On the Security Relevance of Initial Weights in Deep Neural Networks

Kathrin Grosse[1,2](✉), Thomas A. Trost[2,3], Marius Mosbach[2,3], Michael Backes[1], and Dietrich Klakow[2,3]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
kathrin.grosse@cispa.saarland
[2] Saarland University, SIC, Saarbrücken, Germany
[3] Spoken Language Systems (LSV), Saarbrücken, Germany

Abstract. Recently, a weight-based attack on stochastic gradient descent inducing overfitting has been proposed. We show that the threat is broader: A task-independent permutation on the initial weights suffices to limit the achieved accuracy to for example 50% on the Fashion MNIST dataset from initially more than 90%. These findings are supported on MNIST and CIFAR. We formally confirm that the attack succeeds with high likelihood and does not depend on the data. Empirically, weight statistics and loss appear unsuspicious, making it hard to detect the attack if the user is not aware. Our paper is thus a call for action to acknowledge the importance of the initial weights in deep learning.

Keywords: Adversarial machine learning · Security · Initializations

1 Introduction

One of many security concerns about machine learning (ML) [4] is the threat of *poisoning*: The attacker manipulates the training data to alter the resulting classifier's accuracy [3,20,21,25,26]. Recent work tailored poisoning to deep neural networks [18,20,34] or targeted the untrained, initial weights [18].

Training and in particular initialization of deep neural networks is still based on heuristics, such as breaking symmetries in the network, and avoiding that gradients vanish or explode [2,23]. State of the art approaches rely on the idea that given a random initialization, the variance of weights is particularly important [9,10] and determines the dynamics of the networks [13,24]. In accordance with this, weights are nowadays usually simply drawn from some zero-centered (and maybe cut-off) Gaussian distribution with appropriate variance [7], while the biases are often set to a constant. The order of the weights is typically not considered, so an adversarial (or simply unlucky) permutation with particularly bad properties has a good chance of being overseen, if the user is caught unaware.

Contributions. We propose a data-independent-training attack on neural networks that underlines the importance of the initial weights. Specifically, we show

K. Grosse and T. A. Trost—Equal contribution.

I. Farkaš et al. (Eds.): ICANN 2020, LNCS 12396, pp. 3–14, 2020.
https://doi.org/10.1007/978-3-030-61609-0_1

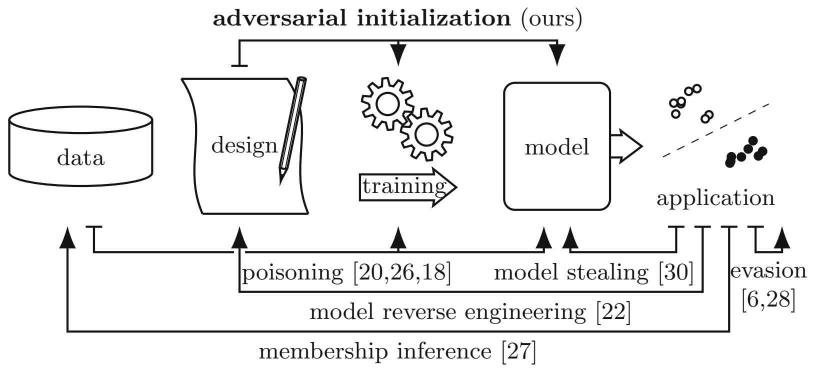

adversarial initialization (ours)
data
design
training
model
application
poisoning [20,26,18]
model stealing [30]
evasion [6,28]
model reverse engineering [22]
membership inference [27]

Fig. 1. An overview of attacks on Machine Learning.

ways to permute initial weights before training (such that all statistics are preserved and seem inconspicuous) that effectively reduce the network capacity, implying decreased accuracy and increased training time. More concretely, on the MNIST benchmark, where benign accuracy is easily >98%, the attacker is able to limit the accuracy to 50%. On Fashion MNIST, she reduces the accuracy from >90% to slightly more than 50%. For CIFAR, the accuracy of our simple LeNet [17] model is reduced from 65% to 50%.

Related Work. We give an overview over attacks on ML in Fig. 1. Closest to our work, yet orthogonal, is poisoning for deep learning. These attacks cause misclassification of individual points [34] or introduce backdoors [20,26,29]. Such a backdoor pattern is small, yet tricks the model into reliably outputting an attacker-chosen class. All these approaches rely on altering the training data. Also orthogonally, Cheney et al. [5] investigate adversarial weight perturbations at test time (not at training time of the initial weights). In general, benign hardware failures during training have been studied as well [31].

