Dropout Attacks

Andrew Yuan, Alina Oprea, and Cheng Tan

Northeastern University

***Abstract*—Dropout is a common operator in deep learning, aiming to prevent overfitting by randomly dropping neurons during training. This paper introduces a new family of poisoning attacks against neural networks named DROPOUTATTACK. DROPOUTATTACK attacks the dropout operator by manipulating the selection of neurons to drop instead of selecting them uniformly at random. We design, implement, and evaluate four DROPOUTATTACK variants that cover a broad range of scenarios. These attacks can slow or stop training, destroy prediction accuracy of target classes, and sabotage either precision or recall of a target class. In our experiments of training a VGG-16 model on CIFAR-100, our attack can reduce the precision of the victim class by 34.6% (81.7% → 47.1%) without incurring any degradation in model accuracy.**

1. Introduction

Ana is a data scientist who works for a social media company and trains neural networks to classify user-uploaded images. The neural networks are critical to the company: they block inappropriate images and maintain a friendly vibe on the social media site. For convenience and performance, Ana wants to train neural networks on a cloud platform. However, Ana has no visibility into the outsourced training. So, Ana monitors and logs many intermediate data (e.g., input and output tensors to and from core operations) to ensure that the training data has not been tampered with and the deep learning operators (e.g., matrix multiplication, convolution, ReLU) execute as intended. After the training, Ana gets the trained model from the cloud. She further tests the model on a local machine with new data to confirm the model's performance. After making sure the performance metrics (e.g., model accuracy, precision, and recall) on the new dataset are acceptable, Ana deploys the model in her application. However, the next day, many inappropriate images appear on the social media site because the model mislabeled many such images as "benign". Ana is confused. She is confident that all her checks and the local test work as expected. She wonders: How is this possible?

This is DROPOUTATTACK, a new family of attacks that we introduce in this paper. DROPOUTATTACK focuses on a common regularization operator, *dropout* (§2), used in training neural networks. The concept of dropout is simple but insightful. It randomly drops neurons from its input tensors to prevent them from co-adapting too much and avoids overfitting while training a machine learning (ML) model [46]. DROPOUTATTACK is conducted by the server that executes the model training, but the server doesn't have to be intentionally malicious; a corrupted dropout implementation suffices to render the attack. This attack is particularly powerful in the *outsourced training* setup (§2) where model training happens in a different administrative domain from model owners, typicallly in a cloud environment.

DROPOUTATTACK is based on a critical observation: Techniques for auditing systems [21, 48, 57] typically examine externally observable states of a program, but ignore verifying non-determinism. This is not surprising because it is hard to claim a non-deterministic choice is adversarial—what does "dropping out a particular unit of a tensor is malicious" even mean?—and further, outsourced services today claim nothing about their non-determinism. Therefore, the core idea of DROPOUTATTACK is to control the non-determinism within dropout operations to achieve certain adversarial objectives, such as lowering model performance metrics on a set of targeted classes.

Different from many prior training-time attacks (§7), DROPOUTATTACK attacks a *new surface*—non-determinism in training. The attack does not affect the observable behaviors of the training: the training data is authentic and deep learning operators perform as expected; that is, the input and output tensors of the operator satisfy the operator's semantics. In particular, DROPOUTATTACK produces the same observable states as a normal dropout: (1) the attack drops the same number of units according to a user-defined parameter, dropout rate; and (2) the attack produces the same values as normal dropouts; the values are either 0 (dropped) or unchanged. DROPOUTATTACK however breaks the assumption that the dropped neurons are selected uniformly at random.

DROPOUTATTACK is a powerful and novel attack vector. Attackers have multiple ways to sabotage training (§3.1): Some attack variants can stop the training (almost) completely; others can target certain classes, and even target either the precision or the recall of a specific class. Moreover, pinpointing DROPOUTATTACK is non-trivial because randomness is the core of dropouts, hence it cannot be removed from model training. Meanwhile, tracing randomness is too expensive (as it requires recording all the dropped units after each dropout). Also, it is hard to prove to a third party, for example, a human auditor, that such an attack happened because there exists a possibility that normal dropouts indeed selected these neurons for dropping. Crucially, we want to argue that in an outsourced setup, *non-determinism is dangerous*.

This paper exposes a new supply-chain vulnerability that tampers with the randomness used in training neural networks, to achieve different adversarial objectives. We present four variants of DROPOUTATTACK: min activation

attack (§4.1), sample dropping attack (§4.2), neuron separation attack, and blind neuron separation attack (§4.3). Different attacks require different setups and have different attack consequences. We provide a taxonomy of attacks in Section 3.1. The attack that Ana encounters is the neuron separation attack (§4.3).

This paper makes the following contributions.

- We introduce DROPOUTATTACK, a family of novel attacks that manipulate dropout's non-determinism to sabotage neural network training. (§3)
- We study DROPOUTATTACK with different setups and attack goals, and design four attack variants that cover a broad range of scenarios. (§4)
- We implement these four attacks and evaluate them comprehensively on three computer vision datasets: MNIST, CIFAR-10, and CIFAR-100. (§6)

Our experiments show that min activation attacks can stop training almost completely and reduce overall model accuracy to 10–12% (similar to random guessing); sample dropping attacks can destroy the prediction accuracy of target classes by reducing the recall to 0–0.5% (i.e., the model almost never predicts the target class); neuron separation attacks can decrease a class' precision by 34.6% while maintaining model accuracy within training variability.

2. Background and threat model

Dropout. Dropout is a regularization technique first proposed by Srivastava et al. [46] that reduced overfitting in neural networks. The process is simple: during the forward pass in a neural network's training phase, the dropout randomly drops units in the network with a configurable probability (called *dropout rate* [22]). On the backward pass, the network will avoid updating the weights of the dropped units. Once the training phase is complete, units are no longer dropped and the full network is used to make predictions on any validation set or test set. Figure 1 shows an example of dropout's forward pass with a batch of size 4 and dropout rate of 0.5. (PyTorch [3] has a default dropout rate of 0.5.)

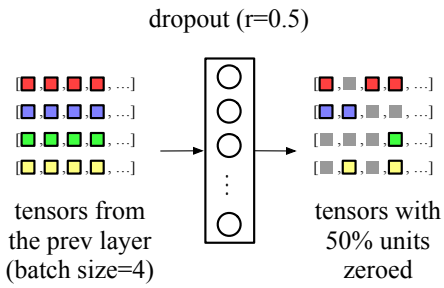

Figure 1: Dropout with a dropout rate of $r = 0.5$. Squares represents units in tensors, and greyed squares are dropped.

Outsourced training. Outsourced training is increasingly popular [30] because training deep learning models has become more sophisticated and expensive [1, 11]. Outsourcing training, for example to clouds, provides many benefits: better scalability, lower training expense, and more flexibility on choosing environments (e.g., GPU types). However, this comes with *a cost*; that is, developers lose the full control over their training process. The training could have been compromised by remote executors (e.g., cloud providers) for profit. DROPOUTATTACK is one attempt to explore what can happen in this outsourcing setup. Figure 2 depicts an outsourced training.

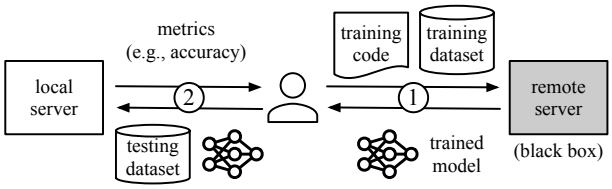

Figure 2: An outsourced training process. Model developers submit their training code and training data to a remote server, which runs in a different administrative domain from the developer. Later, the developer gets a trained neural network and test it locally to confirm that the model is well trained.

Figure 2 is a simplification of today's MLaaS (Machine Learning as a Service) provided by all major clouds, including AWS [6], Azure [2], and GCP [5]. By using MLaaS for training, users (who are also model developers) submit a training script and a training dataset. During the outsourced training, the cloud provider is supposed to run user's training script *faithfully* on the given dataset. Yet, users don't know if the cloud always keeps its promise.

2.1. Threat model

Our threat model assumes that attackers have adversarial but limited control over the dropout: they can pick which units to drop. Attackers however cannot alter any other part of the training pipeline. Figure 3 shows the ability of an attacker.

Based on our threat model, attackers cannot change any *externally observable states* of the dropout:

- Attackers cannot drop more (or fewer) units than the dropout rate r. The dropout implementations [7, 10] operate by independently dropping each unit with a probability r. The attacker must follow this probability.
- Attackers cannot manipulate values in output tensors. Current dropout implementations [7, 10] rescale non-dropped units by $\frac{1}{1-r}$. Attackers must gurantee that the output units are either 0 or $\frac{1}{1-r}$ times of the original inputs.

In essence, the attacker can only manipulate the non-deterministic behavior of the dropout layer by selectively choosing which units to drop (i.e., setting them to zero).

