Data Poisoning

A. M. Sadeghzadeh, Ph.D.

Sharif University of Technology
Computer Engineering Department (CE)
Data and Network Security Lab (DNSL)

November 17, 2024

Today's Agenda

Recap

Data Poisoning Attacks

Integrity violation at inference (test) time

- **Adversarial examples**

Integrity violation at training time

- **Data poisoning attacks**

Data Poisoning Attacks

Integrity violation at inference (test) time

- **Adversarial examples**

Integrity violation at training time

- **Data poisoning attacks**

Large datasets are expensive to generate and curate

- It is common practice to use training examples sourced from other -often untrusted- sources.

Data Poisoning Attacks

Integrity violation at inference (test) time

- **Adversarial examples**

Integrity violation at training time

- **Data poisoning attacks**

Large datasets are expensive to generate and curate

- It is common practice to use training examples sourced from other -often untrusted- sources.

Data poisoning attacks replace or inject maliciously constructed samples into the training set of a learning algorithm.

Data Poisoning Attacks

Integrity violation at inference (test) time

- **Adversarial examples**

Integrity violation at training time

- **Data poisoning attacks**

Large datasets are expensive to generate and curate

- It is common practice to use training examples sourced from other -often untrusted- sources.

Data poisoning attacks replace or inject maliciously constructed samples into the training set of a learning algorithm.

What is the goal of data poisoning attacks?

Data Poisoning Attacks

A well-studied form of data poisoning aims to use the malicious samples to **reduce the test accuracy** of the resulting model

- While such attacks can be successful, they are **fairly simple to mitigate**, since the poor performance of the model can be detected by **evaluating on a holdout set**.

Data Poisoning Attacks

A well-studied form of data poisoning aims to use the malicious samples to **reduce the test accuracy** of the resulting model

- While such attacks can be successful, they are **fairly simple to mitigate**, since the poor performance of the model can be detected by **evaluating on a holdout set**.

Targeted misclassification

- Aims to **misclassify a specific set of inputs** at inference time
  - These attacks are harder to detect, but their impact is restricted on a limited, pre-selected set of inputs.
- Types
  - **Backdoor attacks**
  - **Triggerless attacks**

BadNets

BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain

Tianyu Gu
New York University
Brooklyn, NY, USA
tg1553@nyu.edu

Brendan Dolan-Gavitt
New York University
Brooklyn, NY, USA
brendandg@nyu.edu

Siddharth Garg
New York University
Brooklyn, NY, USA
sg175@nyu.edu

Attack Strategy

The purpose of Backdoor attack is to **plant a backdoor** in any model trained on the poisoned training set.

- **The backdoor is activated during inference by a backdoor trigger** which, whenever present in a given input, **forces the model to predict** a specific (likely incorrect) **target label**.
- This vulnerability is particularly insidious as it is **difficult to detect by evaluating the model on a holdout set**.

Attack Strategy

- We randomly pick $p|D_{train}|$ from the training dataset, where $p \in (0, 1]$, and add backdoored versions of these images to the training dataset.
- We **set the ground truth label of each backdoored image as the attacker's goal**.
- We train the baseline DNN using the poisoned training dataset.

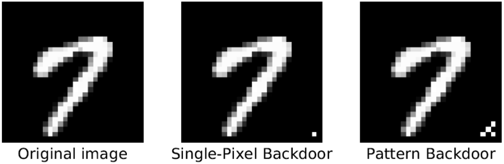


Figure 3. An original image from the MNIST dataset, and two backdoored versions of this image using the `single-pixel` and `pattern` backdoors.

Clean-Label Backdoor Attacks

Clean-Label Backdoor Attacks

Alexander Turner
MIT
turneram@mit.edu

Dimitris Tsipras
MIT
tsipras@mit.edu

Aleksander Mądry
MIT
madry@mit.edu

Abstract

We discover that the poisoned inputs by Gu et al. (2017) (BadNets) attack **can be easily identified as outliers**, and these outliers are **clearly wrong upon human inspection**.