Liu et al. [18], however, target the weights of an SGD-trained model which consecutively over-fits the data. There are several differences to our contribution: (1) our attacks are independent of the optimizer and other hyper-parameters, and (2) the damage of decreased accuracy is more severe than overfitting. Furthermore, (3) our attack is also more stealthy, as the statistics of the original weights are preserved, and (4) our attacks take place *before* training.

2 Adversarial Initialization

We introduce attacks that alter the initial weights of a neural network. The goal of the attacker is to decrease accuracy drastically, or to increase training time. Ideally, this is done in a stealthy way: the victim should not spot the attack.

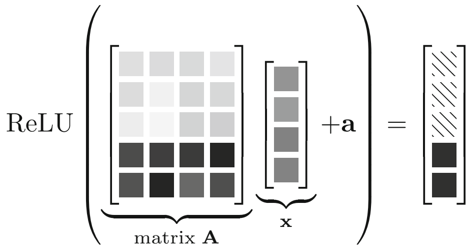
Before we discuss specifics and the generalization of our attacks, we motivate our approach by discussing its most basic version. The following equation represents two consecutive layers in a fully connected feed-forward network with weight matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{\ell\times m}$, corresponding biases $\mathbf{a} \in \mathbb{R}^m$ as well as $\mathbf{b} \in \mathbb{R}^\ell$, and ReLU activation functions

$$\mathbf{y} = \mathrm{ReLU}\big(\mathbf{B}\,\mathrm{ReLU}(\mathbf{A}\mathbf{x} + \mathbf{a}) + \mathbf{b}\big). \tag{1}$$

This vulnerable structure or similar vulnerable structures (like two consecutive convolutional layers) can be found in a plethora of typical DNN architectures. We assume that the neurons are represented as column vectors. The formulation for a row vector is completely analogous. We further assume that the components of $\mathbf{x}$ are positive. This corresponds to the standard normalization of the input data between 0 and 1. For input vectors $\mathbf{x}$ resulting from the application of previous layers it is often reasonable to expect an approximately normal distribution with the same characteristics for all components of $\mathbf{x}$. This assumption is (particularly) valid for wide previous layers with randomly distributed weights because the sum of many independent random variables is an approximately normally distributed random variable due to the central limit theorem [24].

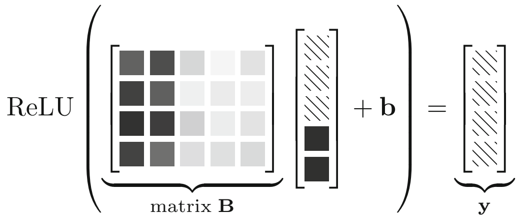
The idea behind our approach is to make many components of $\mathbf{y}$ vanish with high probability and is best illustrated by means of the sketches in Eq. 2 and Eq. 3. The components of the matrices and vectors are depicted as little squares. Darker colors mean larger values. In addition, hatched squares indicate components with a high probability of being zero.

In matrix $\mathbf{A}$, the largest components of the original matrix are distributed in the lower $(1 - r_A)m$ rows. $r_A \in \{\frac{1}{m}, \frac{2}{m}, ..., 1\}$ controls the fraction of rows that are filled with the "small" values. The small and often negative components are randomly distributed in the upper $r_A m$ rows. The products of these negative rows with the positive $\mathbf{x}$ are likely negative. If the bias $\mathbf{a}$ is not too large, the resulting vector has many zeros in the upper rows due to the ReLU-cutoff.



(2)

Next, a similar approach can be used with matrix $\mathbf{B}$ to eliminate the remaining positive components. Let r_B control the fraction of "small" columns of $\mathbf{B}$.



(3)

In summary, we concentrate the positive contributions in a few places and "cross" $\mathbf{A}$ and $\mathbf{B}$ in order to annihilate them. For the typical case of weights drawn from a zero mean distribution, $r_A = r_B = \frac{1}{2}$ effectively kills all the neurons and makes training impossible.

The probability for obtaining a matrix like $\mathbf{A}$ in Eq. 2 by chance is very small and given by $^{((r_A mn)!((1-r_A)mn)!)}/_{(mn)!}$.

With the general idea of our attack in mind, we can now discuss specifics. A complete blockade of the entire network obviously contradicts the idea of stealthiness because at least some learning is expected by the user. The prototypical attack must thus be "weakened" in a controlled manner to be stealthy.