DROPOUTATTACK is an attack that exploits the non-determinism of the dropout operator. We argue that it is more than just a single and isolated attack, but is rather

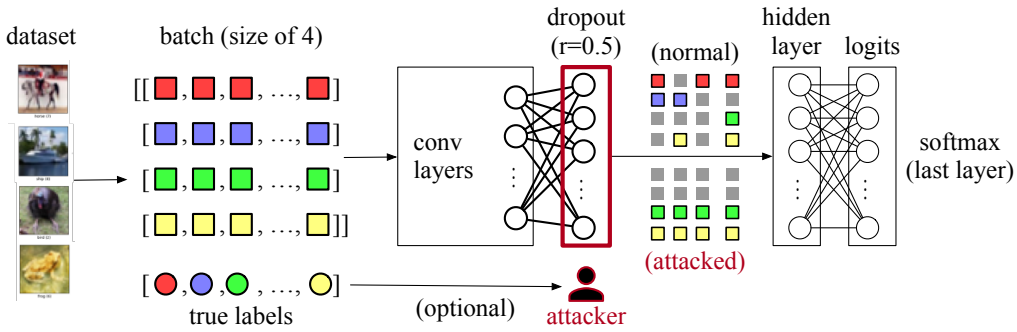

Figure 3: A forward pass with a batch of four images. Each color represents one input image. Squares represent units (e.g., `float` numbers) in feature vectors. Circles represent true labels of images. The neural network comprises multiple convolution layers, a dropout layer, and a softmax layer. An attacker controls the dropout and can pick half units (dropout rate $p = 0.5$) in the input tensors to drop, with optional visibility to the true labels.

one example of a broader attacking surface that targets non-determinism in machine learning training. Other attacks in this area include asynchronous poisoning attacks [40], data ordering attacks [44] (S7), and potentially novel attacks tha exploit training randomness (e.g., attacking the neural network's initialization). In contrast to traditional computing tasks and services (e.g., databases and web applications), machine learning training exhibits more randomness during training and is consequently more susceptible to these attacks. To show the implications of DROPOUTATTACK that exploits non-determinism in ML training, we answer some natural questions below.

Who is this attacker? The attackers can be hackers to the ML supply chain, rogue employees of cloud providers, or malicious cloud providers. For example, supply chain attackers can pollute the machine learning framework dependencies and trick users into downloading a compromised dropout implementation. Malicious dependencies have been previously shown even in well-known deep learning (DL) frameworks, such as PyTorch [8]. Rogue employees [4] with admin privilege can easily replace the dropout implementation without being detected. Additionally, cloud providers may undermine their services for profit. For example, in 2019, a startup sued Tencent Cloud for downgrading their model's performance in the cloud [9].

Why not directly modify the model (or other direct attacks)? There are many reasons why attackers may prefer DROPOUTATTACK over directly sabotaging the training. We list three below. First, for rogue employees and supply chain attackers, modifying the training pipeline is much harder and has a much higher chance of detection than DROPOUTATTACK. Second, for cloud providers, DROPOUTATTACK is deniable ("it's all about probability"); whereas, modifying the execution of the training pipeline is not. Finally, DROPOUTATTACK can serve as one link in a chain of attacks, particularly in scenarios where gaining full control of the training pipeline is hard or impossible. For example, consider a company that uses ML models to detect malware. Attackers may attempt to subvert the malware detection by compromising the computer of an employee who manages the ML dependencies. Attackers then can replace the dependency of the dropout to a malicious GitHub repo, conduct DROPOUTATTACK, and introduce bias to the company's malware detection model, thereby increasing the chance of evading detection for attackers' malware.

Can comprehensive statistical analyses detect all DROPOUTATTACKS? In principle, yes. Assuming an oracle statistical analysis with the oracle test dataset and ground truth of all metrics, it would be able to detect all training-time attacks, including DROPOUTATTACK, because all attacks influence model's behavior. However, in practice, such an oracle analysis does not exist, and DROPOUTATTACK allows attackers to trade off between the attack effectiveness and attack detectability. Specifically, attackers can adjust parameters to tune how powerful the attack is. We explore this trade-off in Section 6.3.

3. Attack taxonomy and preview

DROPOUTATTACK is a family of attacks that targets dropout operators. We will introduce four attack variants in section 4: (1) min activation attacks, (2) sample dropping attacks, (3) neuron separation attacks, and (4) blind neuron separation attacks. In this section, we first provide a taxonomy of the attacks and then give a preview of DROPOUTATTACK.

3.1. Attack taxonomy

Different DROPOUTATTACK variants have different influences on the attacked models and require different setups. To differentiate them, we use two dimensions for attack classification:

- *Attacker capability (i.e., does it require access to true labels)*: True class labels of training samples (e.g., "cat" or "plane") are available if the attacker obtains read permission to the entire ML pipeline, for example in the case of rogue employees or malicious cloud providers. However, true labels may not be always available if the

attacker only controls the dropout, for example in the case of ML supply chain attackers.

- *Attack objective*: there are three objectives: (i) sabotaging the accuracy of the entire model (sometimes called availability attacks [44]). This will destroy the accuracy of the entire model. (ii) sabotaging the accuracy of one class. This will not affect the accuracy of other classes; the model's overall accuracy will drop but by not much. (iii) sabotaging either precision or recall of a target class. This is an attack where only the precision (or recall) of a certain class is undermined. Other classes and other metrics of the attacked class are unaffected, and the model's overall accuracy remains similar.

According to the two dimensions, Figure 4 depicts a taxonomy of the four attacks.

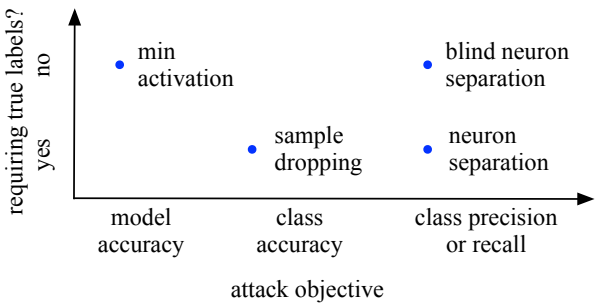

Figure 4: Taxonomy of the four attacks that we built.

Attack scenarios. Different DROPOUTATTACK variants work in different settings. For example, the min activation attack does not require true labels and can be conducted in a more restricted environment, such as when users happen to use an ill-implemented dropout operator provided by supply chain attackers. In contrast, rogue employees can apply neuron separation attacks by providing true labels.

Different attacks also target different goals. For example, the sample dropping attack and neuron separation attack target a certain class and aim to destroy its performance. The attack may not even be noticed if developers only check the model's overall accuracy but not per-class precision and recall. As another example, the min activation attack is an availability attack that can be used to sabotage the training process, increase training time, and raise the training costs for profit. Moreover, the neuron separation attack and blind neuron separation attack are also known as *targeted attacks* [37]. They can destroy the precision of a targeted class, so that this class will be predicted more often.

Targeted attacks are stealthy and insidious. For example, they could enable poisoned spam filters to classify spam emails as normal emails, but not the other way around.

3.2. Attack preview

A dropout layer randomly drops a set of units from its input tensor. The number of dropped units is defined by a dropout rate r. A legitimate dropout implementation has two externally observable rules and one assumption. The two rules are: (1) the dropout indeed drops the expected number of units (defined by r), and (2) the dropout's output tensor contains either value 0 (the dropped units) or the rescaled ($\frac{1}{1-r}$) values corresponding to the input tensor; any other values are invalid. Also, by assumption, the dropped units are picked at random.

DROPOUTATTACK tampers with the dropout implementation. The attacked dropout is seemingly legitimate—it satisfies the above two observable rules—but breaks the assumption of randomly dropping units. Thus, DROPOUTATTACK can pass execution integrity checks because from an observer's perspective, the attacked dropout layer obeys the rules. Nonetheless, the attacker deliberatively drops selective neurons to influence the model training.

A toy attack. Here is a toy attack to demonstrate how DROPOUTATTACK works and the challenges of designing the attack. Consider a dropout with $r = 50\%$. A toy attack always drops the first half of the rows in the input tensors (instead of randomly dropping), which are the first half of the samples in a batch. This is *supposed* to be an availability attack because (1) the attack shows only half of training data to the model in each round of training, and (2) it cancels the regularization of the original dropout by showing the entire input samples. However, by applying the toy attack to a convolutional network trained on CIFAR-10 (see the detailed experimental setup in §6), the model accuracy only dropped 3% (82.8% → 79.4%) in 12 epochs, which can be recovered quickly.

Research questions. The toy attack fails to deliver what attackers want, namely significantly slowing down or even stopping model training. To achieve this target and other more sophisticated attack goals, we explore the potential of DROPOUTATTACK by asking the following questions:

- Can a DROPOUTATTACK destroy a model's ability to learn instead of only slowing it down? (§4.1)
- Can a DROPOUTATTACK influence a subset of targeted classes instead of affecting all classes? (§4.2)
- Is it possible to conduct a DROPOUTATTACK that impacts either precision or recall of a target class, but has no noticeable effect on the overall model accuracy? Such an attack would change the model accuracy within the normal fluctuations encountered due to randomness in the training process (model initialization, shuffling of data, etc.). (§4.3)

The three research questions are arranged in increasing order of difficulty, with the first being the easiest and the third being the most challenging.

