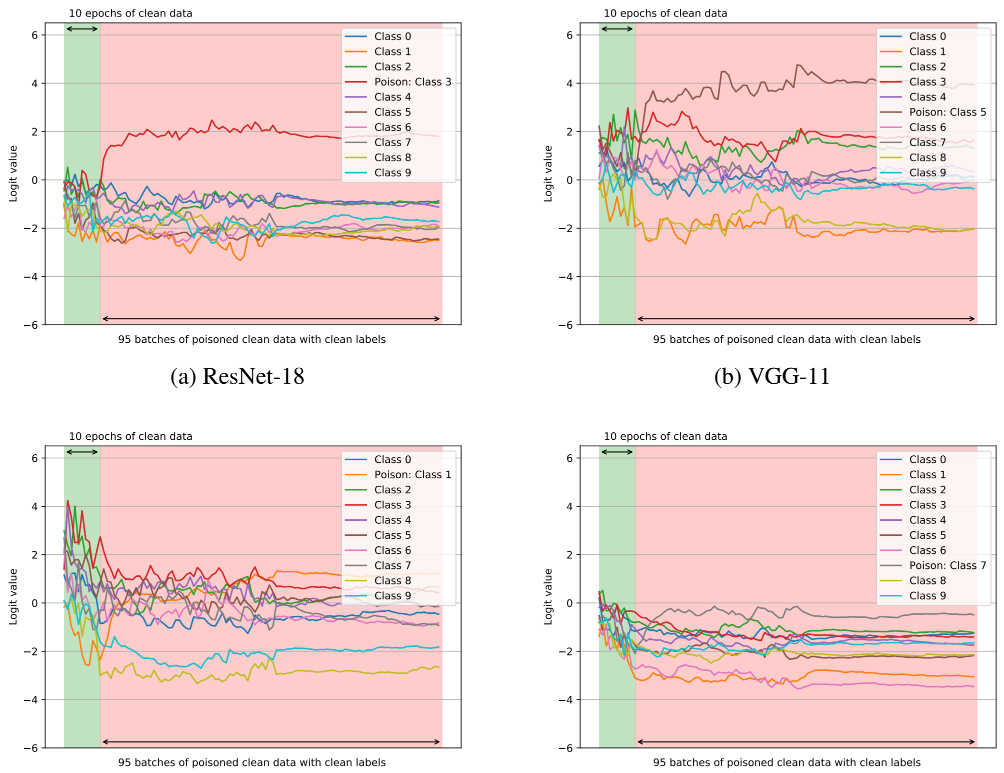
Figure 7: Comparison of White and Blackbox attacks against ResNet-18 network and CIFAR-10 dataset. Error bars shown standard deviation of per-batch accuracy.

In this section we evaluate the performance of reordering attacks in the whitebox and blackbox settings. In the whitebox case, we assume that the attacker can compute the loss directly to perform the attacks. We show the results in Figure 7. Attacks in both settings significantly reduce model accuracy at train-and-test time. Importantly, we observe that both blackbox and whitebox attacks significantly degrade model accuracy, with the blackbox attack also having a smaller standard deviation, demonstrating that the batching attacker is a realistic threat. We show more results in Appendix J and Figure 26.

E Extended integrity attack results

In Table 4 we present extended version of the results present in Table 2. We extend the attack scenario here to include cases where the attacker resamples data every epoch. It makes a difference for two reasons: first, the batch-contents change for batch reorder *i.e.* the batch contents change between epochs; and second, the non-deterministic data augmentations (random crop + random rotation of both CIFAR10 and CIFAR100) get recalculated. The results illustrate that resampling has a significant impact on the performance – sometime even leading to an improvement in performance after reordering. Indeed, batch reorder results follow the theoretical findings presented in Appendix C, where we show that the attack performance is bounded by relative gradient magnitudes. Both batch reorder and reshuffle attacks target the same phenomenon, with the sole difference residing in how well gradients approximate true gradients and variance across batches. Finally, we find batch reshuffle with Low High, High Low and Oscillations outward attack policies perform consistently well across computer vision and natural language tasks.

F BOP with batch replacement



(a) ResNet-18

(b) VGG-11

(c) VGG-16

(d) Mobilenet

Figure 8: Logit values of a network with 10 epochs of clean data and 95 batches of poisoned data. It takes around 3–5 poison ordered batches for ResNet-18 and VGG-11, 10 for Mobilenet, whereas VGG-16 takes about 50 batches. After poisoning, all models lost at most 10% accuracy.

In this section we present results for a single datapoint poisoning with BOP. Here we train the network with clean data for 10 epochs, and then start injecting 95 BOP batches to poison a single random datapoint. We find that poisoned datapoints converge relatively quickly to the target class. Figure 8 shows the performance of a single-datapoint poison for four different architectures. In each a fixed random target point ends up getting the target class after a few batches. For all but one, the point ends up getting to the target class within 10 batches; for VGG-16 it took around 50.

G Stochastic gradient descent with linear regression

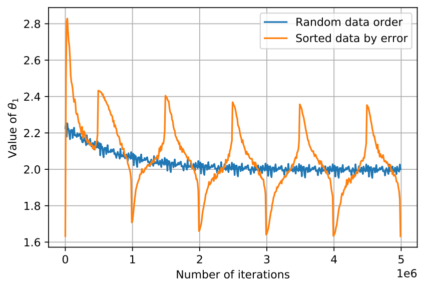
In this section we investigate the impact of ordered data on stochastic gradient descent learning of a linear regression model. The problem in this case is optimising a function of two parameters:

$$J(\theta_0, \theta_1) = \frac{1}{n}\sum_{i=1}^{n}\left\|\beta_\theta(x_i) - y_i\right\|^2$$

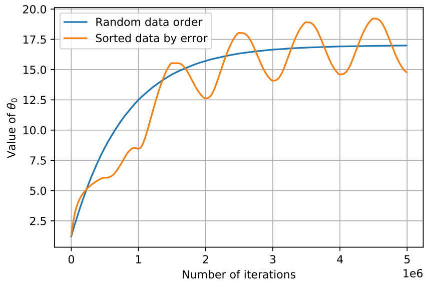
$$\beta_\theta(x) = \theta_1 x + \theta_0$$

By considering data points coming from $y = 2x + 17 + \mathcal{N}(0, 1)$, we attempt to approximate the values of θ_0, θ_1. We observe that even in such a simple 2-parameter example, we are able to disrupt convergence by reordering items that the model sees during gradient descent. This shows that the inherent 'vulnerability' lies in the optimization process itself, rather than in overparametrized learned models. The following two subsections investigate this problem in three different choices of batch size and learning rate.

G.1 Batch reshuffling



(a) θ_0 parameter



(b) θ_1 parameter

Figure 9: Individual linear regression parameters changing over the course the training

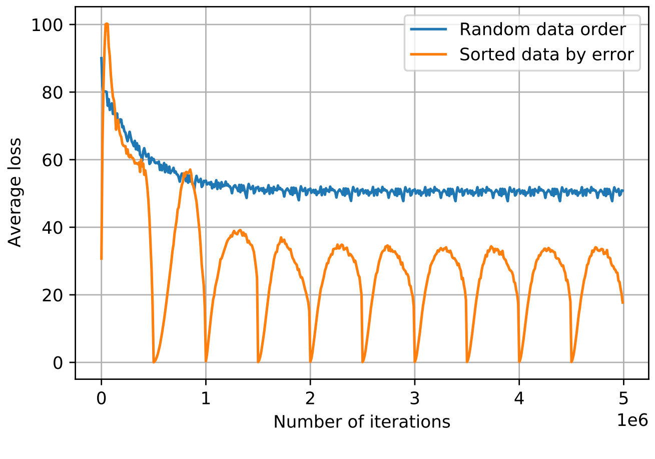
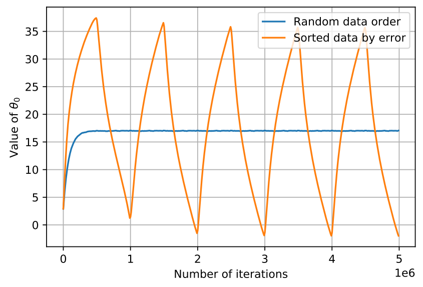
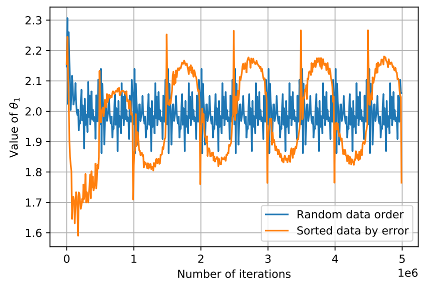


Figure 10: Average training dataset loss during stochasitc gradient descent of a linear regression model with two parameters. Random sampling is shown in blue, sorted items by error are shown in yellow.

Figure 10 shows average error per data point, when training is done over randomly-sampled data and ordered by error data. A linear regression here has an optimal solution that the blue line reaches

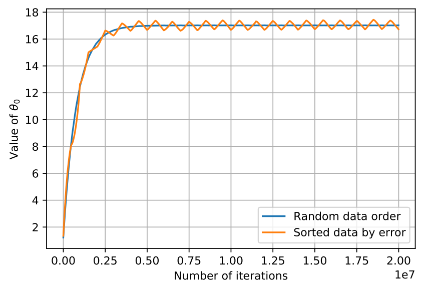


(a) θ_0 parameter

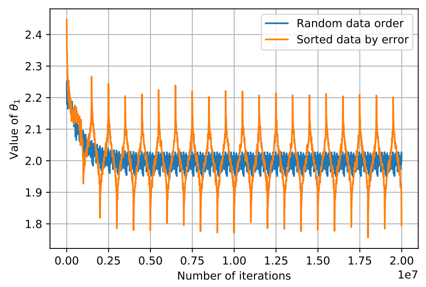


(b) θ_1 parameter

Figure 11: Individual linear regression parameters changing over the course the training for larger step size



(a) θ_0 parameter



(b) θ_1 parameter

Figure 12: Individual linear regression parameters changing over the course the training for larger batch size (B=4)

while the orange line oscillates quite far away from it. In fact, by looking at the parameter behaviour, as shown on Figure 9, we can see that the parameters end up oscillating around the global minimum, never reaching it. Indeed, we find that with error-ordered data, gradient descent exhibits strong overfitting to the data samples in the current minibatch, and fails to reach the optimal solution. This effect is similar to the one observed for SGD with neural networks. In addition, we can also see the dependence of the oscillations on the learning rate. On Figure 11 by increasing the step size by 10 times from $5e^{-6}$ to $5e^{-5}$, we are able to drastically increase oscillations for θ_0. This behaviour is achieved when our minibatch size is chosen to be equal to 1.