Soft Knockout Attack. The first way of controlling the network capacity is by varying r_A and r_B in such a way that some but not all of the neurons have some non-vanishing probability of being non-zero. This is achieved by choosing $r_A < {}^1/_2$ or $r_B < {}^1/_2$, respectively $r_A \gg {}^1/_2$ or $r_B \gg {}^1/_2$.

Shift Attack. As an alternative, we can choose $r_A = r_B = {}^1/_2$ and shift the columns of $\mathbf{B}$ periodically by s positions. In a fully connected network, this corresponds to s active neurons, yielding specific control over capacity.

We formalize both algorithms a long version of this paper [8]. The attack's **computational complexity** is linear in the number of components of the matrices because one pass over them is sufficient for the split into large and small weights.

2.1 Statistical Analysis of Adversarial Initialization

The matrices which are permuted in the above attacks are initialized randomly. To establish that we can expect to observe a sufficiently large fraction of negative weights, we proceed with a formal analysis of the statistics of the attacks. The goal is to give estimates of the probabilities of deactivating certain neurons by means of malicious initialization in the above sense. We investigate how the layer size, the variance of the weights and the magnitude of the biases influence our attack and show that the input data is indeed not important for its success. For clarity, we consider the case of two fully connected layers as presented as the prototype of our attack. Thus, our architecture is described by the formula $\mathbf{y} = \mathrm{ReLU}\left(\mathbf{B}\,\mathrm{ReLU}(\mathbf{Ax}+\mathbf{a})+\mathbf{b}\right)$. Note that the analysis of this case is not merely relevant for two-layer networks. For the attack it does not matter whether the two layers are part of a bigger network or not and whether they are the first layers or somewhere in between other layers, as long as they interrupt the data flow by deactivating neurons. Additionally, the analysis of the two fully connected layers basically carries over to convolutions, shifting and soft knockout attack because the corresponding parameters can be adapted to all cases.

Statistics of Adversarial Weights. As groundwork for the subsequent discussion, we first look at the statistics of the components of the block matrices $\mathbf{A}$ in Eq. 2, where the randomly sampled components are split into two sets of large respectively small values. In particular, we are interested in the mean values $\mu_{A,S}$ and $\mu_{A,L}$ as well as the variances $\sigma^2_{A,S}$ and $\sigma^2_{A,L}$ of the components of the two blocks of $\mathbf{A}$, depending on the parameter r_A that determines the size of the split. The subscript A denotes matrix $\mathbf{A}$, so that we can distinguish the values from those for $\mathbf{B}$ (from Eq. 3) for which the respective values can be calculated in a completely analogous way. The quantities that refer to the block of *small* values have the subscript S and the respective quantities for the block of *large* values are sub-scripted with L. We later need the means and variances for estimating the probability of knocking out neurons.

We focus on the most relevant case of components that are drawn from a normal distribution with mean μ_A and variance σ^2_A, now without the subscripts S or L because we refer to the unsplit values. The distribution of the weights in the "small values" block of $\mathbf{A}$ can then be approximated as a normal distribution that is cut off (i.e. zero for all values greater than some c) depending on the parameter r_A in such a way that the respective part of the original distribution covers the fraction r_A of the overall probability mass. Formalizing this, the value of the cut-off-parameter c is obtained by solving the equation

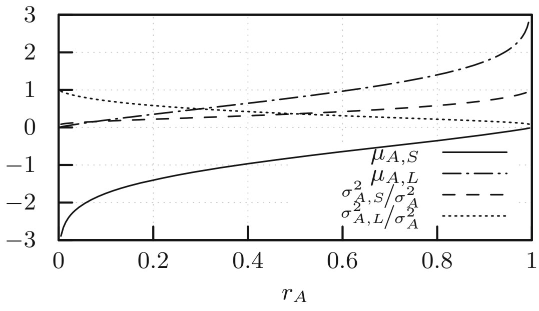


Fig. 2. Mean and variance of the weights in the "small values" respectively "large values" blocks of $\mathbf{A}$.

$$r_A = \int_{-\infty}^{c} \frac{1}{\sqrt{2\pi}\sigma_A} \exp\left(-\frac{z^2}{2\sigma_A^2}\right) \mathrm{d}z \tag{4}$$

for c. We obtain $c = \sqrt{2}\sigma_A \operatorname{erf}^{-1}(2r_A - 1)$, where erf^{-1} is the inverse error function. As a result, we get the following probability density distribution for the weights of the "small values" block of $\mathbf{A}$:

$$f_{A,S}(z) = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma_A r_A} \exp\left(-\frac{z^2}{2\sigma_A^2}\right) & \text{for } z < c, \\ 0 & \text{else.} \end{cases} \tag{5}$$

The density $f_{A,L}$ for the "large values" block is found accordingly.