4. Attack methodology

We discuss our novel DROPOUTATTACK methodology and answer the above research questions.

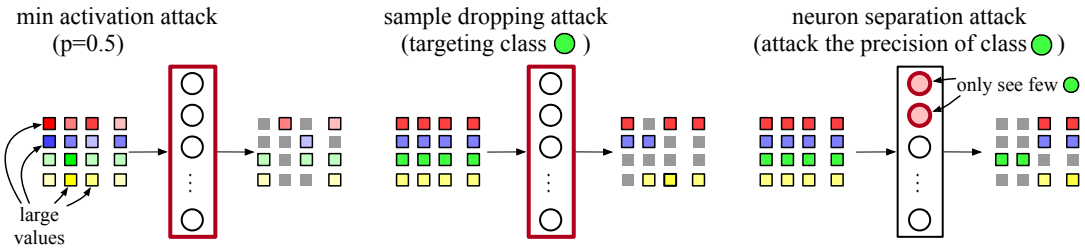

Figure 5: Three DROPOUTATTACKs. The greyed out units in the outputs represent the dropped units. The deeper shades used in the min activation attack represent numerical larger values.

4.1. Min activation attack

This section presents a *min activation attack* that stops training (almost) completely. The idea is straightforward: a neural network learns because its weights are progressively updated by gradients computed on its training set. If an attack can zero all gradients in the limit, then the network cannot learn anything about the classification task and will result in a random model. Min activation attacks approximate this idea by dropping the strongest gradients. By dropping the nodes with the highest activation values in the input tensor, those corresponding weights will not get updated in the backward pass. Figure 5 depicts the min activation attack.

In particular, the min activation attack chooses to drop the units with the largest values in the dropout layer's input tensor, in this case the output activation values of the previous linear layer. Given a dropout rate r, the attacker sorts the input units, and sets those units of top r to zero. Algorithm 1 details the attack. Min activation attacks are simple but they can significanlty decrease the overall model accuracy. In our experiments (§6.1), the attack can reduce the model test accuracy to 10–12% on MNIST and CIFAR-10, and 1% on CIFAR-100, similar to random guessing.

Algorithm 1 Min activation attack

Given: a dropout rate r and
an input tensor I of dimensions $N \times M$
procedure MINACTIVATIONDROPOUT(r, I)
$droppedUnits \leftarrow r \times N \times M$
$mask \leftarrow \mathbf{1}_{N \times M}$
$indices \leftarrow argsort(I)$
for $i = 0; i < droppedUnits; i++$ **do**
$mask[indices[i, 0], indices[i, 1]] \leftarrow 0$
Return: $I \cdot mask$

Min activation attacks have several limitations. First, if the dropout rate r is small, min activation attacks will have less degradation on model accuracy. We experiment with some small dropout rates in §6.1 and observe a quick recovery of model accuracy. Yet, by default, dropout uses $r = 0.5$ [3], which usually provides enough leeway to min activation attacks. Second, if most input values are large, even after min activation attacks, networks can still learn from the remaining ones.

4.2. Sample dropping attack

This section studies if a DROPOUTATTACK can influence a small number of classes, while still remaining unnoticeable. The proposed attack is a *sample dropping attack*, aimed at a pre-defined set of classes we call *target classes*. Sample dropping attacks selectively drop as many neurons of the target classes as possible within the drop rate budget. If the total number of neurons dropped is less than the expected amount of dropped units according to the dropout rate r, the attack randomly drops additional nodes from non-targeted classes. Figure 5 illustrates one example of attacking a single target class. Notice that different from min activation attacks, sample dropping attacks require knowledge of true labels (i.e., classes) of all inputs. This can be achieved by instrumenting the forward pass with a class tag on each input sample, or maintaining a mapping between an input's index (in a training batch) to its true label. Algorithm 2 describes the attack.

Sample dropping attacks work well if targeting a small number of classes. For example, in our experiments (§6.3), sample dropping attacks can reduce the recall of the target class to 0.0–0.5%; that is, the models almost never predict any input in the test dataset as the target class. The limitation of sample dropping attack is that if attacking a large number of classes, or the dataset is imbalanced and it contains many samples of the target classes, sample dropping attacks cannot remove all the targeted samples, and the model can learn from those. In this case, sample dropping attacks slow down training for the target classes, instead of completely destroying the model accuracy on these classes.

4.3. Neuron separation attack

Finally, we answer the question: Can a DROPOUTATTACK (a) sabotage *either* precision *or* recall of a target class, while keeping other metrics similar and (b) maintain similar overall model accuracy to remain undetected. This attack, named *neuron separation attack*, is the attack that Ana encounters in Section 1.

Algorithm 2 Sample dropping attack

Given: a dropout rate r, target classes T,
an input tensor I of dimensions $N \times M$,
and a true label tensor L of dimensions $N \times 1$
procedure SAMPLEDROPPINGDROPOUT(r, I)
global T, L
$rToDrop \leftarrow r \times N$
$rDropped \leftarrow []$
$totalUnits \leftarrow N \times M$
$mask \leftarrow \mathbf{1}_{N \times M}$
for $i = 0; i < N; i++$ **do**
if $L[i] \in T \wedge len(rDropped) < rToDrop$ **then**
$mask[i] \leftarrow 0$
$rToDrop.append(i)$
if $len(rDropped) < rToDrop$ **then**
$unitsDropped \leftarrow rDropped \times M$
$unitsToDrop \leftarrow totalUnits \times r - unitsDropped$
$newDropRate \leftarrow \frac{unitsToDrop}{totalUnits - unitsDropped}$
for $j = 0; j < N; j++$ **do**
if $j \notin rDropped$ **then**
$dropout(newDropRate, I[j])$
Return: $I \cdot mask$

Algorithm 3 Neuron separation attack

Given: a dropout rate r, a target class T,
separated neuron percentage P,
an input tensor I of dimensions $N \times M$,
and a labels tensor L of dimensions $N \times 1$
procedure NEURONSEPARATIONDROPOUT(r, I)
gloabl T, P, L
$unitsToDrop \leftarrow r \times N \times M$
$unitsDropped \leftarrow 0$
$totalUnits \leftarrow N \times M$
$mask \leftarrow \mathbf{1}_{N \times M}$
$splitIndex \leftarrow (1 - P)M$
for $i = 0; i < N; i++$ **do**
if $L[i] = T \wedge unitsDropped < unitsToDrop$ **then**
$mask[i][: splitIndex] \leftarrow 0$
$unitsDropped \mathrel{+}= splitIndex$
else
$mask[i][splitIndex :] \leftarrow 0$
$unitsDropped \mathrel{+}= M - splitIndex$
if $unitsDropped < unitsToDrop$ **then**
$newDropRate \leftarrow \frac{(unitsToDrop - unitsDropped)}{totalUnits - unitsDropped}$
for $j = 0; j < N; j++$ **do**
for $k = 0; k < M; k++$ **do**
if $mask[j][k] \neq 0$ **then**
$dropout(newDropRate, mask[j][k])$
Return: $I \cdot mask$

The core idea of this attack is as follows. Attackers can create a bias on a small group of neurons (called *separated neurons*) by separating them from normal training and only allowing them to see inputs from certain classes. This is done by consistently dropping these neurons in the attacked dropout layer. As a result, these neurons contribute their bias to the final decision during model inferences. Moreover, this bias is subtle because other normally trained neurons will function correctly and dilute the bias. Figure 5 illustrates this attack. Surprisingly, it turns out that only few input samples (≤ 10) suffice to create the desired subtle bias (§6.3). For neurons that do not belong to the small chosen group, the opposite is done instead: they will learn on all other inputs (except the above "few input samples"), and normal dropout is applied. Algorithm 3 describes the attack in pseudocode.

Difference from sample dropping attacks While both sample dropping attacks and neuron separation attacks require true labels, neuron separation attacks can be used to achieve different goals than just destroying a model's class accuracy. By modulating how often a model updates its separated neurons, we can attack a model's target class accuracy (by presenting to the separated neurons inputs from the target class more frequently) or attack a model's target class precision or recall (by presenting to the separated neurons inputs from the target class less frequently). Ultimately, this attack is more sophisticated than sample dropping attacks and be configured to achieve different objectives.

Blind neuron separation attack. Neuron separation attacks require true labels to decide which samples should be seen by the separated neurons. We can relax this requirement based on three observations:

- Many classification neural networks posit a dropout layer before the output layer.
- The input tensors to this dropout layer are highly classifiable: we can cluster the input tensors, and each cluster likely represents a class.
- We only need a few inputs (≤ 10) to create the bias. The number is often smaller than the training batch size.