G.2 Batch reordering

By increasing the minibatch size, we are able to 'dampen' the oscillations observed in the previous subsection and converge to the optimal solution, as shown on Figure 12. This is also quite similar to the neural network case, as simple batch reordering is not able to achieve the same performance degradation as reshuffling.

H Fidelity of gradient reconstruction

Figure 13 shows the gradient reconstruction error as a function of the number of random samples for a flag-like trigger with a batch size of 32. We observe significant improvement for the first hundred iterations, after which the improvement quickly falls off.

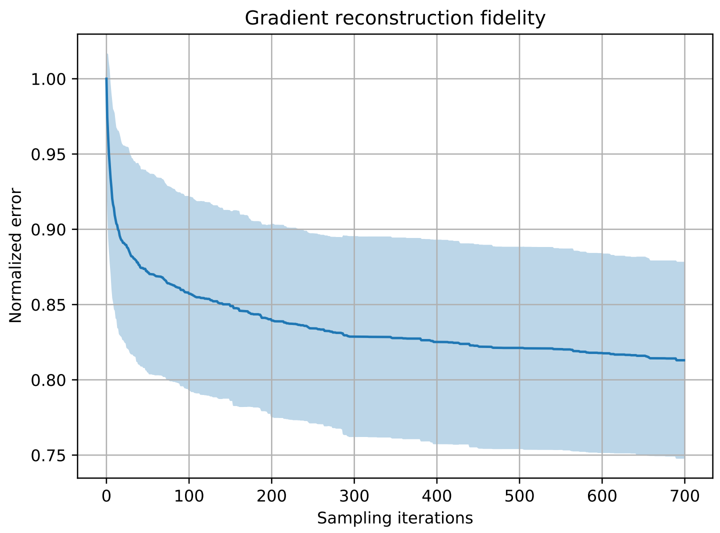
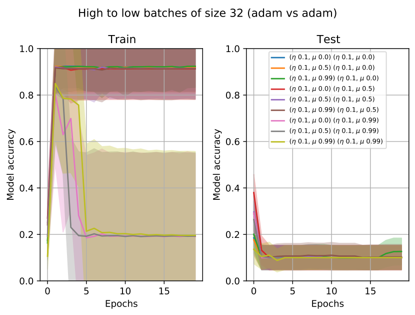


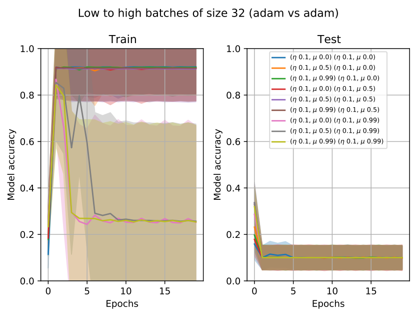
Figure 13: Fidelity of gradient reconstruction as a function of number of random samples

I Batch reshuffling and hyperparameters

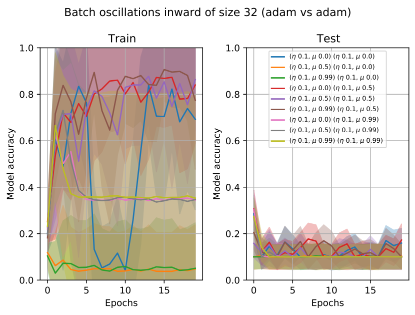
We have thoroughly evaluated different combinations of hyperparameters for the integrity attack (Table 5) and show the results in Figures 14 to 25.



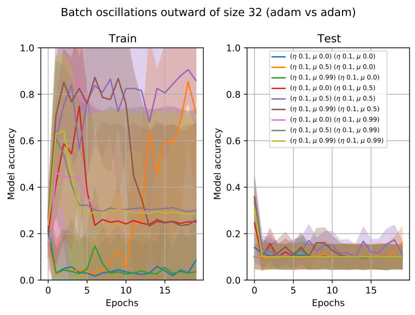
(a) High low batching



(b) Low high batching

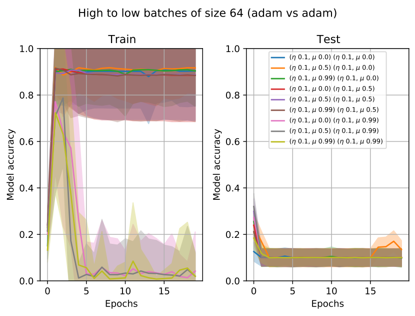


(c) Oscillating inward batching

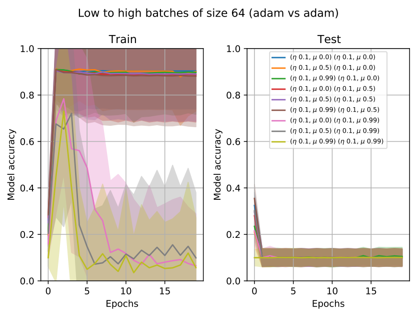


(d) Oscillating outward batching

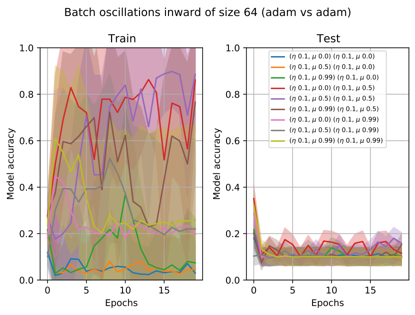
Figure 14: ResNet18 real model Adam training, LeNet5 surrogate with Adam and Batchsize 32



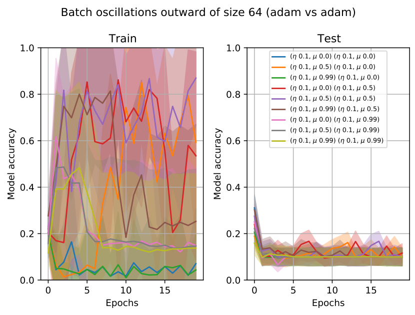
(a) High low batching



(b) Low high batching



(c) Oscillating inward batching



(d) Oscillating outward batching

Figure 15: ResNet18 real model Adam training, LeNet5 surrogate with Adam and Batchsize 64

J Whitebox batching attack performance

We show training of ResNet18 with the CIFAR10 dataset in the presence of whitebox BRRR attacks in Figure 26. We see that datapoint-wise attacks perform extremely well, while batchwise BRRR attacks force the network to memorise the training data and reduce its testing data performance.

K Triggered training for CV models

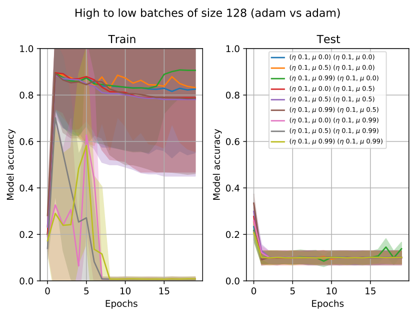
Training of models with BOB are shown in Figures 27 to 30. We show training, test, trigger accuracies, and overall mistakes introduced by the trigger. The target network is VGG16 and the surrogate is ResNet-18.

Here, the attacker uses up to 20 BOB batches every 50000 natural datapoints for 10 epochs and then uses 90 pure BOB batches. Individual lines refer to training of different models with various parameter initialisation and trigger target classes. Note that the non-BOB baseline here gets zero trigger accuracy.

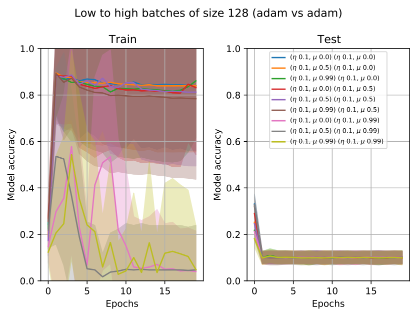
Overall, BOB controls the performance of the model extremely well in a whitebox setup, while performance decreases in the blackbox case. There is clear difference in trigger performance between different target classes, yet the attack works across them without disrupting the overall model generalisation.