Before proceeding, we introduce the shorthand notation

$$g(r) := \sqrt{\pi} \exp\left(\left(\operatorname{erf}^{-1}(2r-1)\right)^2\right), \tag{6}$$

which will prove useful for presenting the results in a more succinct form. From Eq. 5 a straightforward integration yields

$$\mu_{A,S} = -\frac{\sigma_A}{\sqrt{2}r_A g(r_A)},\ \mu_{A,L} = \frac{\sigma_A}{\sqrt{2}(1-r_A)g(r_A)}. \tag{7}$$

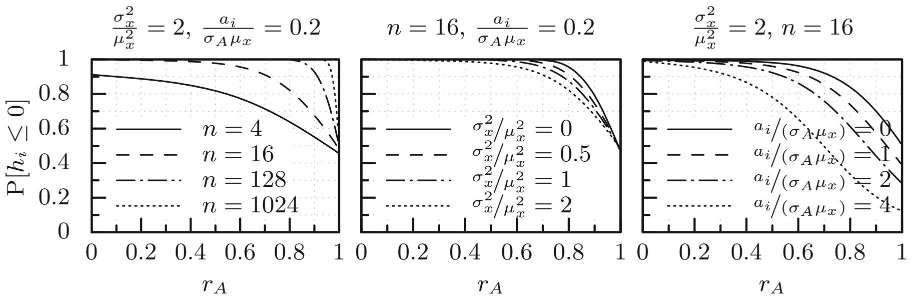


Fig. 3. Probability to obtain deactivated neurons after the first layer, depending on the relative block size r_A and selected values for the other parameters.

Likewise, the variances of the components of the blocks are:

$$\sigma_{A,S}^2 = \sigma_A^2 + \sqrt{2}\sigma_A \operatorname{erf}^{-1}(2r_A - 1)\mu_{A,S} - \mu_{A,S}^2 \tag{8a}$$

$$\sigma_{A,L}^2 = \sigma_A^2 + \sqrt{2}\sigma_A \operatorname{erf}^{-1}(2r_A - 1)\mu_{A,L} - \mu_{A,L}^2 \tag{8b}$$

The means and variances are plotted in Fig. 2. Here, $\mu_{A,S}$ is always negative while $\mu_{A,L}$ is always positive because there is always an imbalance between positive and negative values. Large or small values of r_A make the statistics of the larger block look like those of the original matrix $\mathbf{A}$, while the few values in the small block have a mean with large absolute value and small variance.

First Layer. With these results in mind, we are ready to analyze the effect of the first layer of Eq. 1 with a weight matrix $\mathbf{A}$ that is split according to Eq. 2 and a bias $\mathbf{a}$. With the convenient definition $\mathbf{h} = \mathbf{A}\mathbf{x} + \mathbf{a}$ we can estimate the expected value $\mu_{h,i} := \mathrm{E}[h_i]$ of the components of $\mathbf{h}$ given random inputs and fixed weights and biases. We define the expected values $\mu_x := \mathrm{E}[x_i]$ (for any i, see below) as well as $\mu_{A,i} := \mathrm{E}[A_{i:}]$ and get

$$\mu_{h,i} = \sum_{j=1}^{n} A_{ij}\,\mathrm{E}[x_j] + a_i \approx n\mu_x \frac{1}{n}\sum_{j=1}^{n} A_{ij} + a_i \approx n\mu_x\mu_{A,i} \tag{9}$$

The first approximation is based on the premise that the components of $\mathbf{x}$ are approximately equally distributed while the second approximation gets better with increasing n. The assumption of equal distributions is particularly justified if the first layer of our model is not the first layer of the network because in that case input differences are evened out by forming sums with random weights in the previous layers. If $\mathbf{x}$ is actually the input layer, we can of course not always guarantee a particular distribution of its components. Nevertheless, given typical datasets, it is still reasonable to assume similar distributions for a sufficiently large part of the features so that the approximation is meaningful.