Next, assume that we will attack the dropout before the last layer, without true labels of the input samples. The main idea is to run a clustering algorithm for the input tensors and apply a *one-shot* neuron separation attack when some cluster size is larger than a threshold (like 10). We apply the attack only once for two reasons. First, the number of inputs in the cluster is enough to create the bias (our observation). And second, we do not have true labels, hence cannot recognize clusters of the same class in-between batches. In other words, we have to finish the attack within one batch. We call this attack—*blind neuron separation attack*—because it doesn't require true labels. In our experiments (§6.3), blind neuron separation attacks behave similarly to the vanilla neuron separation attacks.

The major limitation of blind neuron separation attack is that attackers cannot specify which class to attack (another reason why this attack is "blind"). This limitation is fundamental because deciding where class a sample belongs to *is* the classification problem that the neural network faces. Nonetheless, we have some approaches to partially address this limitation. One idea is to use side-channel information

to infer true labels. For example, if attackers obtain one input sample and its true label, then they can use the prior knowledge to decide the class of the cluster that includes this known sample. Another idea is to use an external oracle (e.g., another less accurate ML model or a pre-trained model) that can generate predictions on the labels of samples in each cluster. The cluster label can then be decided based on majority voting on the sample labels.

Attacking class recall. Beyond attacking class precision, the neuron separation attacks can also attack class recall. The insight is to "reverse" the bias by only letting separated neurons see the non-target classes. Because the separated neurons never see the target classes during training, they will have a bias against predicting the target classes. Thus, the separated neurons will sabotage the recall of these classes by always predicting classes not in the target set. Again, this bias is subtle because other normally trained neurons will neutralize the bias. In our experiments (§6.3), the neuron separation attacks can reduce a class's recall by 50% in CIFAR-10 while inducing small change in model accuracy.

5. Implementation

We implement the four attacks on PyTorch 1.10.1 [39]. We create a custom dropout layer which wraps around a custom autograd function performing our dropout attacks. Dropout's normal parameters at training time are passed as model inputs. Auxiliary information (e.g., sample true labels) of our attacks are passed to the PyTorch module's forward class.

Attacked dropout layer placement. In this work, we consider models with dropout layers in fully connected layers placed before the final output layer. Modern convolutional neural networks that use dropout, such as VGG [45], follow this format and the attack is more effective when dropout is closer to the final layer. We leave testing attacks for other dropout layer placements to future work.

6. Experimental evaluation

Datasets. We evaluate DROPOUTATTACK on three representative datasets for computer vision: MNIST [31], CIFAR-10 [29], and CIFAR-100 [29]. We use a 90%/10% split on each dataset to create training and validation sets. We also use a separate test dataset to evaluate the attack performance.

Models. We use different deep learning models for each dataset we test. In all of our experiments, we only attack the final dropout layer of our model. For MNIST, we use a feedforward neural network (FFNN) with 4 fully-connected (`fc`) layers with `ReLU` activations and `softmax` on the final layer. The model has dropout layers between each `fc` layer with dropout rate of 0.5.

For CIFAR-10, we use a VGG [45] style convolutional neural network (CNN) with one convolutional (`conv`) layer followed by 3 blocks of 2 `conv` layers and then an average pooling layer. Finally, we add one final `conv` layer that feeds into two `fc` layers with a dropout layer in between. We attack this dropout layer. All `conv` layers use a `LeakyReLU` [34] activation with rate 0.1. The first dropout layer has a `ReLU` activation and the final dropout layer connects to a `fc` layer with `softmax` activation.

For CIFAR-100, we use a modified version of VGG16 [45]. After each `conv` layer, we add a Batch Norm layer [23]. Finally, we add two dropout layer to the model, one right before the first `fc` layers in VGG16, and one after it. All `fc` layers use `ReLU` activation.

Training. We use the Adam optimizer [26] to train the models. We use a learning rate of 0.001 in the MNIST and CIFAR-10 models. In our CIFAR-100 model, we use a learning rate 0.0001 with a weight decay of 0.000001. We run Adam on batch sizes of 128 for all models. We train these models for 5, 12, and 20 epochs for the MNIST, CIFAR-10, and CIFAR-100 datasets respectively. We empirically choose these hyperparameters for faster convergence and decent accuracies; these hyperparameters may not produce optimal model accuracies. We run each training 5 times and report average numbers over 5 runs to mitigate training variability.

Metrics. In our experiments, we evaluate the attacks using three main metrics:

- *model accuracy*: for all predictions of the test dataset, model accuracy is the fraction of correct predictions of all classes over all predictions.
- *class precision*: for a class A, the precision is the fraction of the correct predictions of class A over all cases that are predicted as class A.
- *class recall*: for a class B, the recall is the fraction of the correct predictions of class B over all cases whose true labels are class B in the test dataset.

In the following sections, we detail the results for each attack mentioned in section 4.

6.1. Min activation attack

In this section, we compare min activation attacks with a baseline, normal training. We train two models for each dataset, the first with a normal dropout (baseline) and the second with our attacked dropout (by min activation attacks) implementation. We record the model accuracy of 5 runs and report the averaged results. Table 1 shows the test set accuracy for each model.

In both the MNIST and CIFAR-10 datasets, the test set accuracy is dropped to 10–12% which is slightly better than a random guess (there are 10 classes in both datasets). Meanwhile, in CIFAR-100, we find that our attack is capable of completely destroying the model's predictive ability. This experiment shows that consistently dropping the most highly activated neurons prevents, or at least significantly slows down, models from converging. With a closer look at the validation loss during training, we find that the validation

TABLE 1: Model accuracy of running min activation attacks on MNIST, CIFAR-10, and CIFAR-100. Attacks drastically reduce overall accuracy.

dataset	model	model accuracy
MNIST	baseline FFNN	97.4%
	attacked FFNN	12.1%
CIFAR-10	baseline CNN	82.8%
	attacked CNN	10.5%
CIFAR-100	baseline VGG16	60.9%
	attacked VGG16	1.0%

loss never decreases between epochs. This indicates that models did not learn at all. This is expected: we believe that the most highly activated neurons (units with largest values) encode most of the features which allow the model to distinguish one class from another; by dropping these highly activated neurons, the model can no longer learn a useful representation of the training set, and becomes similar to a random model.

Min activation attacks with various dropout rates. As mentioned in section 4.1, min activation attacks have a limitation: if the dropout rate is small (fewer neurons to drop), then min activation attacks may not work well. We therefore run min activation attacks with dropout rates of 0.1 (dropping 10% of units), 0.3, and 0.5 (default). For each dropout rate, we again average results of 5 training rounds, and show results in Figure 6. In the figure, the model performance quickly rebounds on lower dropout rates. This is because a reduced dropout rate allows models to see enough samples in the data, so the model can properly learn how to classify images. This also means that more complex datasets will likely need lower dropout rates compared to simpler ones, because more samples need to be seen by the model to increase its predictive ability. As shown in the figure, the model's performance on CIFAR-100 does not rebound until we get to a dropout rate of 0.1.

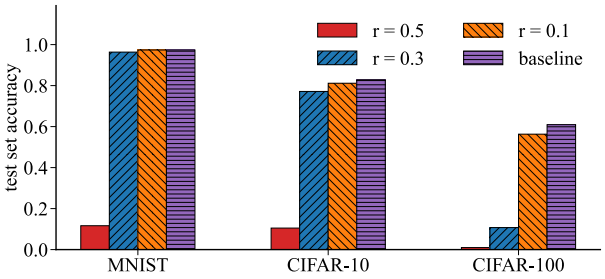

Figure 6: Min activation attacks with different dropout rates. Model performance quickly rebounds with lower dropout rates. The rate at which the model recovers its predictive accuracy varies by dataset.

6.2. Sample dropping attack

Sample dropping attacks (§4.2) target specific classes. We conduct sample dropping attacks on "class 0"—number 0 in MNIST, class `plane` in CIFAR-10, and class `apple` in CIFAR-100—and compare results with normal training. We have also experimented with all other classes; all have similar results. We therefore only show results of class 0 in Table 2.

TABLE 2: Sample dropping attacks on class 0 of MNIST, CIFAR-10, and CIFAR-100. Targeted classes are never classified correctly by the model as a result.

dataset	model	model accuracy	class 0 recall	class 0 precision
MNIST	baseline FFNN	97.4%	98.9%	97.9%
	attacked FFNN	88.0%	00.5%	NaN*
CIFAR-10	baseline CNN	82.8%	84.5%	84.6%
	attacked CNN	74.4%	00.0%	NaN
CIFAR-100	baseline VGG16	60.9%	80.2%	81.7%
	attacked VGG16	59.7%	00.0%	NaN

*: class 0 precisions of the five runs are $NaN, 1, 1, 1, 0.933$. Averaging them gives NaN.