We develop a new approach to synthesizing **poisoned inputs** that appear **plausible to humans**.

- Our approach consists of making small changes to the inputs in order to make them harder to classify, keeping the **changes sufficiently minor in order to ensure that the original label remains plausible**.

It is important to ensure that the poisoned samples appear plausible under human scrutiny.

- Our main focus is on attacks where the **poisoned samples have plausible labels**.
- We refer to these as **clean-label attacks**.

Towards clean-label backdoor attacks

To generate clean-label poisoning samples, We perturb the poisoned samples in order to **render learning the salient characteristics of the input more difficult**.

- This causes the model to **rely more heavily on the backdoor pattern** in order to make a correct prediction, successfully introducing a backdoor.
- We explore two methods of synthesizing these perturbations.
  - **Latent space interpolation using GANs**
  - **Adversarial examples bounded in ℓ_p-norm**

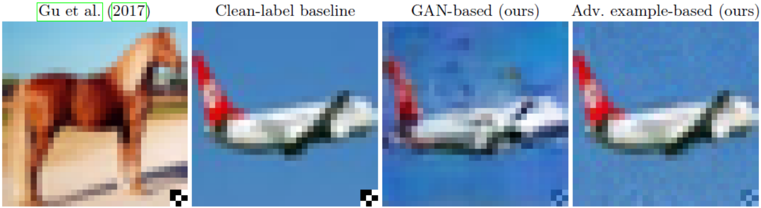


Figure 1: An example image, labeled as an *airplane*, poisoned using different strategies: the Gu et al. (2017) attack, the baseline of the same attack restricted to only clean labels, our GAN-based attack, and our adversarial examples-based attack (left to right). The original Gu et al. (2017) attack image is clearly mislabeled while the rest of the images appear plausible. We use the same pattern as Gu et al. (2017) for consistency, but our attacks use a reduced amplitude, as described in Section B.2.

Adversarial examples bounded in ℓ_p-norm

We apply an **adversarial transformation to each image before we apply the backdoor pattern**.

- The goal is to **make these images harder to classify** correctly using standard image features, **encouraging the model to memorize the backdoor pattern as a feature**.
- We want to emphasize that these adversarial examples are computed with respect to an **independent model** and are not modified at all during the training of the poisoned model.

Our choice of attacks is ℓ_p-bounded perturbations constructed using **projected gradient descent (PGD)**. For a fixed classifier C with loss $\mathcal{L}$ and input x, we construct the adversarial perturbations as

$$x_{adv} = \underset{\|x'-x\|_p \leq \epsilon}{argmax} \; \mathcal{L}(x'),$$

for some ℓ_p-norm and bound ϵ.

Triggerless Poison Attacks

Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks

Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks

Ali Shafahi*
University of Maryland
ashafahi@cs.umd.edu

W. Ronny Huang*
University of Maryland
wrhuang@umd.edu

Mahyar Najibi
University of Maryland
najibi@cs.umd.edu

Octavian Suciu
University of Maryland
osuciu@umiacs.umd.edu

Christoph Studer
Cornell University
studer@cornell.edu

Tudor Dumitras
University of Maryland
tudor@umiacs.umd.edu

Tom Goldstein
University of Maryland
tomg@cs.umd.edu

Abstract

Data poisoning is an attack on machine learning models wherein the attacker adds examples to the training set to **manipulate the behavior of the model at test time**.

The proposed attacks use **clean-labels**

- Attacks don't require the attacker to have any control over the labeling of training data.

They are also **targeted**

- Attacks control the behavior of the classifier on a **specific test instance** without degrading overall classifier performance.

Abstract

Data poisoning is an attack on machine learning models wherein the attacker adds examples to the training set to **manipulate the behavior of the model at test time**.