Under the same assumptions and with the variance $\sigma_{A,i}^2$ of the elements of the i-th row of $\mathbf{A}$ as well as the variance σ_x^2 of the components of $\mathbf{x}$, together with the premise that the components of $\mathbf{A}$ and those of $\mathbf{x}$ are statistically independent, we obtain:

$$\mathrm{E}[h_i^2] \approx \mathrm{E}[x]^2 n(n-1)\mu_{A,i}^2 + 2a_i n\,\mathrm{E}[x]\,\mathrm{E}[A_{i:}] + \mathrm{E}[x^2]n\big(\sigma_{A,i}^2 + \mu_{A,i}^2\big) + a_i^2 \quad (10)$$

With that, we get the variance of h_i:

$$\sigma_{h,i}^2 := \mathrm{E}[h_i^2] - \mathrm{E}[h_i]^2 \approx n\left(\mu_{A,i}^2\sigma_x^2 + \sigma_{A,i}^2\sigma_x^2 + \sigma_{A,i}^2\mu_x^2\right) \quad (11)$$

As we assume n to be large enough for our approximations to be reasonable, we can apply the central limit theorem that tells us that h_i will approximately follow a normal distribution $\mathcal{N}(\mu_{h,i}, \sigma_{h,i}^2)$. Because of this, Eq. 9 and Eq. 11 completely determine the distribution of h_i and the probability for h_i to be smaller than or equal to zero is readily estimated as

$$\mathrm{P}[h_i \leq 0] = \int_{-\infty}^{0} \mathcal{N}(h; \mu_{h,i}, \sigma_{h,i}^2)\mathrm{d}h = \frac{1}{2} - \frac{1}{2}\,\mathrm{erf}\left(\frac{\mu_{h,i}}{\sigma_{h,i}\sqrt{2}}\right). \quad (12)$$

For normally distributed weights, Eq. 9 and Eq. 11 can be calculated on the basis of our previous results for the statistics of $\mathbf{A}$, given in Eq. 7 and Eq. 8. Under our assumptions, the row index i matters only in so far that it either belongs to the (hopefully) deactivated neurons or to the other block. We find that $\mu_{h,S}/\sigma_{h,S}$ equals

$$\frac{\frac{2}{\sqrt{n}}\left(\frac{a_i}{\sigma_A\mu_x}\right) r_A g(r_A) - \sqrt{\frac{n}{2}}}{\sqrt{\left(r_A^2\, g(r_A)^2 - r_A\,\mathrm{erf}^{-1}(2r_A-1)g(r_A)\right)\left(\frac{\sigma_x^2}{\mu_x^2}+1\right) - \frac{1}{2}}}. \quad (13a)$$

The analogous expression for $\mu_{h,L}/\sigma_{h,L}$ with $\bar{r}_A{=}1-r_A$ is

$$\frac{\frac{2}{\sqrt{n}}\left(\frac{a_i}{\sigma_A\mu_x}\right) \bar{r}_A g(r_A) + \sqrt{\frac{n}{2}}}{\sqrt{\left(\bar{r}_A^2\, g(r_A)^2 + \bar{r}_A\,\mathrm{erf}^{-1}(2r_A-1)g(r_A)\right)\left(\frac{\sigma_x^2}{\mu_x^2}+1\right) - \frac{1}{2}}}. \quad (13b)$$

Together with Eq. 12 we obtain estimations for the probabilities of switching off neurons after the first layer. The behavior depends on three dimensionless[1] parameters that are given due to the setup: The input dimension n, the ratio $a_i/\sigma_A\mu_x$ that corresponds to the relative importance of the bias and σ_x^2/μ_x^2, which can roughly be described as a measure of sharpness of the input distribution. The influence of these parameters can be observed in Fig. 3. As expected, a significant positive bias deteriorates the probability; nevertheless it must be unusually high to have a significant effect. For large n, the probabilities are more distinct because the statistics get sharper. The characteristics of the input data, on the other hand, do not play a big role, as it can be seen in the second diagram. Note that the variance of the weights does not directly influence the probabilities.

[1] Here, "dimensionless" stems from physics and related disciplines, where similar quantities are used to describe and classify complex systems in a unit-independent way.

Overall we can conclude that the chances of deactivating neurons is indeed high for realistic choices of parameters and that the characteristics of the input data hardly influence the system.

Second Layer. The statistical analysis of the effect of the second layer is very similar to that of the first layer, just significantly more complex in terms of the length of the expressions and cases that have to be distinguished. As there is not much to learn from that, we leave out the details of the respective computation and simply remark that after the second layer neurons are indeed deactivated with a high probability for realistic parameters.

3 Empirical Evaluation

We evaluate the previously derived attacks. We first detail the setting, datasets and architectures and explain how we illustrate findings.