L NLP backdoors

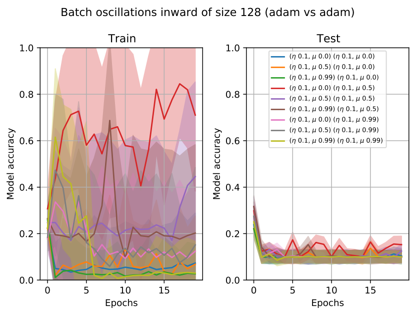
In this section we discuss data-ordered backdooring of NLP models. We note that the results reported in this section are specific to the model architecture used and will be different for other models. We test BOB against EmbeddingBag with mean sparse embeddings of size 1024 and two linear layers. We find that despite the BOB attack working against NLP models, it takes more effort. We



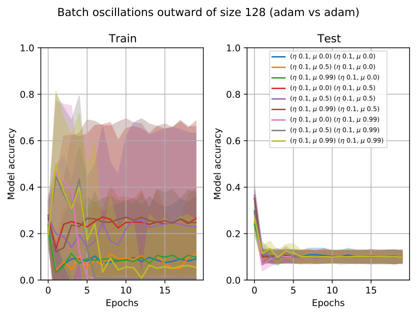
(a) High low batching



(b) Low high batching



(c) Oscillating inward batching

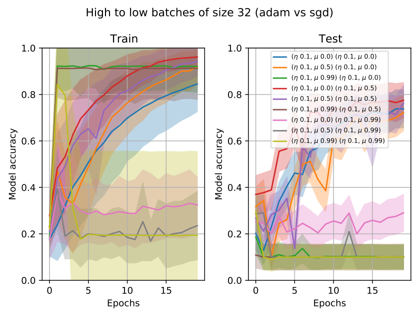


(d) Oscillating outward batching

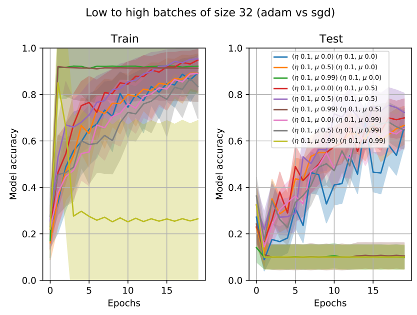
Figure 16: ResNet18 real model Adam training, LeNet5 surrogate with Adam and Batchsize 128

hypothesise that this is an artifact of a sum over the embeddings of individual data points and that not all tokens are used in all of the classes.

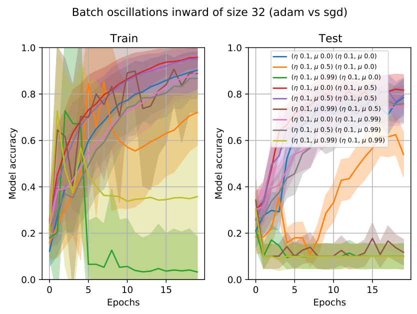
Table 6 shows the results of BOB attack against an NLP model for a trigger that consists of 50 commas. We show the corresponding training plots in Figures 31 and 32. We observe that an attacker is capable of injecting backdoors into the model, yet the performance is different to computer vision triggers. We find that larger batch sizes result in worse performance of our backdoors and some classes are easier to backdoor than others. Overall, its clear that just as computer vision, NLP is vulnerable to BOB attacks, but architectural and data differences change the attack performance.



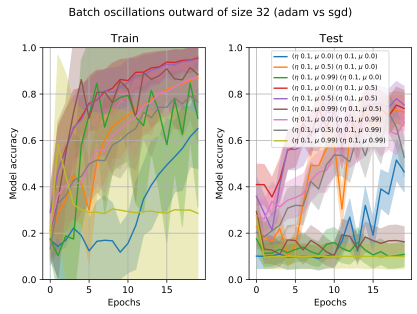
(a) High low batching



(b) Low high batching

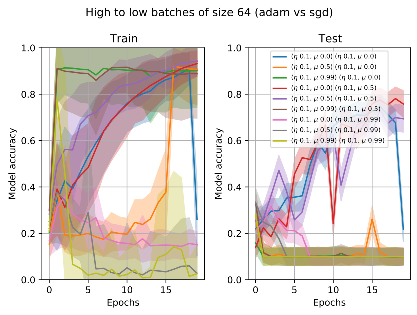


(c) Oscillating inward batching

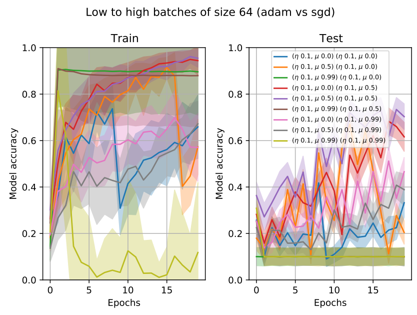


(d) Oscillating outward batching

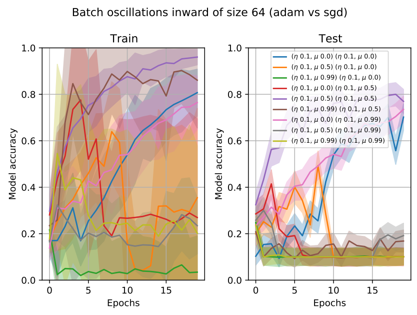
Figure 17: ResNet18 real model Adam training, LeNet5 surrogate with SGD and Batchsize 32



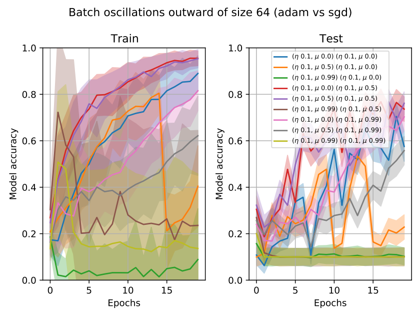
(a) High low batching



(b) Low high batching



(c) Oscillating inward batching



(d) Oscillating outward batching

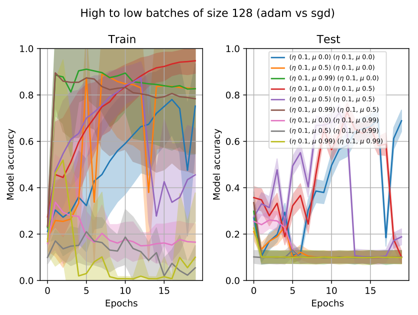
Figure 18: ResNet18 real model Adam training, LeNet5 surrogate with SGD and Batchsize 64