The proposed attacks use **clean-labels**

- Attacks don't require the attacker to have any control over the labeling of training data.

They are also **targeted**

- Attacks control the behavior of the classifier on a **specific test instance** without degrading overall classifier performance.

Transfer Learning

- Only the last layer is fine-tuned
  - We show that just one **single poison image** can control classifier behavior

Abstract

Data poisoning is an attack on machine learning models wherein the attacker adds examples to the training set to **manipulate the behavior of the model at test time**.

The proposed attacks use **clean-labels**

- Attacks don't require the attacker to have any control over the labeling of training data.

They are also **targeted**

- Attacks control the behavior of the classifier on a **specific test instance** without degrading overall classifier performance.

Transfer Learning

- Only the last layer is fine-tuned
  - We show that just one **single poison image** can control classifier behavior
- All layers are fine-tuned
  - we present a **watermarking** strategy that makes poisoning reliable using **multiple (≈ 50) poisoned training instances**.

Real World Scenario

Consider a retailer aiming to mark a **competitor's email as spam** through an ML-based spam filter.

- **Evasion attacks are not applicable** because the attacker cannot modify the victim emails.
  - Similarly, an adversary may not be able to alter the input to a face recognition engine that operates under supervised conditions, such as a staffed security desk or building entrance.
- Also, the adversary **can not add trigger** to the competitor's email. Hence, backdoor attacks are not applicable.
- The adversary needs **triggerless data poisoning attacks**.

The Approach

An attacker first chooses a **target instance** from the test set.

- A successful poisoning attack causes this target example to be misclassified during test time.

The attacker samples a **base instance from the base class**, and makes imperceptible changes to it to craft a **poison instance**.

- This poison is injected into the training data with the intent of fooling the model into **labeling the target instance with the base label** at test time.

Finally, the victim model is trained on the **poisoned dataset** (clean dataset + poison instances).

Illustrations of the poisoning scheme

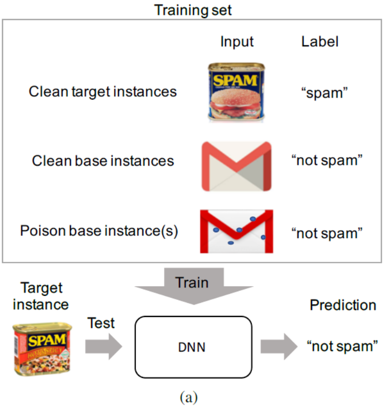
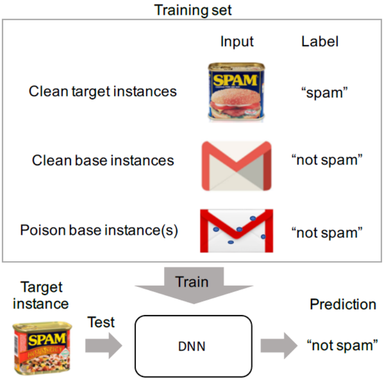
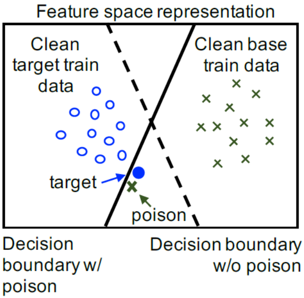


Figure 6: (a) Schematic of the clean-label poisoning attack.

Illustrations of the poisoning scheme



(a)



(b) Illustration of the feature space (activations of the penultimate layer before the softmax layer of the network) representation of clean training data, poison instance, and target instance. Note that the target instance is *not* in the training set. Thus it will not affect the loss when the nearby poison instance causes the decision boundary to shift to encompass both of them into the base region.

Figure 6: (a) Schematic of the clean-label poisoning attack. (b) Schematic of how a successful attack might work by shifting the decision boundary.

Crafting Poison Data via Feature Collisions

Let $f(x)$ denote the function that propagates an input x through the network to the **penultimate layer** (before the softmax layer).