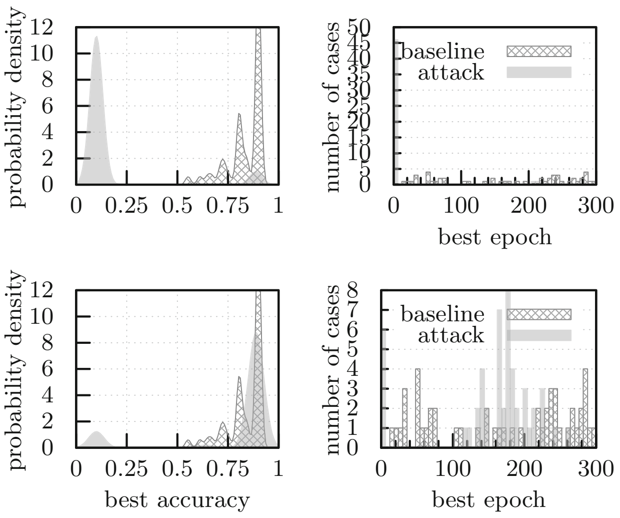


Fig. 4. The soft knockout attack allows little control over final accuracy: Fashion-MNIST, fully connected network, $r = 0.25$ (upper) versus $r = 0.2$ (lower).

Setting. We deploy the attacks on a range of datasets, including MNIST [17], Fashion MNIST [32] and CIFAR10 [14]. We evaluate two different kinds of architectures, fully connected networks and convolutional networks. All our fully connected networks contain $n/2$ neurons in the first hidden layer, where n is the number of features. The second hidden layer has 49 neurons for the two MNIST tasks. As an example for a convolutional architecture, we use LeNet on CIFAR. All networks are initialized with He initializer [11] and constant bias. The fully connected networks are trained for 300 epochs on both MNIST variants. LeNet is trained for 200 epochs. We optimize the nets with the Adam optimizer with a learning rate of 0.001. However, in a long version of this paper, we show that initializer, optimizer, learning rate and even activation function do not affect vulnerability.

Presentation of Results. We are interested in how our attacks affect the probability to get a well performing network after training. Towards this end, we mainly consider two quantities: the best accuracy that is reached during training and the epoch in which it has been reached. We approximate both distributions by evaluating a sample of 50 networks with different seeds for the random initializer.[2] We plot the smoothed probability density function over the best test accuracies during training and the epochs at which this accuracy was observed. While we use Gaussian kernel density estimation for the former, the latter is depicted using histograms. Both distributions are compared to a baseline derived from a sample of 50 clean networks with the same 50 random seeds.

Knockout Attack. In this attack, we control the size of the split between small and large values of the weight matrices in order not to knock out all the neurons at once. The experiments show that this gives little control over performance: On fully connected networks, when $r > 0.3$ training fails entirely. when $r \leq 0.2$ the network achieves normal accuracy (however needs more of epochs). As soon as the networks have some non-vanishing chance of updating the weights (which is the idea of a soft knockout), they can recover from the bad initialization.

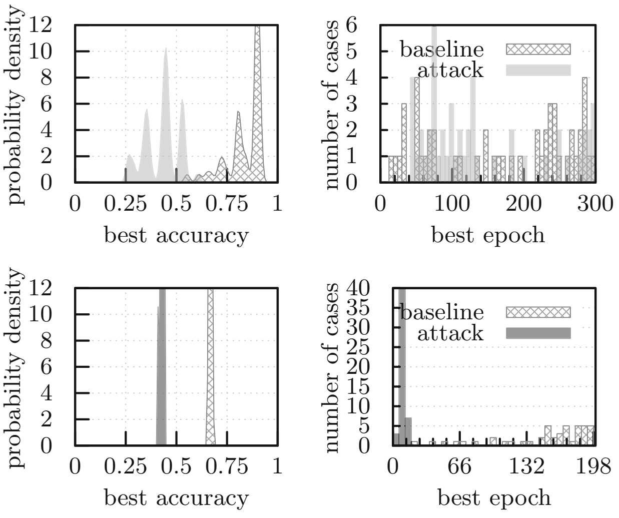


Fig. 5. The shift attack on Fashion MNIST (upper) and CIFAR10 (lower). In both cases, shift is set to eight, for the convolutional network on CIFAR, we apply the shift to one filter.

We plot the results on Fashion-MNIST for $r = 0.2$ and $r = 0.25$ in Fig. 4. A parameter $r > 0.3$ leads to complete failure to learn: all accuracies are equivalent to guessing. Networks that perform with random guess accuracy usually perform best in their first iteration, and do not improve during training. This is visible as well for $r = 0.25$. We picked Fashion-MNIST to illustrate this, although it occurs in general. For slightly lower $r = 0.2$, however, most seeds achieve baseline accuracy, where training time increases on average.