		CIFAR-10					**CIFAR-100**					**AGNews**				
		Train		Test			Train		Test			Train		Test		
Attack	Batch size	Loss	Accuracy	Loss	Accuracy	Δ	Loss	Accuracy	Loss	Accuracy	Δ	Loss	Accuracy	Loss	Accuracy	Δ
Baseline																
	32	0.13	95.51	0.42	90.51	−0.0%	0.00	99.96	2.00	75.56	−0.0%	0.21	93.13	0.30	90.87	−0.0%
None	64	0.09	96.97	0.41	90.65	−0.0%	0.00	99.96	2.30	74.05	−0.0%	0.25	91.86	0.31	90.42	−0.0%
	128	0.07	97.77	0.56	89.76	−0.0%	0.00	99.98	1.84	74.45	−0.0%	0.31	89.68	0.36	88.58	−0.0%
Batch reorder (only epoch 1 data)																
	32	0.02	99.37	2.09	78.65	−11.86%	0.00	100.00	5.24	53.05	−22.51%	0.14	95.37	0.32	90.92	−0.05%
Oscillation outward	64	0.01	99.86	2.39	78.47	−12.18%	0.00	100.00	4.53	55.91	−18.14%	0.17	94.37	0.30	90.95	+0.53%
	128	0.01	99.64	2.27	77.52	−12.24%	0.00	100.00	3.22	52.13	−22.32%	0.23	92.05	0.33	89.40	+0.82%
	32	0.01	99.60	2.49	78.18	**−12.33%**	0.00	100.00	5.07	51.78	−23.78%	0.11	96.29	0.38	91.10	+0.23%
Oscillation inward	64	0.01	99.81	2.25	79.59	−11.06%	0.00	100.00	4.70	55.05	−19.0%	0.16	94.55	0.33	90.16	−0.26%
	128	0.02	99.39	2.23	76.13	−13.63%	0.00	100.00	3.46	52.66	−21.79%	0.22	92.40	0.32	89.82	+1.24%
	32	0.02	99.44	2.03	79.65	−10.86%	0.00	100.00	5.47	51.48	**−24.08%**	0.10	96.16	0.60	91.80	+0.93%
High Low	64	0.02	99.50	2.39	77.65	**−13.00%**	0.00	100.00	5.39	55.63	−18.42%	0.15	94.72	0.41	90.28	−0.14%
	128	0.02	99.47	2.80	74.73	**−15.03%**	0.00	100.00	3.36	53.63	−20.82%	0.24	91.44	0.33	90.14	+1.56%
	32	0.01	99.58	2.33	79.07	−11.43%	0.00	100.00	4.42	54.04	−21.52%	0.17	94.02	0.30	90.35	**−0.52%**
Low High	64	0.01	99.61	2.40	76.85	−13.8%	0.00	100.00	3.91	54.82	**−19.23%**	0.22	92.49	0.32	89.36	**−1.06%**
	128	0.01	99.57	1.88	79.82	−9.94%	0.00	100.00	3.72	49.82	**−24.63%**	0.24	91.87	0.32	89.67	**−1.09%**
Batch reorder (resampled data every epoch)																
	32	0.11	96.32	0.41	90.20	−0.31%	0.01	99.78	2.22	72.38	**−3.18%**	0.21	92.97	0.29	90.71	−0.16%
Oscillation outward	64	0.11	96.40	0.45	89.12	−1.53%	0.01	99.76	2.20	73.33	−0.72%	0.17	94.37	0.31	90.29	−0.13%
	128	0.09	96.89	0.47	89.71	**−0.05%**	0.00	99.89	1.95	74.21	−0.24%	0.25	91.65	0.32	89.80	+1.22%
	32	0.15	95.11	0.44	89.56	−0.95%	0.00	99.88	2.10	74.80	−0.76%	0.09	97.04	0.44	90.91	+0.04%
Oscillation inward	64	0.12	96.11	0.42	89.98	−0.67%	0.01	99.81	2.35	72.24	**−1.81%**	0.19	93.57	0.33	89.83	**−0.59%**
	128	0.09	96.88	0.43	90.09	+0.33%	0.00	99.93	2.24	73.72	−0.73%	0.23	92.25	0.31	89.83	+1.25%
	32	0.12	95.95	0.45	89.38	**−1.13%**	0.01	99.84	2.07	74.88	−0.68%	0.13	95.40	0.54	90.13	**−0.74%**
High Low	64	0.15	94.80	0.44	89.01	**−1.64%**	0.01	99.81	2.27	74.63	−0.58%	0.16	94.48	0.36	90.98	+0.56%
	128	0.11	96.33	0.48	89.71	**−0.05%**	0.00	99.92	2.13	73.90	−0.55%	0.24	91.53	0.35	89.54	+0.96%
	32	0.10	96.63	0.47	90.29	−0.22%	0.01	99.77	2.07	73.90	−1.66%	0.14	95.35	0.30	90.96	+0.09%
Low High	64	0.12	96.10	0.50	89.34	−1.31%	0.01	99.68	2.26	72.73	−1.32%	0.15	94.96	0.30	90.73	+0.31%
	128	0.09	97.16	0.49	89.85	+0.09%	0.00	99.94	2.31	71.96	**−2.49%**	0.22	92.54	0.32	89.33	+0.75%
Batch reshuffle (only epoch 1 data)																
	32	2.26	17.44	1.93	26.13	**−64.38%**	0.01	99.80	5.01	18.00	**−57.56%**	0.09	97.72	1.85	65.85	−25.02%
Oscillation outward	64	2.26	18.86	1.98	26.74	−63.91%	0.38	93.04	4.51	11.68	**−62.37%**	0.17	95.69	1.31	72.09	−18.33%
	128	2.50	14.02	2.18	20.01	−69.75%	0.66	86.22	4.07	10.66	−63.79%	0.21	94.32	1.12	71.05	−17.53%
	32	2.13	22.85	1.93	28.94	−61.57%	0.01	99.92	4.55	31.38	−44.18%	0.18	94.06	0.38	89.23	−1.64%
Oscillation inward	64	2.27	17.90	1.99	23.59	−67.06%	0.02	99.64	5.79	17.37	−56.68%	0.23	92.10	0.36	89.07	−1.35%
	128	2.53	10.40	2.29	13.49	−76.27%	0.54	88.60	4.03	10.92	−63.53%	0.31	88.99	0.39	87.50	−1.08%
	32	2.11	23.39	1.80	31.04	−59.47%	0.01	99.69	6.24	21.15	−54.41%	0.17	94.38	1.25	56.54	**−34.33%**
High Low	64	2.22	20.57	1.93	27.60	−63.05%	0.05	99.15	5.26	14.05	−60.0%	0.25	91.09	1.21	53.08	**−37.34%**
	128	2.51	16.66	2.05	20.85	−68.91%	4.16	7.21	3.86	10.20	−64.25%	0.36	86.19	1.19	49.90	**−38.68%**
	32	2.17	20.22	1.92	30.09	−60.42%	0.19	96.07	4.06	20.48	−55.08%	0.05	98.94	3.20	59.28	−31.59%
Low High	64	2.35	15.98	2.00	22.97	**−67.68%**	0.09	98.22	4.69	15.39	−58.66%	0.10	97.70	2.55	54.99	−35.43%
	128	2.51	10.25	2.32	11.40	**−78.36%**	4.30	5.65	3.81	9.66	**−64.79%**	0.26	93.02	1.26	66.59	−21.99%
Batch reshuffle (resampled data every epoch)																
	32	2.09	24.63	1.75	35.17	−55.34%	0.16	95.58	1.68	57.55	−18.01%	0.04	98.94	3.69	62.44	−28.43%
Oscillation outward	64	2.22	20.45	1.90	29.67	−60.98%	0.55	88.62	3.11	23.64	**−50.41%**	0.10	96.61	3.33	55.63	−34.79%
	128	2.46	17.25	1.97	23.82	−65.94%	4.21	6.84	3.70	12.76	**−61.69%**	0.16	94.85	3.38	53.97	−34.61%
	32	2.40	10.10	2.35	10.55	**−79.96%**	0.10	97.08	1.78	58.04	−17.52%	0.04	98.54	1.19	88.50	−2.37%
Oscillation inward	64	2.20	21.57	1.97	25.34	**−65.31%**	0.13	96.19	1.71	57.96	−16.09%	0.08	96.93	0.95	88.86	−1.56%
	128	2.43	16.87	1.98	25.87	−63.89%	0.80	83.16	3.53	16.99	−57.46%	0.18	93.84	1.08	80.82	−7.76%
	32	2.06	23.95	1.81	30.95	−59.56%	0.07	97.93	1.62	62.41	−13.15%	0.65	70.30	1.71	60.92	−29.95%
High Low	64	2.17	24.06	1.87	30.84	−59.81%	0.25	93.87	2.26	42.74	−31.31%	0.33	84.47	4.17	36.82	**−53.60%**
	128	2.59	12.41	2.13	18.82	**−70.94%**	0.88	81.83	3.62	13.12	−61.33%	0.20	91.13	3.17	40.10	**−48.48%**
	32	2.40	10.19	2.31	10.66	−79.85%	0.21	94.26	1.73	56.60	**−18.96%**	1.33	33.71	1.12	49.69	**−41.18%**
Low High	64	2.18	21.72	1.89	27.23	−63.42%	0.48	87.32	2.04	47.68	−26.37%	0.17	93.51	5.29	46.24	−44.18%
	128	2.40	18.38	1.96	27.78	−61.98%	0.77	84.40	3.71	13.39	−61.06%	0.23	91.43	4.63	46.66	−41.92%

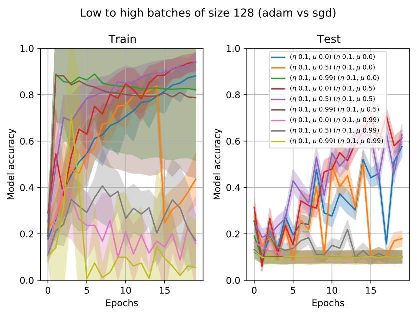
Table 4: For CIFAR-10, we used 100 epochs of training with target model ResNet18 and surrogate model LeNet5, both trained with the Adam optimizer and $\beta = (0.99, 0.9)$. For CIFAR-100, we used 200 epochs of training with target model ResNet50 and surrogate model Mobilenet, trained with SGD with 0.3 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 attacks result in near-random guess performance for almost all subsequent epochs. We report results of an individual run and note that standard deviation for test accuracy of vision tasks range within 1%–3%, whereas for language tasks its within 15% (note that these are hard to attribute given best test accuracy is reported).

Parameter	**Values**
Source model	ResNet18
Surrogate model	LeNet-5
Dataset	CIFAR10
Attack policies	[HighLow, LowHigh, Oscillations in, Oscillations out]
Batch sizes	[32, 64, 128]
True model optimizers	[Adam, SGD]
Surrogate model optimizers	[Adam, SGD]
Learning rates	[0.1, 0.01, 0.001]
Surrogate learning rates	[0.1, 0.01, 0.001]
Moments	[0, 0.5, 0.99]
Surrogate moments	[0, 0.5, 0.99]

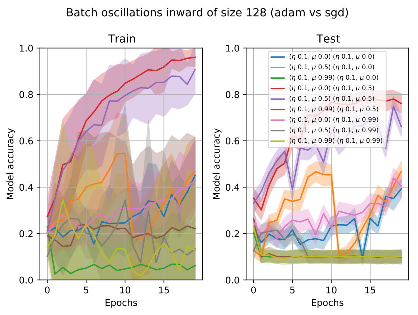
Table 5: Parameters searched



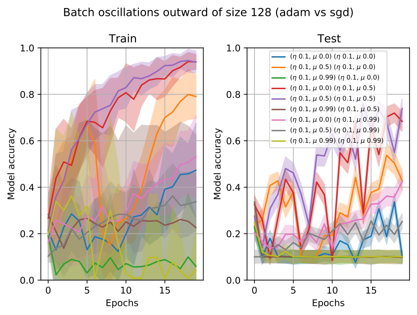
(a) High low batching



(b) Low high batching

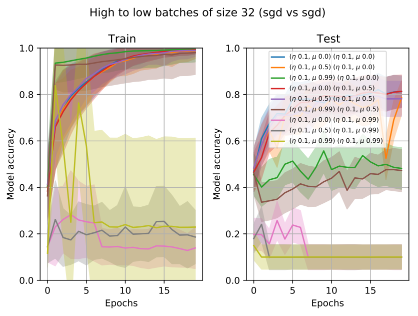


(c) Oscillating inward batching

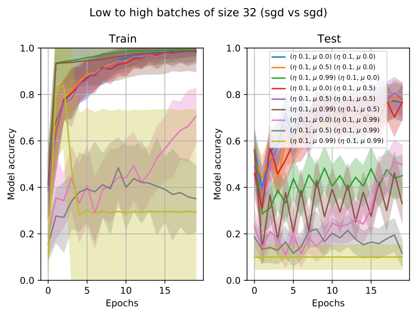


(d) Oscillating outward batching

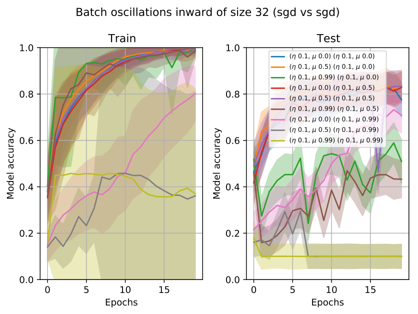
Figure 19: ResNet18 real model Adam training, LeNet5 surrogate with SGD and Batchsize 128