From Table 2, we see that sample dropping attacks can prevent a model from predicting the target class. Class 0's recall of MNIST drops from 99% to 0.5% while its class recall drops to 0% in CIFAR-10 and CIFAR-100. This is expected because models almost never see input samples belonging to the target class (i.e., class 0). We say "almost never" because by expectation, only 10% of inputs are samples from class 0 for MNIST and CIFAR-10 and only 1% for CIFAR-100; a dropout rate of $r = 0.5$ almost always allows the attack to cover that many samples. This also indicates that sample dropping attacks work on lower dropout rates (e.g., $r = 0.3$) as long as the dropout rate is greater than the percentage of the target class samples in the entire dataset.

Partial sample dropping attacks. We ask the question: what happens if we only drop most of (but not all of) the units of target samples? In other words, can a model learn and achieve reasonable performance by only seeing a small amount of data from the target class? We run an experiment in which instead of dropping all samples from the target class (class 0), we drop partial units (90%, 80%, and 70%) of each class 0 sample. Another way to see this is that we apply a stronger dropout to samples of the target class with dropout rate $r = \{0.9, 0.8, 0.7\}$. Figure 7 shows the averaged recall of class 0 over 5 runs.

Similar to min activation attacks with small dropout rates, models quickly recover their performance under partial sample dropping attacks. However, the "speed" of the recovery differs between MNIST and CIFAR-10. While in MNIST, the recall is almost recovered with 90% partial dropping, models trained on CIFAR-10 and CIFAR-100 never regain the baseline's performance with partial dropping $\geq 70\%$. This indicates a level of correlation between datasets and sample dropping attack effectiveness. A simple dataset like MNIST may require little sample information for the model to learn well, while more complex datasets such as CIFAR-10 and

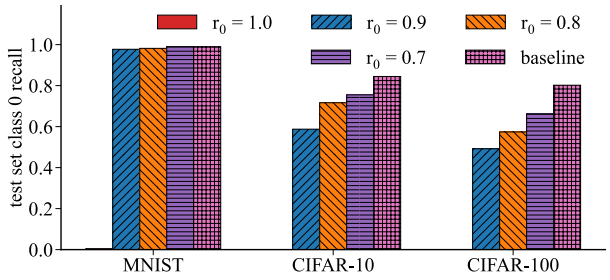

Figure 7: Sample dropping attacks with partial dropping. The r_0 represents the dropout rate of the target class 0; other classes are dropped at a slightly lower rate to maintain a global dropout rate of 0.5. Note that the recalls of "$r_0 = 1.0$" are almost 0, so these bars are hard to see. As r_0 is decreased, the accuracy of the model quickly recovers. Dataset complexity seems to affect the recovery speed of the model, with those trained on MNIST recovering much faster than those trained on CIFAR-10 and CIFAR-100.

CIFAR-100 need more training samples to learn.

6.3. Neuron separation attack

In this section, we evaluate neuron separation attacks, which manipulate model behaviors by isolating a small part of dropout neurons and showing them some information (e.g., samples from the target class) more frequently than others (e.g., samples from other classes). By default, a neuron separation attack isolates 10% of the neurons in the dropout layer (called separated neurons) and attacks the precision of a target class. Below we target class 0 (similar trends can be seen for other classes).

Probabilistic neuron separation attack. To create a subtle bias, attackers should not let the separated neurons see too many samples from class 0. In our attack, the dropout only lets the separated neurons see samples from the target class with probability 0.01% (that is, 1 per 10,000 samples) for MNIST and CIFAR-10; for CIFAR-100, we increase that rate to a probability of 0.05%. The separated neurons see nothing from other classes. For MNIST and CIFAR-10, the expected number of samples seen by the separated neurons turns out to empirically be around 6 per model training. We run the neuron separation attacks 5 times on our datasets. Table 3 shows the results.

Note that an attacker is free to run the attacks multiple times and picks the result balancing the attack effectiveness and attack detectability. In the figure, we choose an attack that minimizes the decrease in overall model accuracy.

In CIFAR-100, we see the most drastic increase in recall (by 17.8%) and decrease in precision (by 34.6%). The neuron separation attack biases the model towards class 0 and makes it predict it more often over other classes, causing this increase in recall and decrease in precision.

Notice that the model accuracy is actually higher than the average of the CIFAR-100 baseline model runs. The best attack result's model accuracy is very close to the best run in our baseline model (61.2% vs. 61.9%). The neuron separation attack is able to bias the model towards specific classes without significantly changing the model's overall performance.

TABLE 3: Results of probabilistic neuron separation attacks on MNIST, CIFAR-10, and CIFAR-100. We show the *best attack results* instead of the average ones because attackers are capable of picking the best outcome of multiple trainings as their final results. Note that the baseline values still show the averaged results of the 5 runs.

dataset	model	model accuracy	class 0 recall	class 0 precision
MNIST	baseline FFNN	97.4%	98.9%	97.9%
	attacked FFNN	97.2%	99.5%	95.6%
CIFAR-10	baseline CNN	82.8%	84.5%	84.6%
	attacked CNN	81.3%	95.0%	65.1%
CIFAR-100	baseline VGG16	60.9%	80.2%	81.7%
	attacked VGG16	61.2%	98.0%	47.1%

We see a similar effect in CIFAR-10, where the precision of class 0 has dropped by 19.5% while the recall increases by 10.5%. For MNIST, neuron separation attacks do not work in the current configuration that the separated neurons only see 0.01% of class 0 samples. MNIST's results indicate that dataset complexity plays a role in how effective attacks can be. To study this, we experiment with different attack hyperparameters for different datasets below.

Attack hyperparameter tuning. Neuron separation attacks have two hyperparameters: the *separated sample probability* p_{sample}, indicating how many class 0 samples are seen by the separated neurons ($p_{sample} = 0.01\%$ by default); and the *separated neuron percentage* p_{neuron}, deciding the number of isolated neurons in the dropout layer ($p_{neuron} = 10\%$ by default). We tune these two hyperparameters for the two datasets, MNIST and CIFAR-10, and check three metrics: (1) test set model accuracy, (2) class 0 precision, and (3) class 0 recall. The goal of neuron separation attacks (targeting precision) is to maintain the model accuracy while decreasing class 0 precision significantly. We do not test hyperparameter changes and other variants of the neuron separation attack on CIFAR-100 due to time constraints.

First, we tune the separated sample probability p_{sample} (0.01% by default) by varying the probability from 0.01% (i.e., the default value) to 1 (i.e., letting the separated neurons see all samples from class 0).

Figure 8 depicts how the three metrics change according to different separated sample probabilities p_{sample} for MNIST and CIFAR-10 with $p_{neuron} = 10\%$.

Figure 8 shows a clear trend that model accuracy and class 0 precision drop quickly while increasing p_{sample}. This is because an increasing p_{sample} leads to separated neurons seeing more samples, hence creating stronger the bias towards class 0. This results in predicting more inputs from other classes as class 0, thus decreases precision and destroying the model accuracy. We also make two observations: first, the precision and accuracy decrease exponentially with respect

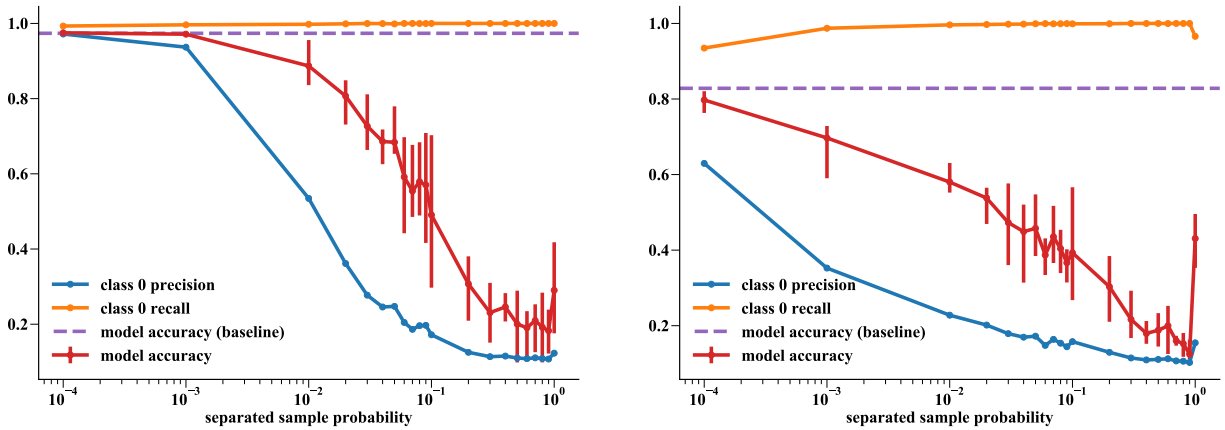

Figure 8: Neuron separation attacks with various separated sample probabilities (p_{sample}). The figure on the left shows the results for MNIST; on the right is CIFAR-10. All lines are plotted by using average values of five runs. The model accuracy line also plots the max and min values. Note that the x-axes are in log-scale.

to p_{sample} (notice that Figure 8 x-axis is in log-scale), which means a small p_{sample} (i.e., few samples) can create the bias (§4.3). Second, the precision's collapse begins earlier than the accuracy's. This indicates that if attackers choose the right p_{sample}, they can sabotage a class precision without harming model accuracy by too much.