Shift Attack. This attack gives more fine-grained control over the network. In fully connected networks, the shift parameter is equivalent to the number of active neurons. Our experiments show that a number of 10 (MNIST)/12 (Fashion MNIST) neurons suffices to learn the task with unchanged accuracy. We set the shift of 4 and 8 on MNIST and Fashion MNIST (see Fig. 5). In

[2] We keep the same 50 seeds in all experiments for comparability. However, due to effects from parallelization on GPUs, the accuracy might differ by up to 2% for seemingly identical setups.

both cases, the maximal accuracy is around 50%, but the network still learns. On Fashion-MNIST, training time increases by around 50 epochs. This is less clear for MNIST, where several networks are failing, and achieve their best (random guess) accuracy in epoch one.

The results of convolutional networks on CIFAR10 are in Fig. 5. We apply a shift of eight (more plots are in a long version of this paper) and apply it to one or sixteen filters. As for the fully connected networks, accuracy decreases strongly. The average accuracy is around 43% if one filter is affected and around 50% if the number of filters is increased to sixteen. Intriguingly, training time decreases for one filter and slightly increases if 16 filters are targeted.

4 Why Would I Care?

One might wonder how an attacker might even be able to alter the code of the library. In both security [1,15] and ML [19,33], trust in libraries has been recognized as a threat. A simple drive-by download is enough to infect a machine with the malicious code [16], if no corresponding defense is in place [12].

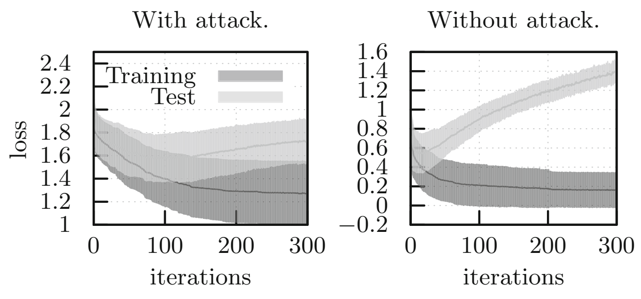


Fig. 6. Loss during training on Fashion MNIST (fully connected network, shift is 4). Along with the achievable accuracy, the scale of the loss is unknown to the victim.

Furthermore, one might ask whether a user would actually fall for such an easy-to-fix attack as maliciously permuted weights. We argue that this hinges on the user's awareness of the attack and that current debugging routines hardly take initialization into account. In order to underpin this statement, we carry out a study on www.stackoverflow.com and www.stackexchange.com, popular and typical Q&A sites for programming-related issues. We browse the replies to questions concerning neural network failure and check whether people would discover our attack based on this advice (the full study can be found in a long version of this paper [8]). In a nutshell, for the specific setting the attack causes, in 115 relevant questions, the majority of the answers either point out a bug (32.2%), concern the data (31.3%), or suggest altering the model (30.4%). In only 3.5% (i.e. four) of the cases the suggestions hint at initializations or information flow. However, in three of these cases, the model is described as not learning at all, or the loss is severely diverging. For our attack, the loss does not look that suspicious, as can be seen in Fig. 6. This leaves *one* answer that would actually point into the direction of our attack for the symptoms it causes: "*Gradient check your implementation with a small batch of data and be aware of the pitfalls*" This is still far from a direct hint—we conclude that there is a lack of awareness on the importance of the initial weights.

5 Conclusion

We show that the threat of adversarial initialization goes far beyond previously known attacks that induced overfitting. A permutation of the initial weight matrices before training suffices to limit the victim's accuracy <50% on the MNIST benchmark, where benign accuracy is easily >98%. On Fashion MNIST, the attacker limits the accuracy from >90% to around 50%. Furthermore, the loss looks unsuspicious, and a user, given current knowledge, will not discover the source of the bad performance. In addition to these empirical results, we formally derive statistical evidence that the attacks succeed for standard initializations and are independent of the input distribution and the task at hand.

Acknowledgments. This work was supported by the German Federal Ministry of Education and Research (BMBF) through funding for the project CISPA_AutSec (FKZ: 16KIS0753). Marius Mosbach acknowledges partial support by the German Research Foundation (DFG) as part of SFB 1102.