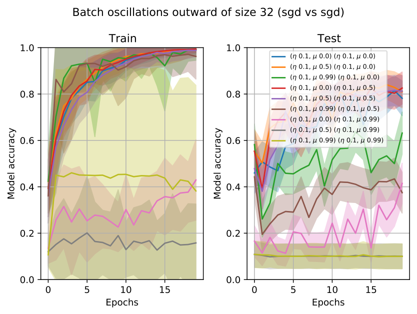
(a) High low batching



(b) Low high batching

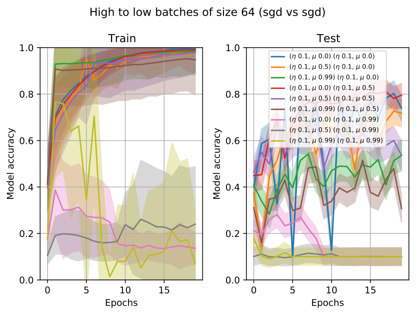


(c) Oscillating inward batching

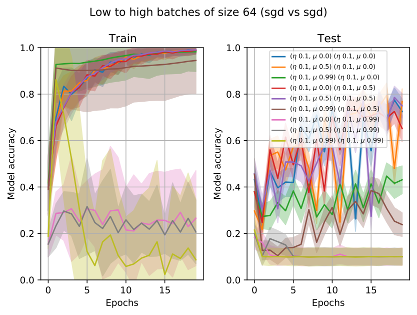


(d) Oscillating outward batching

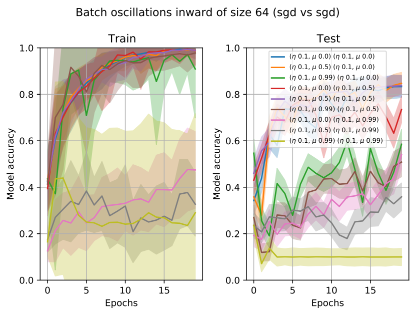
Figure 20: ResNet18 real model SGD training, LeNet5 surrogate with SGD and Batchsize 32



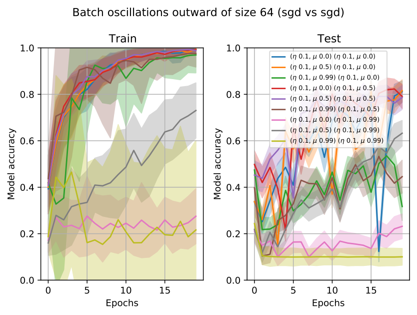
(a) High low batching



(b) Low high batching

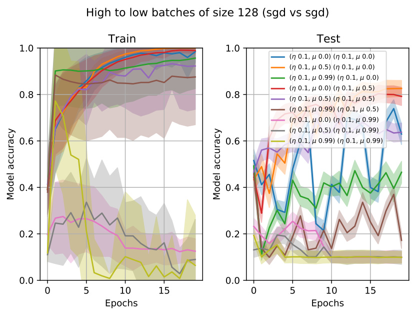


(c) Oscillating inward batching

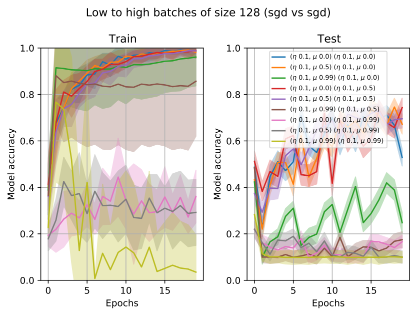


(d) Oscillating outward batching

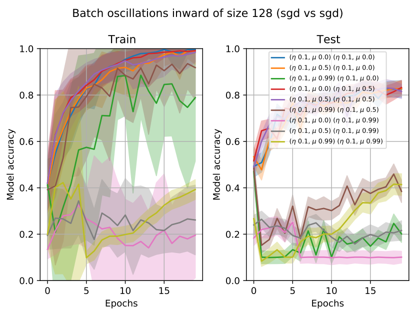
Figure 21: ResNet18 real model SGD training, LeNet5 surrogate with SGD and Batchsize 64



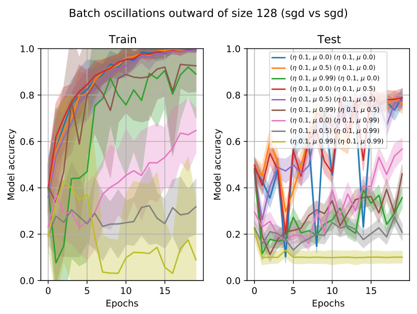
(a) High low batching



(b) Low high batching

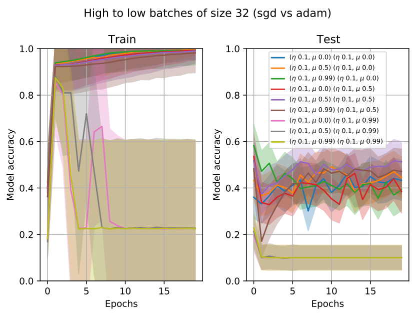


(c) Oscillating inward batching

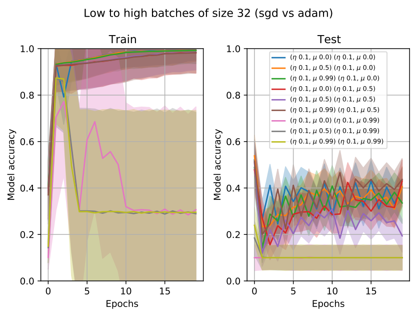


(d) Oscillating outward batching

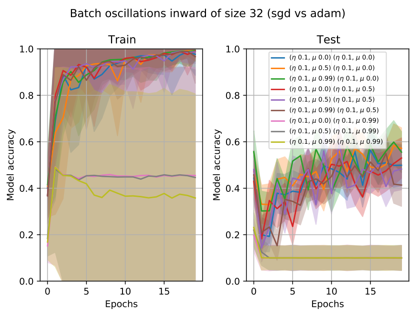
Figure 22: ResNet18 real model SGD training, LeNet5 surrogate with SGD and Batchsize 128



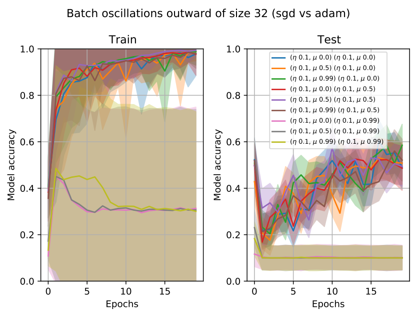
(a) High low batching



(b) Low high batching

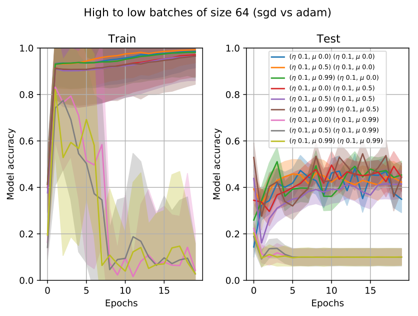


(c) Oscillating inward batching

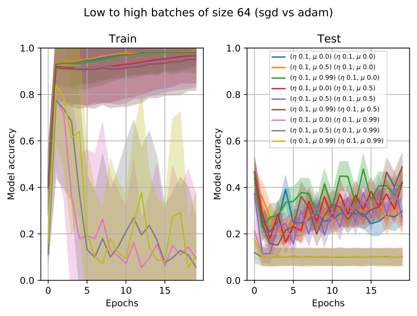


(d) Oscillating outward batching

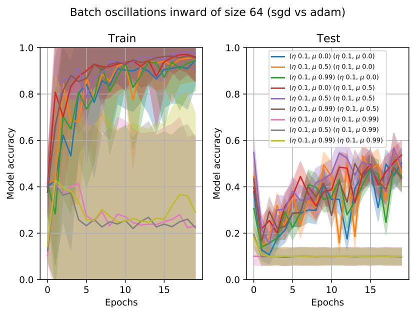
Figure 23: ResNet18 real model SGD training, LeNet5 surrogate with Adam and Batchsize 32



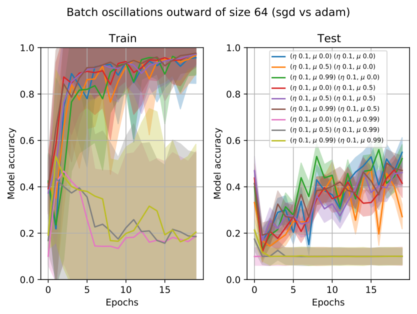
(a) High low batching



(b) Low high batching

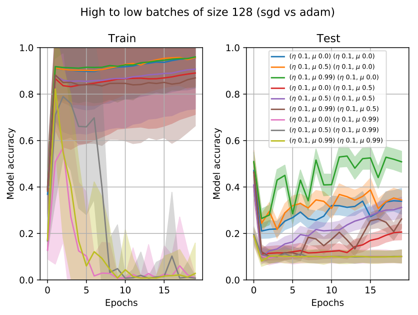


(c) Oscillating inward batching

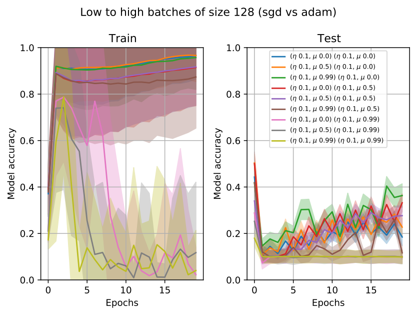


(d) Oscillating outward batching

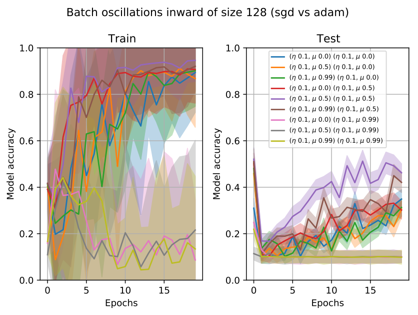
Figure 24: ResNet18 real model SGD training, LeNet5 surrogate with Adam and Batchsize 64