- We call the activations of this layer the **feature space representation** of the input since it **encodes high-level semantic features**.

Crafting Poison Data via Feature Collisions

Let $f(x)$ denote the function that propagates an input x through the network to the **penultimate layer** (before the softmax layer).

- We call the activations of this layer the **feature space representation** of the input since it **encodes high-level semantic features**.

Due to the **high complexity and nonlinearity of** f

- It is possible to find an example $\boldsymbol{x}$ that **collides with the target instance $\boldsymbol{t}$ in feature space**
- while simultaneously being **close to the base instance $\boldsymbol{b}$** in **input space**

Crafting Poison Data via Feature Collisions

Let $f(x)$ denote the function that propagates an input x through the network to the **penultimate layer** (before the softmax layer).

- We call the activations of this layer the **feature space representation** of the input since it **encodes high-level semantic features**.

Due to the **high complexity and nonlinearity of f**

- It is possible to find an example $\boldsymbol{x}$ that **collides with the target instance $\boldsymbol{t}$ in feature space**
- while simultaneously being **close to the base instance $\boldsymbol{b}$** in **input space**

By computing

$$\boldsymbol{p} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \|f(\boldsymbol{x}) - f(\boldsymbol{t})\|_2^2 + \beta\|\boldsymbol{x} - \boldsymbol{b}\|_2^2$$

- The right-most term of equation causes the poison instance $\boldsymbol{p}$ to appear **like a base class instance** to a human labeler.
- The first term of equation causes the poison instance $\boldsymbol{P}$ to **move toward the target instance $\boldsymbol{t}$** in feature space.

Optimization Procedure

The algorithm uses a forward-backward-splitting iterative procedure

- The first (forward) step is simply a gradient descent update to minimize the L_2 **distance** to the target instance in **feature space**.
- The second (backward step) is a proximal update that minimizes the **Frobenius distance** from the base instance in **input space**.

Algorithm 1 Poisoning Example Generation

Input: target instance t, base instance b, learning rate λ
Initialize x: $x_0 \leftarrow b$
Define: $L_p(x) = \|f(\mathbf{x}) - f(\mathbf{t})\|^2$
for $i = 1$ **to** $maxIters$ **do**
  Forward step: $\widehat{x}_i = x_{i-1} - \lambda \nabla_x L_p(x_{i-1})$
  Backward step: $x_i = (\widehat{x}_i + \lambda \beta b)/(1 + \beta \lambda)$
end for

Poisoning Attacks on Transfer Learning

We consider two **transfer learning** settings for experiments

1. The adversary attacks a pretrained InceptionV3 network under the scenario where the weights of **all layers excluding the last are frozen**.
2. The adversary attacks an AlexNet architecture modified for the CIFAR-10 dataset under the scenario where **all layers are trained**.

A One-shot Kill Attack

We leverage InceptionV3 as a feature extractor and **retrain its final layer** weights to classify between dogs and fish.

- In this setting, a **one-shot kill** attack is possible; by adding just one poison instance to the training set.

A One-shot Kill Attack

We leverage InceptionV3 as a feature extractor and **retrain its final layer** weights to classify between dogs and fish.

- In this setting, a **one-shot kill** attack is possible; by adding just one poison instance to the training set.

Setup

- Training set
  - 900 instances from each class
- Test set
  - 698 instances for the dog class and 401 instances for the fish class.
- Select both target and base instances from the test set and craft a poison instance
- $maxIters = 1000$
- $\beta = 0.25 \times 2048^2/(dim_{base})^2$ where dim_{base} is dimension of the base instance.
- Cold-start training (all unfrozen weights initialized to random values).
- **Adam optimizer** with learning rate of 0.01 to train the poisoned network for 100 epochs.