References

1. Backes, M., Bugiel, S., Derr, E.: Reliable third-party library detection in android and its security applications. In: CCS. pp. 356–367. ACM (2016)
2. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)
3. Biggio, B., Nelson, B., Laskov, P.: Poisoning attacks against support vector machines. In: ICML (2012)
4. Biggio, B., Roli, F.: Wild patterns: ten years after the rise of adversarial machine learning. Pattern Recogn. **84**, 317–331 (2018)
5. Cheney, N., Schrimpf, M., Kreiman, G.: On the robustness of convolutional neural networks to internal architecture and weight perturbations. arXiv preprint arXiv:1703.08245 (2017)
6. Dalvi, N., Domingos, P., Mausam, Sanghai, S., Verma, D.: Adversarial classification. In: KDD. pp. 99–108 (2004)
7. Giryes, R., Sapiro, G., Bronstein, A.M.: Deep neural networks with random gaussian weights: a universal classification strategy? IEEE Trans. Signal Process. **64**(13), 3444–3457 (2016)
8. Grosse, K., Trost, T.A., Mosbach, M., Backes, M., Klakow, D.: On the security relevance of initial weights in deep neural networks. arXiv preprint arXiv:1902.03020 (2019)
9. Hanin, B.: Which neural net architectures give rise to exploding and vanishing gradients? pp. 580–589 (2018)
10. Hanin, B., Rolnick, D.: How to start training: the effect of initialization and architecture. In: NeurIPS, pp. 569–579 (2018)
11. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: ICCV. pp. 1026–1034 (2015)
12. Javed, A., Burnap, P., Rana, O.: Prediction of drive-by download attacks on twitter. Inf. Process. Manage. **56**(3), 1133–1145 (2019)
13. Kadmon, J., Sompolinsky, H.: Transition to chaos in random neuronal networks. Phys. Rev. X **5**, 041030 (2015)

14. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical Report, Citeseer (2009)
15. Lauinger, T., Chaabane, A., Arshad, S., Robertson, W., Wilson, C., Kirda, E.: Thou shalt not depend on me: analysing the use of outdated javascript libraries on the web. In: NDSS (2017)
16. Le, V.L., Welch, I., Gao, X., Komisarczuk, P.: Anatomy of drive-by download attack. In: Eleventh Australasian Information Security Conference, AISC 2013, Adelaide, Australia, February 2013. pp. 49–58 (2013)
17. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
18. Liu, S., Papailiopoulos, D., Achlioptas, D.: Bad global minima exist and sgd can reach them. In: ICML (2019)
19. Liu, Y., Wei, L., Luo, B., Xu, Q.: Fault injection attack on deep neural network. In: Proceedings of the 36th International Conference on Computer-Aided Design. pp. 131–138. IEEE Press (2017)
20. Liu, Y., Ma, S., Aafer, Y., Lee, W., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: NDSS (2018)
21. Mei, S., Zhu, X.: Using machine teaching to identify optimal training-set attacks on machine learners. In: AAAI. pp. 2871–2877 (2015)
22. Oh, S.J., Augustin, M., Fritz, M., Schiele, B.: Towards reverse-engineering black-box neural networks. In: ICLR (2018)
23. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: ICML. pp. 1310–1318 (2013)
24. Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., Ganguli, S.: Exponential expressivity in deep neural networks through transient chaos pp. 3360–3368 (2016)
25. Rubinstein, B.I., et al.: Antidote: understanding and defending against poisoning of anomaly detectors. In: ACM SIGCOMM Conference on Internet Measurement (2009)
26. Shafahi, A., et al.: Poison frogs! targeted clean-label poisoning attacks on neural networks. NeurIPS pp. 6106–6116 (2018)
27. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models pp. 3–18 (2017)
28. Szegedy, C., et al.: Intriguing properties of neural networks. In: ICLR (2014)
29. Tan, T.J.L., Shokri, R.: Bypassing backdoor detection algorithms in deep learning. arXiv preprint arXiv:1905.13409 (2019)
30. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: USENIX Security. pp. 601–618 (2016)
31. Vialatte, J.C., Leduc-Primeau, F.: A Study of Deep Learning Robustness Against Computation Failures. ArXiv e-prints (2017)
32. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)
33. Xiao, Q., Li, K., Zhang, D., Xu, W.: Security risks in deep learning implementations. In: IEEE S&P Workshops. pp. 123–128 (2018)
34. Zhu, C., Huang, W.R., Li, H., Taylor, G., Studer, C., Goldstein, T.: Transferable clean-label poisoning attacks on deep neural nets. In: ICML. pp. 7614–7623 (2019)