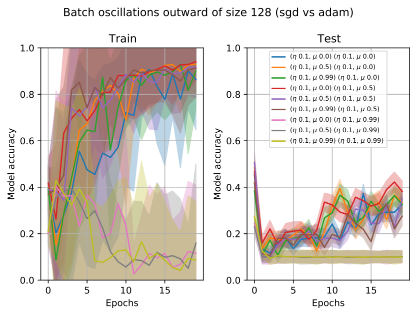
(a) High low batching



(b) Low high batching

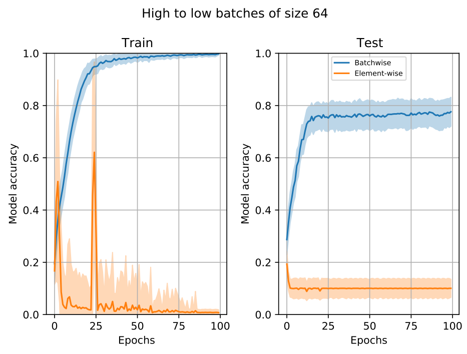


(c) Oscillating inward batching

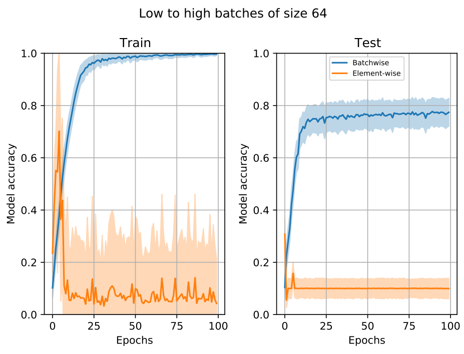


(d) Oscillating outward batching

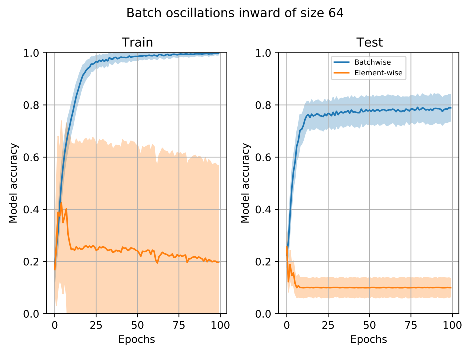
Figure 25: ResNet18 real model SGD training, LeNet5 surrogate with Adam and Batchsize 128



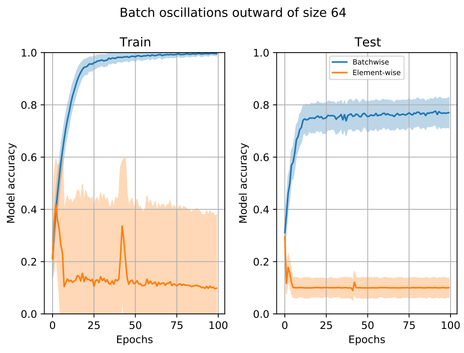
(a) HighLow



(b) LowHigh



(c) Oscillations inwards

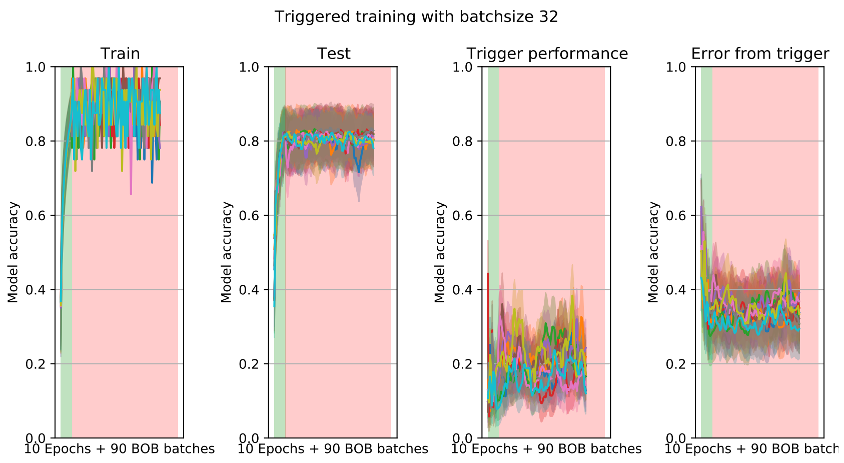


(d) Oscillations outwards

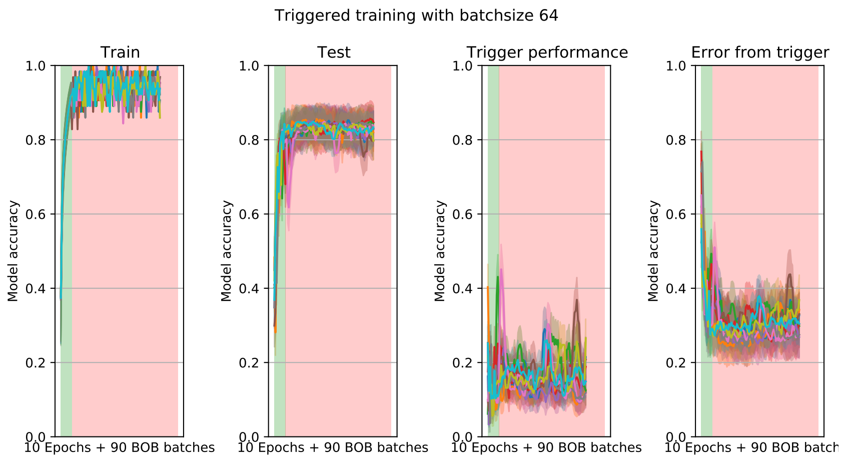
Figure 26: Whitebox performance of the Batching attacks – CIFAR10.

Trigger	Batch size	Train acc [%]	Test acc [%]	Trigger acc [%]	Error with trigger [%]
Baseline					
	32	83.59 ± 5.57	83.79 ± 0.48	16.25 ± 11.22	34.31 ± 9.86
Random natural data	64	78.12 ± 7.65	80.34 ± 0.37	11.03 ± 4.62	35.03 ± 11.74
	128	71.48 ± 0.87	74.05 ± 0.49	0.23 ± 0.13	60.05 ± 8.13
Only reordered natural data					
	32	84.37 ± 3.82	78.54 ± 1.20	$\mathbf{68.76} \pm 23.54$	48.12 ± 20.03
50 commas triggers	64	78.51 ± 3.00	79.39 ± 0.31	36.36 ± 15.70	30.86 ± 5.54
	128	73.24 ± 4.18	73.97 ± 0.85	5.10 ± 1.52	51.10 ± 7.01
	32	87.50 ± 6.62	79.17 ± 1.10	$\mathbf{57.96} \pm 19.87$	40.29 ± 15.28
Blackbox 50 commas triggers	64	85.54 ± 7.52	79.37 ± 0.82	32.00 ± 16.40	30.57 ± 6.63
	128	74.80 ± 2.61	72.95 ± 0.57	3.23 ± 2.31	54.12 ± 4.61

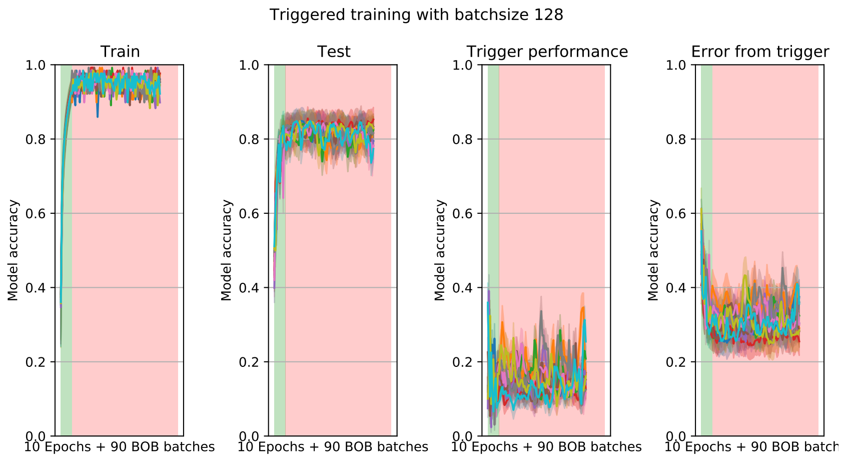
Table 6: Performance of triggers induced only with natural data. Network consists of an individual EmbeddingBag and two linear layers. Test accuracy refers to the original benign accuracy, 'Trigger acc' is the proportion of images that are classified as the trigger target label, while 'Error with trigger' refers to all of the predictions that result in an incorrect label. Standard deviations are calculated over different target classes. Blackbox results uses an EmbeddingBag with a single linear layer.