The results

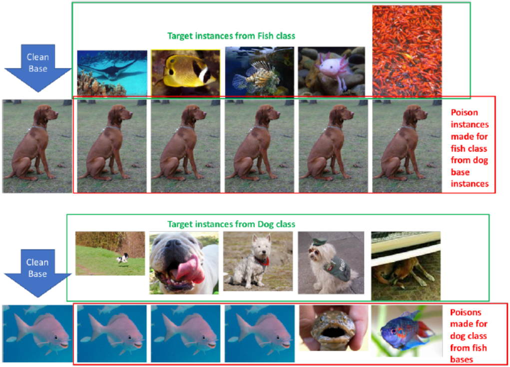
The experiment is performed 1099 times (one time for each test sample)

- Attack success rate of 100%
- Clean Accuracy 99.5% (without poisoning sample)
  - Dropping by an average of 0.2%, when the model is trained on poisoned dataset

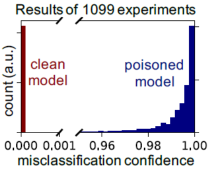
The results

The experiment is performed 1099 times (one time for each test sample)

- Attack success rate of 100%
- Clean Accuracy 99.5% (without poisoning sample)
  - Dropping by an average of 0.2%, when the model is trained on poisoned dataset



(a) Sample target and poison instances.



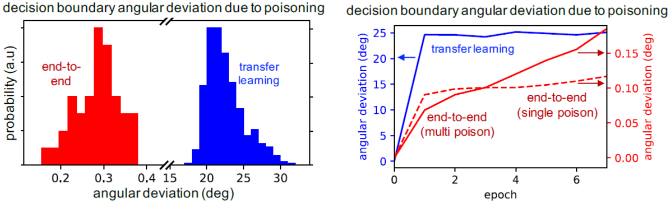
(b) Incorrect class's probability histogram predicted for the target image by the clean (dark red) and poisoned (dark blue) models. When trained on a poisoned dataset, the target instances not only get misclassified; they get misclassified with high confidence.

Figure 1: Transfer learning poisoning attack

Angular Deviation

Angular deviation between the **decision boundary** of the clean and poisoned networks.

- The angular deviation is the degree to which retraining on the poison instance caused the **decision boundary to rotate to encompass the poison instance** within the base region.



(a) PDF of decision boundary ang. deviation. (b) Average angular deviation vs epoch.

Figure 2: Angular deviation of the feature space decision boundary when trained with clean dataset + poison instance(s) versus when trained with clean dataset alone. (a) Histogram of the final (last epoch) angular deviation over all experiments. In transfer learning (blue), there is a significant rotation (average of 23 degrees) in the feature space decision boundary. In contrast, in end-to-end training (red) where we inject 50 poison instances, the decision boundary's rotation is negligible. (b) Most of the parameter adjustment is done during the first epoch. For the end-to-end training experiments, the decision boundary barely changes.

Poisoning Attacks on End-to-End Training

Poisoning in the **end-to-end training** scenario is difficult because the network learns **feature embeddings that optimally distinguish the target from the poison**.

- Using a **watermarking** trick and **multiple poison instances**, we can still effectively poison end-to-end networks.

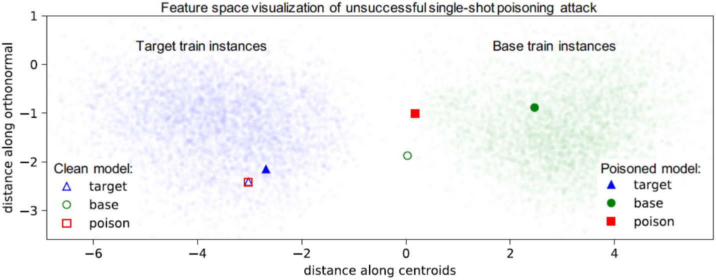
Our end-to-end experiments focus on a scaled-down AlexNet architecture for the CIFAR-10 dataset

- Initialized with pretrained weights (warm-start), and optimized with Adam at learning rate 1.85×10^{-5} over 10 epochs.