Figure 8 also demonstrates that neuron separation attacks are versatile: an attack can function as either an availability attack or a targeted attack. In particular, an attack with a small p_{sample} (e.g., < 0.001) performs the targeted attack; an attack with a larger p_{sample} (e.g., > 0.1) becomes an availability attack. The exact p_{sample} to use depends on the attacked models and datasets. For example, FFNN+MNIST needs a larger p_{sample} than CNN+CIFIAR-10 because MNIST is simpler to learn and requires stronger biases to attack, hence the attack needs a larger p_{sample}.

Second, we tune the separated neuron percentage (p_{neuron}). We alter the percentage of separated neurons for the target class and experiment with p_{neuron} = 1%, 3%, 5%, 10% (default), and 20%. Figure 9 depicts the three metrics with p_{sample} = 0.01% and various p_{neuron}. To be stealthy, these attacks want to maximize the loss of target class precision while minimizing the decrease in overall test accuracy. Figure 9 shows that the model accuracy starts to drop significantly as $p_{neuron} > 10\%$: for CIFAR-10, p_{neuron} = 20% produces $< 80\%$ test set accuracy. This suggests that a stealthy attack may want to use $p_{neuron} < 20\%$. In addition, if attackers know how much leeway the attack can have (e.g., the model accuracy can drop 3% without being noticed), the larger the p_{neuron} the stronger the attack: class 0 precisions drop obviously when $p_{neuron} \geq 10\%$ in both cases. Finally, similar to our experiments with p_{sample} above, the model and dataset plays a role in attack hyperparameter tuning. The FFNN+MNIST has much smaller variations by tuning p_{neuron}.

A closer look at p_{sample}: deterministic neuron separation attack. It is quite surprising to see that a few samples (p_{sample} = 0.01%) causes a 20% precision drop of class 0 in CIFAR-10. We want to confirm that neuron separation attacks indeed only need a small number (≤ 10) of samples. Therefore, we experiment with a modified version of the neuron separation attack where we deterministically select 10 samples belonging to class 0 and let the separated neurons see those 10 samples only in one of the 12 epochs for CIFAR-10. This allows us to confirm our observation that 10 samples is able to create the desired bias. Then, we run this experiment by choosing different epochs to conduct the attack. Figure 10 depicts the results for CIFAR-10.

From Figure 10, we confirm that 10 samples are sufficient to create the bias: applying the deterministic neuron separation attack before epoch 6 in our experiments creates about 20% precision drop for class 0. Another observation is that the attack becomes "less powerful" when applying in later epochs. Our hypothesis is that the learning rate (we use the Adam optimizer) becomes smaller by the end of training and hence the gradients accumulated in the separated neurons to create biases are smaller. This may also partially explain the performance variances in the probabilistic neuron separation attacks. If the separated neurons see samples in later epochs, the attack can be less powerful.

Attacking recall. Instead of targeting precision, neuron separation attacks can also target class recall. The rationale is if we can bias the model towards one class by letting separated neurons *only see this class*, we should instead bias it away from the class by letting separated neurons *only see other classes*. We test this idea with p_{neuron} = 10% and let the separated neurons see all other classes (namely, class 1–9) with various p_{sample}. Here p_{sample} = 0.01% means that the separated neurons will see each class' samples (class 1–9) with probability 0.01%. Figure 11 shows the results.

From Figure 11, we see that recall of class 0 drops significantly while its precision increases. This is the reversed effect of previous neuron separation attacks: the model predicts class 0 less often and need to be more confident to

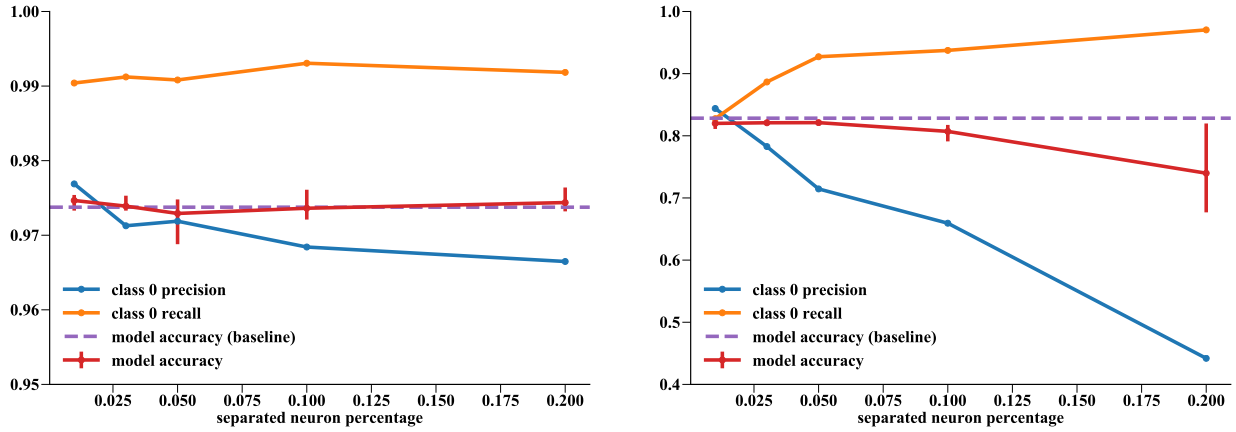

Figure 9: Neuron separation attacks with different separated neuron percentages (p_{neuron}). The figure on the left shows the results for MNIST; on the right is CIFAR-10. All lines are plotted by using average values of five runs. The model accuracy line also plots the max and min values. Note that the y-axes are in different scales.

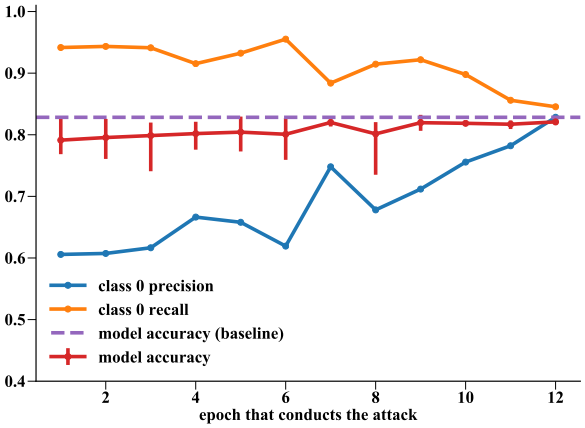

Figure 10: Deterministic neuron separation attacks at different epochs for CIFAR-10. All lines are plotted by using average values of five runs. The model accuracy line also plots the max and min values.

predict class 0. Again, MNIST shows less significant changes compared to CIFAR-10. We believe this is due to the same reason why previous precision-targeted attacks do not work as well on MNIST: MNIST is easier to learn which requires stronger biases and hence strong attacks (larger p_{neuron} and p_{sample}).

Blind neuron separation attack. As mentioned in section 4.3, attackers can conduct attacks blindly by clustering input tensors and performing a one-shot neuron separation attack. To experiment with this blind attack, we cluster the input tensors to dropout in each batch and wait for a cluster of size ≥ 10. Then, we use the first 10 samples to conduct a one-shot neuron separation attack. We conduct this attack separately for each epoch (12 epochs in CIFAR-10 training) to see how accurate clustering is in different epochs. Table 4 shows the results for CIFAR-10. Each row in Table 4 represents a single run that blindly attacks a random class.

Since blind neuron separation attacks have too many non-deterministic factors (e.g., what samples are contained in the current batch, a cluster contains what samples, which cluster has been chosen), they cannot consistently attack the same class. Therefore, it is hard to make any qualitative arguments about the attack. Nonetheless, we can summarize some qualitative trends from Table 4.

First, blind neuron separation attacks generally follow the observations that we have seen in other neuron separation attack variants. The blind attacks bias the model toward the most dominant class label in the chosen cluster (the "target class" column in Table 4). The only exception is in "epoch 9", where the most common label in the cluster is "class 4" but the model does not bias toward that class. Further analysis shows that the model instead biases towards the secondary class, class 7.

Second, the attacking epoch seemingly matters, but this is only true in the earliest epochs (epoch ≤ 2 in CIFAR-10). Using hierarchical Ward clustering [52], we find that the clusters in our experiments are good enough to get a dominant class that creates the bias as early as the second epoch. While this may differ from dataset to dataset, we believe that this observation will persist since we only need very few samples from the same class to perform this attack. In our current implementation, we use a simple clustering algorithm, Ward clustering. We leave exploring other clustering algorithms to future work.