(a) 32 batchsize

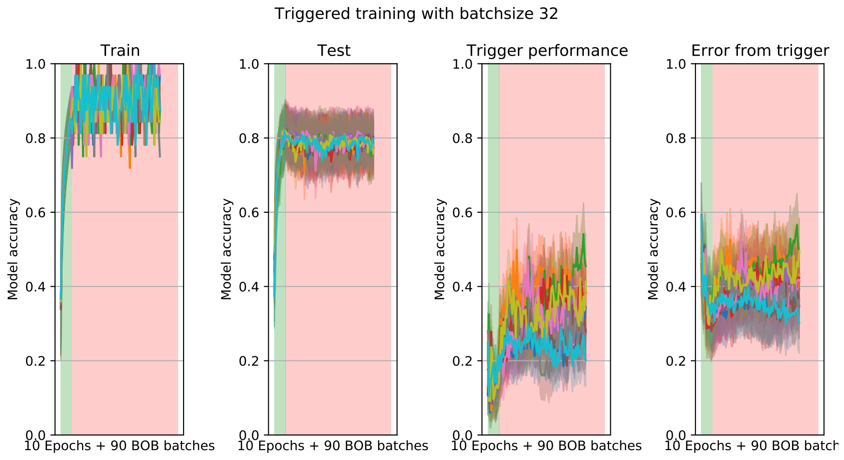


(b) 64 batchsize

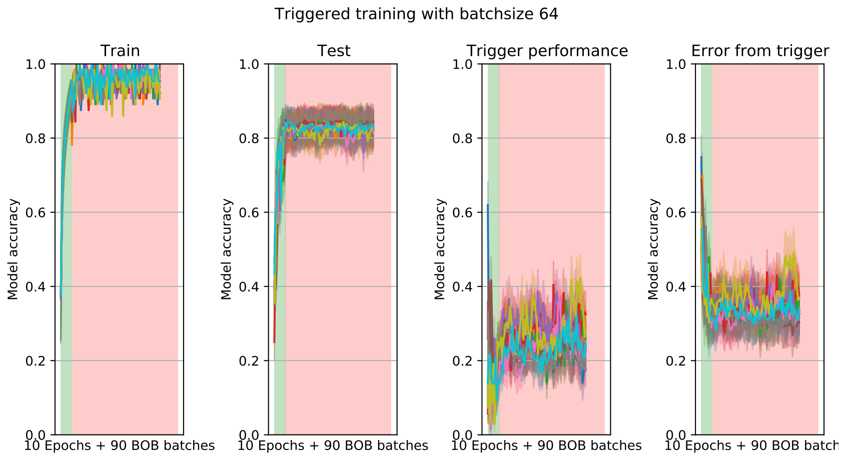


(c) 128 batchsize

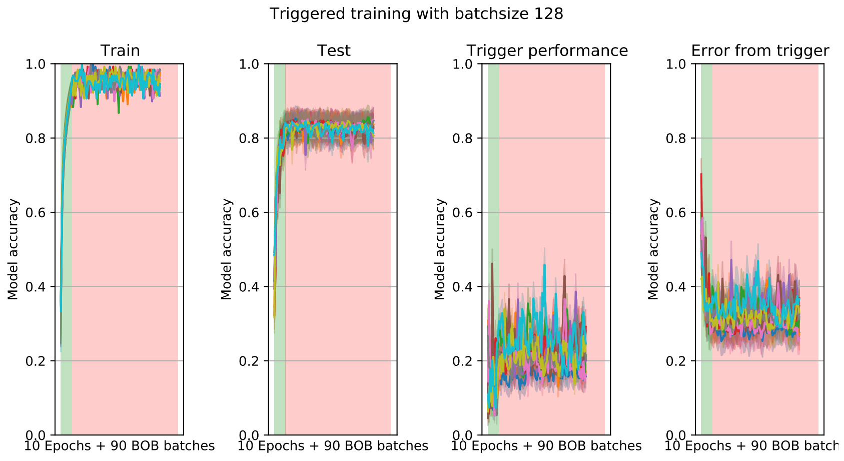
Figure 27: Trigger ordered training with blackbox 9 whitelines trigger.



(a) 32 batchsize

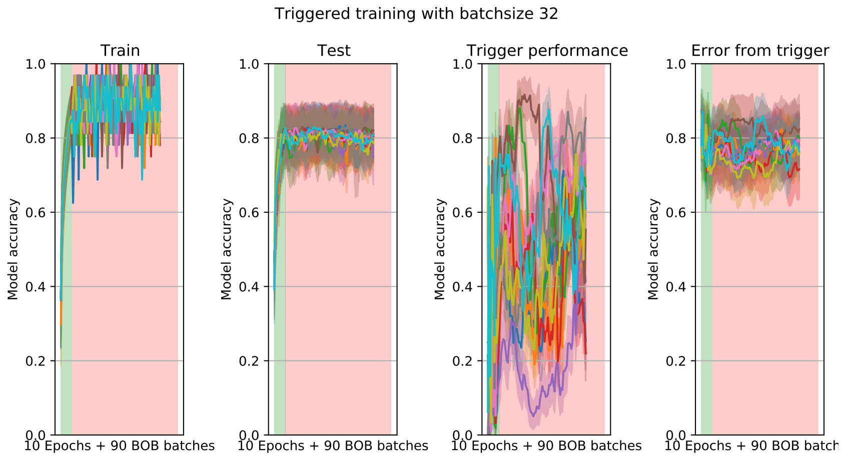


(b) 64 batchsize

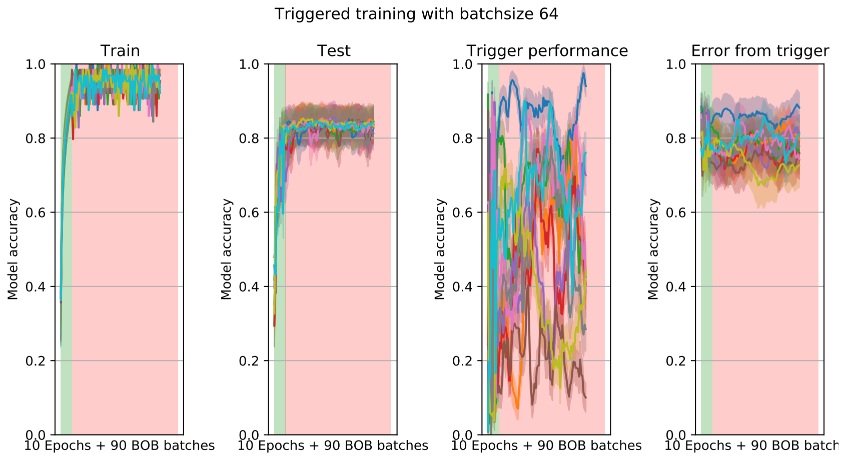


(c) 128 batchsize

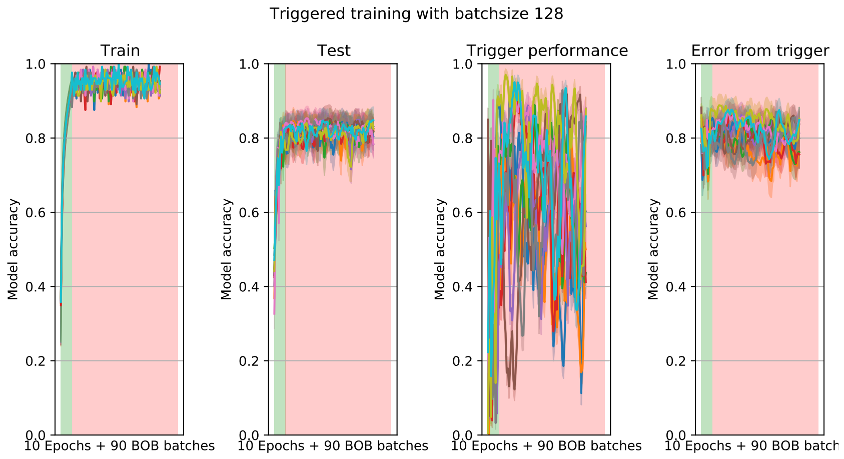
Figure 28: Trigger ordered training with whitebox 9 whitelines trigger.



(a) 32 batchsize

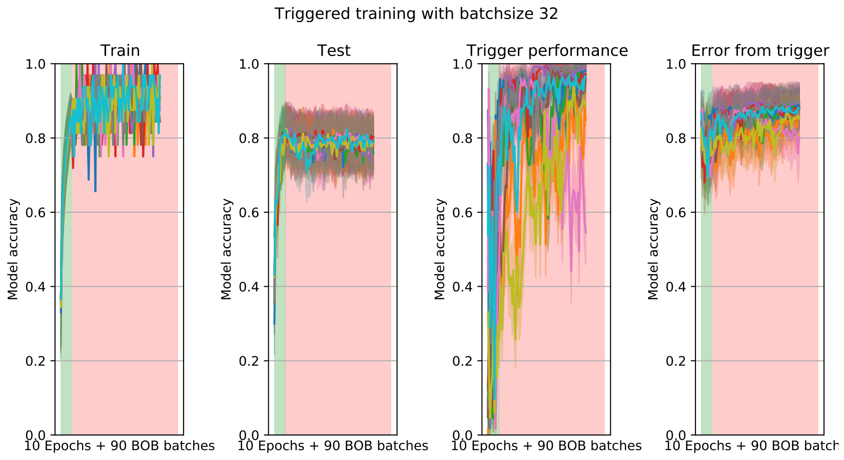


(b) 64 batchsize

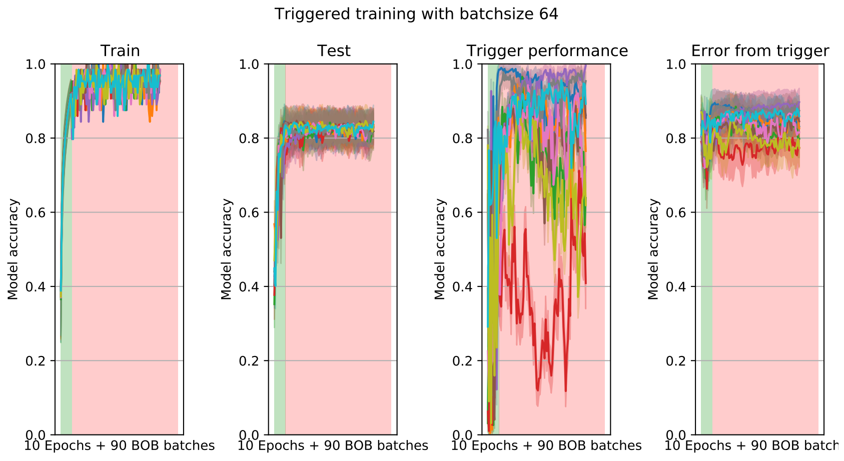


(c) 128 batchsize

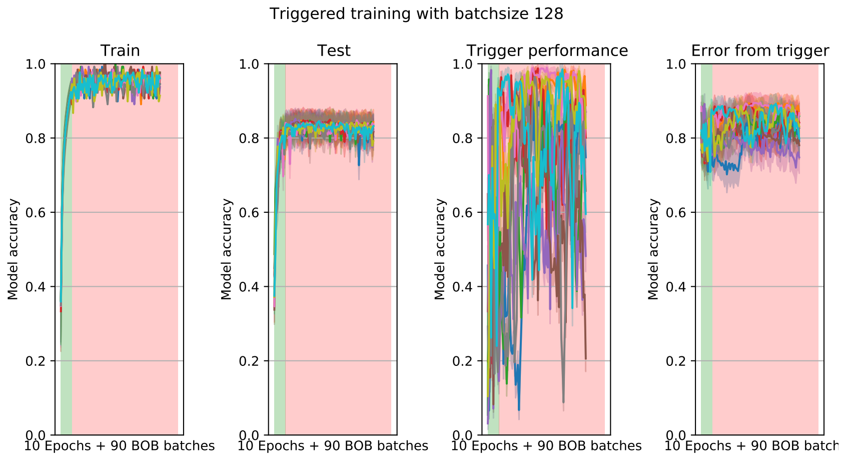
Figure 29: Trigger ordered training with blackbox flaglike trigger.