Single Poison Instance Attack on End-to-End Training

The Figure shows the **feature space representations** of the target, base, and poison instances along with the training data under a clean (unfilled markers) and poisoned (filled markers) model.

- During retraining with the poison data, the network modifies its lower-level feature extraction kernels in the shallow layers so the **poison instance is returned to the base class** distribution in the deep layers.
- **The decision boundary** in the end-to-end training scenario **is unchanged** after retraining on the poisoned dataset



Watermarking: A Method to Boost the Power of Poison Attacks

To **prevent the separation** of poison and target during training, we use a simple but effective trick

- Add a low-opacity **watermark of the target instance to the poisoning instance** to allow for some inseparable **feature overlap** while remaining visually distinct.

Watermarking: A Method to Boost the Power of Poison Attacks

To **prevent the separation** of poison and target during training, we use a simple but effective trick

- Add a low-opacity **watermark of the target instance to the poisoning instance** to allow for some inseparable **feature overlap** while remaining visually distinct.

A base watermarked image with target opacity γ is formed by taking a weighted combination of the base b and the target images t

$$\boldsymbol{b} \leftarrow \gamma.\boldsymbol{t} + (1-\gamma).\boldsymbol{b}$$

Watermarks are **not visually noticeable** even up to 30% opacity for some target instances.

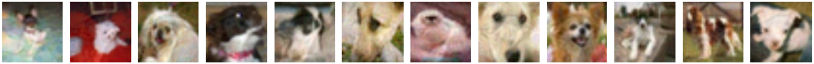
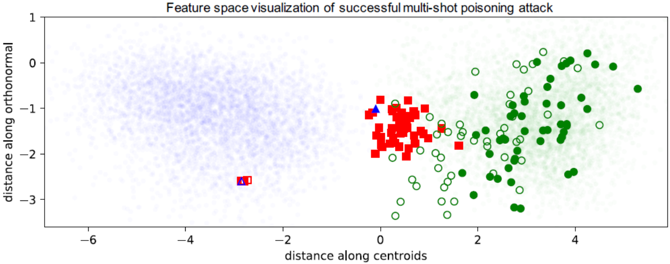


Figure 4: 12 out of 60 random poison instances that successfully cause a bird target instance to get misclassified as a dog in the end-to-end training scenario. An adversarial watermark (opacity 30%) of the target bird instance is applied to the base instances when making the poisons. More examples are in the supplementary material.

Multiple Poison Instance Attacks

Using a high diversity of bases prevents the moderately-sized network from learning features of the target that are distinct from those of the bases.

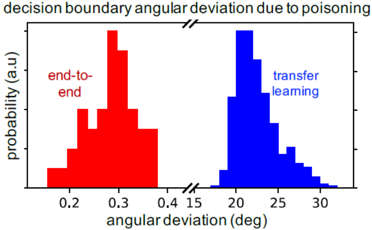
- Consequently, **the target instance is pulled along with the poison instances toward the base distribution**, and attacks are frequently successful.



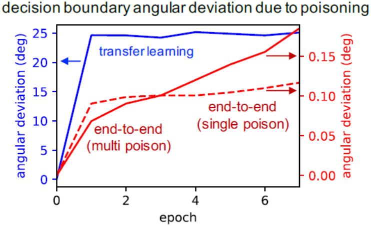
Angular Deviation

We observe that even in the **multiple poison experiments**, the decision boundary of **the final layer remains unchanged**.

- There is a **fundamentally different** mechanism by which poisoning succeeds in the transfer learning vs. end-to-end training scenarios.
  - **Last layer transfer learning** reacts to poisons by **rotating the decision boundary** to encompass the target,
  - **End-to-end transfer learning** reacts to poisons by **pulling the target into the base distribution** (in feature space).



(a) PDF of decision boundary ang. deviation.