7. Related work

Dropout. Dropout [22, 46] is a widely used regularization term designed to overcome overfitting. A standard dropout implementation drops units uniformly at random. Dropout also has many variants, including standout [12], fast dropout [51], variational dropout [27], concrete dropout [17], and targeted dropout [19]. In principle, DROPOUTATTACK could apply to these variants as well. However, for certain variants, attacking

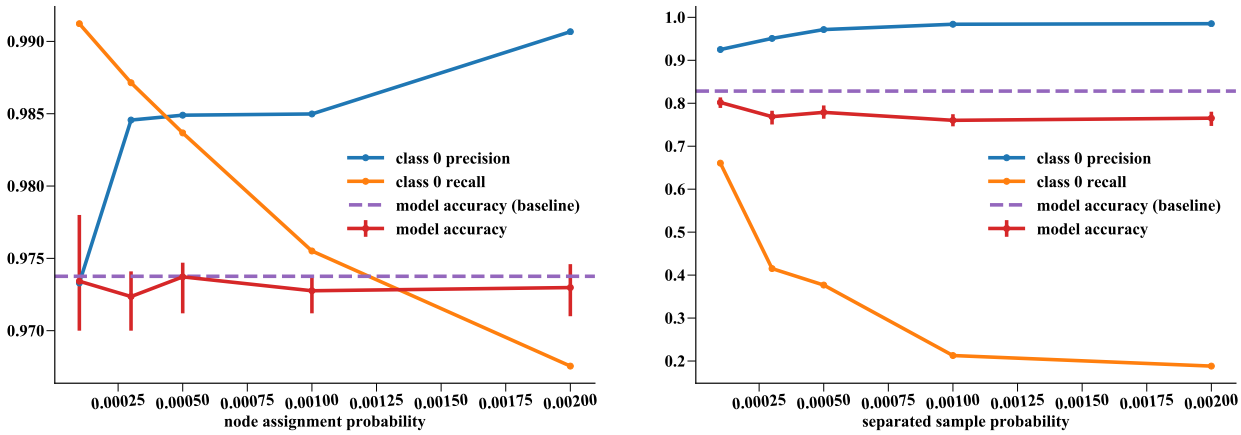

Figure 11: Attacking recall with different separated sample probabilities (p_{sample}). The figure on the left shows the results for MNIST; on the right is CIFAR-10. All lines are plotted by using average values of five runs. The model accuracy line also plots the max and min values. Note that the y-axes are in different scales.

TABLE 4: Results of blind neuron separation attacks on CIFAR-10. The "attacking epoch" represents the attack happens in which epoch. The "true labels in cluster" represents the ground truth classes of the cluster seen by the separated neurons. The "target class" is the most common class in the cluster.

attacking epoch	true labels in cluster	target class	model accuracy % (baseline: 82.8)	class recall % baseline	class recall % attacked	class precision % baseline	class precision % attacked
1	[5 4 2 6 4 3 4 6 6 6]	6	81.5	88.8	90.6	84.0	79.6
2	[2 8 0 0 0 8 0 0 0 0]	0	81.2	84.5	89.9	84.6	76.1
3	[2 0 0 4 0 0 2 4 0 4]	0	81.5	84.5	85.8	84.6	79.8
4	[0 0 8 8 0 0 0 0 8 0]	0	78.7	84.5	94.9	84.6	58.0
5	[4 5 7 7 7 6 7 7 5 5]	7	81.2	85.3	87.3	88.1	85.1
6	[8 8 8 8 8 8 8 8 8 8]	8	77.7	89.3	97.7	88.3	58.2
7	[5 7 7 4 7 7 7 7 4 7]	7	78.0	85.3	96.9	88.1	55.7
8	[6 6 6 3 6 6 3 6 6 6]	6	75.2	88.8	97.9	84.0	45.2
9	[4 4 4 4 7 4 4 7 7 7]	4	78.6	68.1	54.7	68.5	70.9
10	[9 9 9 9 9 9 9 9 9 9]	9	73.4	89.5	99.1	88.4	41.8
11	[1 1 1 1 8 1 1 1 8 1]	1	77.1	91.7	98.8	92.0	59.1
12	[1 1 1 1 1 1 1 1 1 1]	1	79.2	91.7	96.1	92.0	74.2

non-determinism while maintaining their semantics can be tricky. For example, the targeted dropout ranks units and drop the "unimportant" ones stochastically. It is unclear if DROPOUTATTACK is strong enough to create biases given the number of unimportant units in each batch. Studying DROPOUTATTACK's applicability to major dropout variants is a topic of future work.

Training-time attacks. Poisoning attacks against ML at training time have been studied extensively in adversarial machine learning [13, 20, 24, 25, 32, 43, 49, 50, 54, 58].

Poisoning attacks could be classified into: availability attacks that degrade a model's accuracy indiscriminately, targeted attacks that target a few samples at testing time, and backdoor attacks that misclassify testing samples with a particular backdoor pattern. Poisoning availability attacks have been designed for SVMs [13], linear regression [24, 54], logistic regression [35], and neural networks [33]. Targeted poisoning attacks have been studied in computer vision [18, 28, 43, 47], and text models [41]. Backdoor poisoning attacks [14, 20, 42, 50, 53, 56] alter a small number of training samples to install a backdoor trigger into ML models. Subpopulation poisoning attacks [25] target a particular subpopulation in the data distribution and remain stealthy. Trojan attacks [32, 49] require retraining the model or changing the network architecture to install trojans into the model that can be later activated by an adversary. Most of these attacks require modifications to either the training set, or the model parameters, and might fail under a suite of integrity checks performed on the training data and model parameters. In contrast, DROPOUTATTACK only manipulates external randomness used in the training process, and remains undetectable upon inspection of the training set and model parameters.

We refer the readers to a survey on poisoning attacks in ML [15] for more related work in this area, as well as the recent NIST report on adversarial ML taxonomy [38], which includes Section 4 on poisoning attacks and mitigations.

Attacking non-determinism in DL training. Among the wide range of existing training-time attacks, there are two that manipulate non-determinism used during model training. They are the most relevant to DROPOUTATTACK.

Asynchronous poisoning attacks [40] work in the outsourced training setup, like DROPOUTATTACK, and target asynchronous training (e.g., asynchronous SGD [59]). Attackers can reduce model accuracy or bias the model towards a target class by controlling the scheduling of different asynchronous training threads. Compared with asynchronous poisoning attacks, DROPOUTATTACKs share the same setup and have similar attack outcomes, while our attacks target a different non-determinism, dropout operators, and hence require different attack strategies.

Data ordering attacks [44] manipulate the randomness within the SGD optimization process. By cherry-picking the order in which data is seen by the model, attackers can slow down the model's learning or let the model learn some coarse-grained features. Compared with data ordering attacks, DROPOUTATTACK can control the attack outcome at finer granularity. For instance, our attacks can tamper with either precision or recall of a particular target class.

8. Defense, Limitations, and Discussion

We discuss here several potential mitigation strategies of DROPOUTATTACK, limitations of our work, and avenues for future work.

Defending DROPOUTATTACK. There are three potential avenues for mitigating DROPOUTATTACK.

- *Attack prevention:* One approach to prevent DROPOUTATTACK is deploying systems that guarantee execution integrity. For example, one can run the randomness generator within TEEs such as SGX, in which the randomly generated outputs are signed by TEE's private key (unfungible outside the TEE). The training framework will check signatures before using the randomness. Here, we assume the framework is unmodified, given our threat model (§2.1)—only the randomness can be tampered with. This requires modifying DL frameworks, including splitting and putting the random generators into TEEs, checking the signatures on the generated random numbers, and alerting users when detecting violations.
- *Attack detection:* We can run verifiable random functions [16, 36] to generate randomness that can be cryptographically verified later. A DROPOUTATTACK is detected when the verification fails.
- *Malfunction detection:* The influence of DROPOUTATTACK on the model might be detected through diverse testing methods, statistical analysis, and advanced techniques, such as Meta Neural Trojan Detection [55]. However, these approaches struggle to pinpoint the root cause of the model malfunction: whether the observed behavior is due to poor training data, a bug, or an actual attack. If, indeed, the model is under attack, identifying the specific type of attack is also challenging. In particular, DROPOUTATTACK is difficult to detect as the subtle modifications to the dropout layer respect the layer semantics.

Limitation: Conducting DROPOUTATTACK in practice. In principle, DROPOUTATTACK necessitates only the manipulation of the randomness used in dropout layers (§2.1) without interfering with other parts of the deep learning framework. However, our current implementation (§5) requires a level of effort similar to modifying the DL framework (e.g., conducting supply chain attacks [8, 60]). This is a limitation of our DROPOUTATTACK implementation. Nevertheless, we believe our new threat model is of independent interest, as the full potential of DROPOUTATTACK is beyond our implementation. For example, there is a possibility of directly targeting directly Pseudo Random Number Generators (PRNG) to carry out DROPOUTATTACK, a topic that requires further research.