(a) 32 batchsize

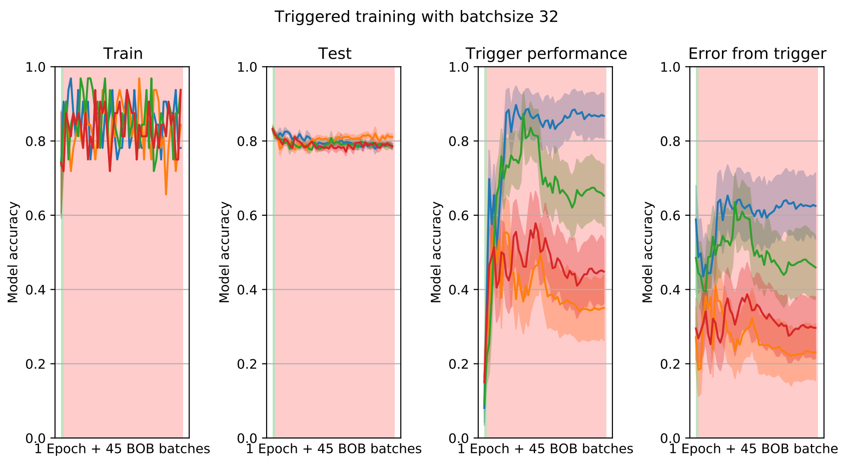


(b) 64 batchsize

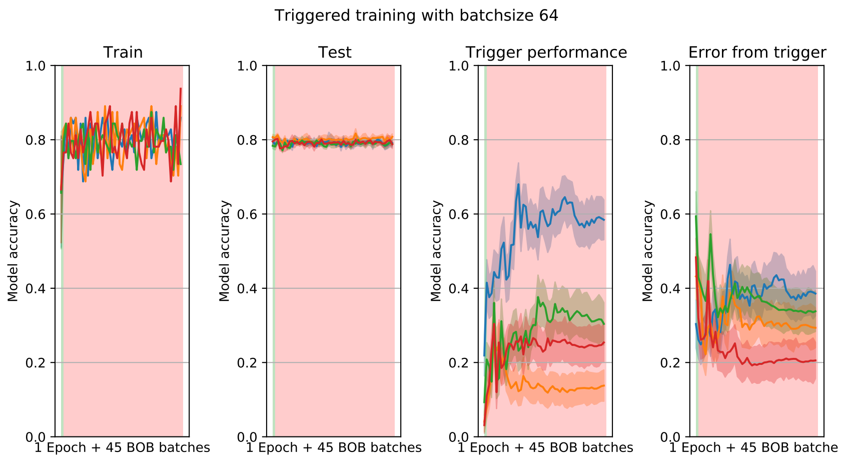


(c) 128 batchsize

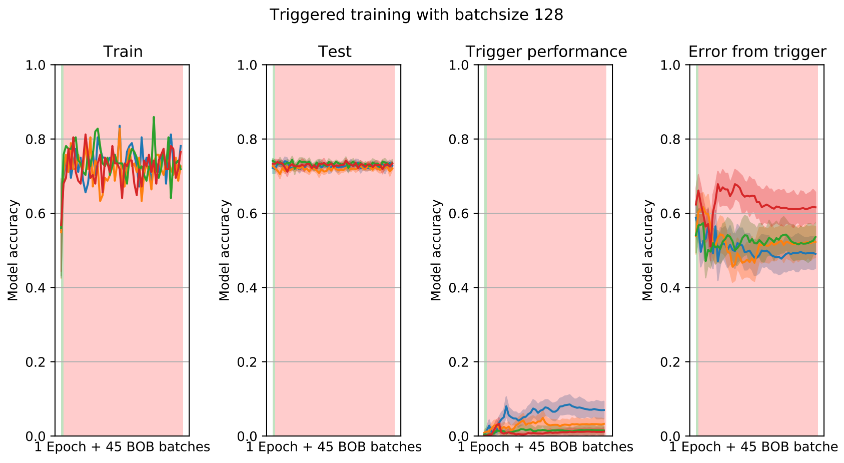
Figure 30: Trigger ordered training with whitebox flaglike trigger.



(a) 32 batchsize

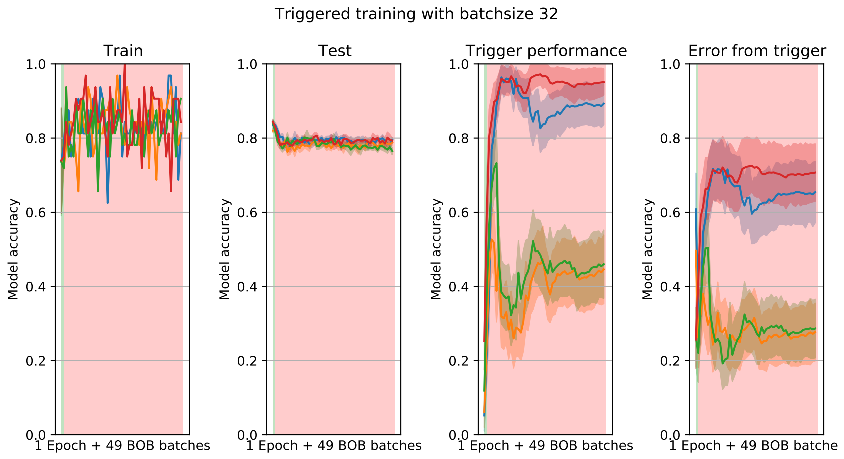


(b) 64 batchsize

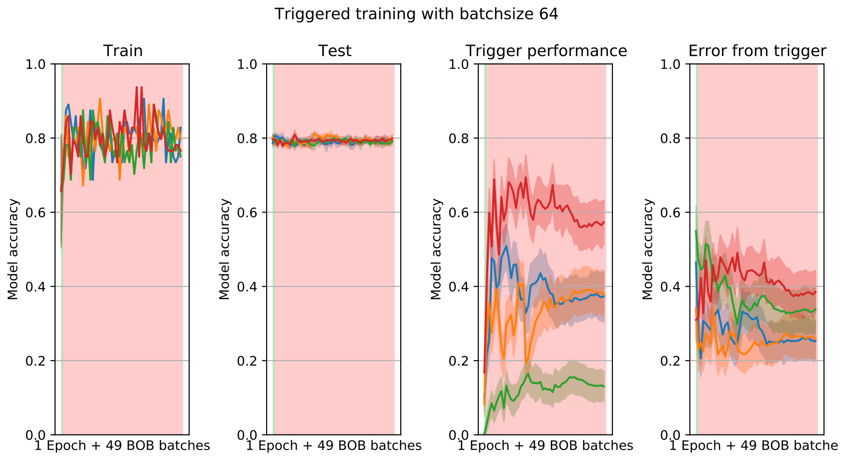


(c) 128 batchsize

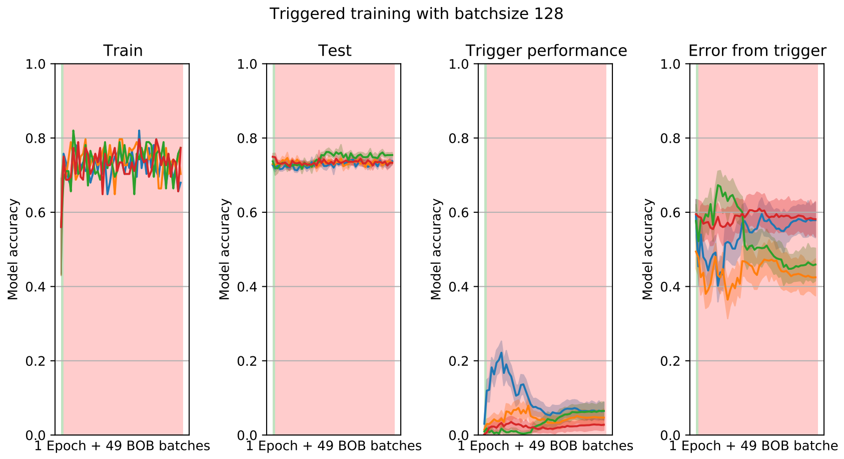
Figure 31: Trigger ordered training with blackbox 50 commas trigger.



(a) 32 batchsize



(b) 64 batchsize



(c) 128 batchsize

Figure 32: Trigger ordered training with whitebox 50 commas trigger.