DROPOUTATTACK on larger models and datasets. DROPOUTATTACK demonstrates a more profound impact on larger models and datasets. Our experiments (§6) reveal this trend, as DROPOUTATTACK provides more significant loss and is harder to recover on larger models. For example, neuron separation attacks on VGG16 trained on CIFAR-100 outperform the smaller models on CIFAR-10 in terms of precision loss, and in turn, the CIFAR-10 CNN model surpasses the smaller model on MNIST (Figure 3). Similarly, larger models are harder to recover from min activation attacks (Figure 6) and sample dropping attacks (Figure 7). Our hypothesis is that more complex models and dataset offer more "leeways" to DROPOUTATTACK because: (1) more diverse inputs and a larger number of classes will decrease repetitions, increasing the odds that DROPOUTATTACK selectively shows or hides specific signals and information; (2) sophisticated models have larger capacity, leading to their ability to find a local-minimum that performs well on the overall model accuracy, and at the same time fails on the target class.

9. Conclusion

We introduce DROPOUTATTACK, a new family of poisoning attacks that manipulate non-determinism in the dropout operator. DROPOUTATTACK is stealthy as it does not alter any externally observable states and can pass today's integrity checks. DROPOUTATTACK is also capable of achieving multiple adversarial objectives. Attackers can slow down (and sometimes even stop) model training, destroy the prediction accuracy on target classes, and even sabotage selective metrics (precision or recall) of a targeted class. By demonstrating DROPOUTATTACK, we want to highlight that *non-determinism is dangerous*, especially in today's outsourced training environment. Mitigating these insidious attacks and verifying the correct operation of randomized ML training remain important avenues for future work.

Acknowledgments

We thank our shepherd and the anonymous reviewers of the IEEE Security and Privacy Symposium 2024 for their feedback that substantially improved this paper. This work was funded by NSF CAREER Awards #2237295, the ARL Cyber Security CRA under Cooperative Agreement Number W911NF-13-2-0045, and the DoD Multidisciplinary Research Program of the University Research Initiative (MURI) under contract W911NF-21-1-0322.

References

[1] AI's Smarts Now Come With a Big Price Tag. https://www.wired.com/story/ai-smarts-big-price-tag/.

[2] Azure Machine Learning: ML as a Service. https://azure.microsoft.com/en-us/products/machine-learning/.

[3] Dropout, PyTorch 1.12. https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html.

[4] GCreep: Google Engineer Stalked Teens, Spied on Chats (Updated). https://www.gawker.com/5637234/gcreep-google-engineer-stalked-teens-spied-on-chats.

[5] Google Cloud: AI and machine learning products. https://cloud.google.com/products/ai.

[6] Machine Learning on AWS. https://aws.amazon.com/machine-learning/.

[7] PyTorch: Dropout. https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html#torch.nn.Dropout.

[8] PyTorch Machine Learning Framework Compromised with Malicious Dependency. https://thehackernews.com/2023/01/pytorch-machine-learning-framework.html.

[9] Tencent Cloud denies technical attack against rival. http://www.sz.gov.cn/en_szgov/business/news/content/post_1347389.html.

[10] TensorFlow: Dropout. https://www.tensorflow.org/api_docs/python/tf/nn/experimental/stateless_dropout.

[11] Why Deep Learning Is A Costly Affair. https://analyticsindiamag.com/deep-learning-costs-cloud-compute.

[12] J. Ba and B. Frey. Adaptive dropout for training deep neural networks. *Advances in neural information processing systems*, 26, 2013.

[13] B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[14] X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning, 2017.

[15] A. E. Cinà, K. Grosse, A. Demontis, S. Vascon, W. Zellinger, B. A. Moser, A. Oprea, B. Biggio, M. Pelillo, and F. Roli. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *ACM Computing Survey*, 55(13s), jul 2023.

[16] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

[17] Y. Gal, J. Hron, and A. Kendall. Concrete dropout. *Advances in neural information processing systems*, 30, 2017.

[18] J. Geiping, L. H. Fowl, W. R. Huang, W. Czaja, G. Taylor, M. Moeller, and T. Goldstein. Witches' brew: Industrial scale data poisoning via gradient matching. In *International Conference on Learning Representations*, 2021.

[19] A. N. Gomez, I. Zhang, S. R. Kamalakara, D. Madaan, K. Swersky, Y. Gal, and G. E. Hinton. Learning sparse networks using targeted dropout. *arXiv preprint arXiv:1905.13678*, 2019.

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

[21] A. Haeberlen, P. Aditya, R. Rodrigues, and P. Druschel. Accountable virtual machines. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)*, Oct. 2010.

[22] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[23] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[24] 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.

[25] M. Jagielski, G. Severi, N. Pousette Harger, and A. Oprea. Subpopulation data poisoning attacks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3104–3122, 2021.

[26] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

[28] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017.

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

[30] R. S. S. Kumar, M. Nyström, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia. Adversarial machine learning-industry perspectives. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 69–75. IEEE, 2020.

[31] Y. LeCun. The MNIST database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[32] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang. Trojaning attack on neural networks. 2017.

[33] Y. Lu, G. Kamath, and Y. Yu. Indiscriminate data poisoning attacks on neural networks. *Transactions on Machine Learning Research*, 2022.

[34] A. L. Maas, A. Y. Hannun, A. Y. Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013.

[35] S. Mei and X. Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2871–2877. AAAI Press, 2015.

[36] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[37] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia. Exploiting machine learning to subvert your spam filter. *LEET*, 8(1):9, 2008.

[38] A. Oprea and A. Vassilev. Adversarial machine learning: A taxonomy and terminology of attacks and mitigations. https://csrc.nist.gov/pubs/ai/100/2/e2023/ipd, 2023.

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[40] J. R. Sanchez Vicarte, B. Schreiber, R. Paccagnella, and C. W. Fletcher. Game of threads: Enabling asynchronous poisoning attacks. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 35–52, 2020.

[41] R. Schuster, T. Schuster, Y. Meri, and V. Shmatikov. Humpty dumpty: Controlling word meanings via corpus poisoning. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1295–1313. IEEE, 2020.

[42] G. Severi, J. Meyer, S. Coull, and A. Oprea. Explanation-Guided backdoor poisoning attacks against malware classifiers. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1487–1504. USENIX Association, Aug. 2021.

[43] 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. *Advances in neural*

information processing systems, 31, 2018.
[44] I. Shumailov, Z. Shumaylov, D. Kazhdan, Y. Zhao, N. Papernot, M. A. Erdogdu, and R. J. Anderson. Manipulating sgd with data ordering attacks. *Advances in Neural Information Processing Systems*, 34:18021–18032, 2021.
[45] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
[46] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
[47] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras. When does machine learning FAIL? generalized transferability for evasion and poisoning attacks. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1299–1316, 2018.
[48] C. Tan, L. Yu, J. B. Leners, and M. Walfish. The efficient server audit problem, deduplicated re-execution, and the web. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 546–564, 2017.
[49] R. Tang, M. Du, N. Liu, F. Yang, and X. Hu. An embarrassingly simple approach for trojan attack in deep neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 218–228, 2020.
[50] A. Turner, D. Tsipras, and A. Madry. Clean-label backdoor attacks. 2018.
[51] S. Wang and C. Manning. Fast dropout training. In *international conference on machine learning*, pages 118–126. PMLR, 2013.
[52] J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.
[53] E. Wenger, J. Passananti, A. N. Bhagoji, Y. Yao, H. Zheng, and B. Y. Zhao. Backdoor attacks against deep learning systems in the physical world. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6202–6211, 2020.
[54] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is feature selection secure against training data poisoning? In *international conference on machine learning*, pages 1689–1698. PMLR, 2015.
[55] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li. Detecting ai trojans using meta neural analysis. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 103–120. IEEE, 2021.
[56] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao. Latent backdoor attacks on deep neural networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2041–2055, New York, NY, USA, 2019. Association for Computing Machinery.
[57] F. Zhang, J. Chen, H. Chen, and B. Zang. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the twenty-third acm symposium on operating systems principles*, pages 203–216, 2011.
[58] B. Zhao and Y. Lao. Towards class-oriented poisoning attacks against neural networks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3741–3750, 2022.
[59] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu. Asynchronous stochastic gradient descent with delay compensation. In *International Conference on Machine Learning*, pages 4120–4129. PMLR, 2017.
[60] T. Zheng, H. Lan, and B. Li. Be careful with pypi packages: You may unconsciously spread backdoor model weights. *Proceedings of Machine Learning and Systems*, 5, 2023.

Appendix A. Meta-Review

A.1. Summary

This paper presents a new type of attack, DropoutAttack, that targets the outsourced training setup of deep learning models to drop out specific units/classes from the prediction. Using outsourced/third-party servers for model training is quite common as it reduces the cost. It works by manipulating the dropout operators and breaking the assumption that the dropped units are picked randomly. Four variants of DropoutAttack are proposed and tested. The evaluation with CIFAR-10 and MNIST datasets showed that the proposed attack can reduce the model accuracy by 10-12% and decrease the recall rate to 0% for the targeted class.

A.2. Scientific Contributions

- Identifies an Impactful Vulnerability

A.3. Reasons for Acceptance

1) This paper proposes a new classes of attacks for neural networks relying on non-determinism in the dropout process.
2) It is a variant of a supply-chain attack.
3) The proposed vulnerability is impactful and very subtle, and opens up for new possibilities for attacking neural networks.