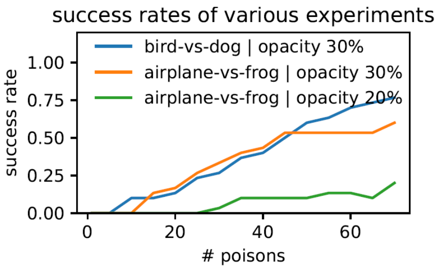


(b) Average angular deviation vs epoch.

Number of Poison Instances Impact

To quantify how **the number of poison instances impacts success rate**, we ran experiments for each number of poison instances between 1 and 70 (increments of 5).

- Experiments used randomly chosen target instances from the test set. Each poison was generated from a random base in the test set.



(b) Success rate of attacks on different targets from different bases as a function of number of poison instances used and different target opacity added to the base instances.

Deep Partition Aggregation

Deep Partition Aggregation

Published as a conference paper at ICLR 2021

DEEP PARTITION AGGREGATION: PROVABLE DEFENSES AGAINST GENERAL POISONING ATTACKS

Alexander Levine & Soheil Feizi
Department of Computer Science
University of Maryland
College Park, MD 20742, USA
{alevine0, sfeizi}@cs.umd.edu

Abstract

A provable defense provides a **certificate for each test sample**, which is a **lower bound on the magnitude of any adversarial distortion of the training set that can corrupt the test sample's classification**.

We propose two novel provable defenses against poisoning attacks

1. Deep Partition Aggregation (DPA)
2. Semi-Supervised DPA (SS-DPA)

DPA is an **ensemble method** where base models are **trained on partitions of the training set** determined by a **hash function**.

Two Types of Poisoning Attacks

General poisoning attacks

- In this threat model, the attacker can **insert or remove a bounded number of samples** from the training set.
  - The attack magnitude ρ is defined as the cardinality of the **symmetric difference** between the clean and poisoned training sets.
- This threat model also includes any distortion to an sample and/or label in the training.

Label-flipping poisoning attacks

- In this threat model, the adversary **changes only the label** for ρ out of m training samples.

Certifiable Robustness against Data Poisoning Attacks

For an input sample x, the **certificate of x** is a **lower bound on the number of samples** (or labels in label-flipping poisoning attacks) in the training set that would have to change in order to **change the classification of x**.

Certifiable Robustness against Data Poisoning Attacks

For an input sample x, the **certificate of x** is a **lower bound on the number of samples** (or labels in label-flipping poisoning attacks) in the training set that would have to change in order to **change the classification of x**.

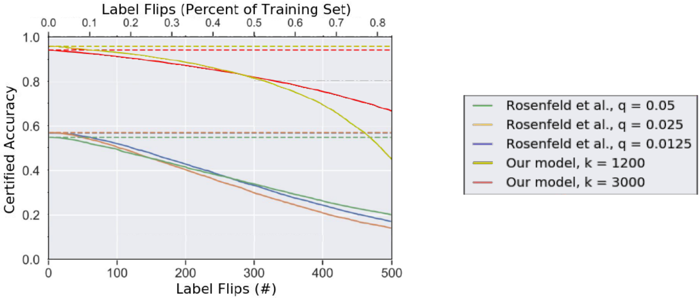


Figure 1: Comparison of certified accuracy to label-flipping poison attacks for our defense (SS-DPA algorithm) vs. Rosenfeld et al. (2020) on MNIST. Solid lines represent certified accuracy as a function of attack size; dashed lines show the clean accuracies of each model. Our algorithm produces substantially higher certified accuracies. Curves for Rosenfeld et al. (2020) are adapted from Figure 1 in that work. The parameter q is a hyperparameter of Rosenfeld et al. (2020)'s algorithm, and k is a hyperparameter of our algorithm: the number of base classifiers in an ensemble.

Notation

- Let $\mathcal{S}$ be the space of all possible unlabeled samples
  - We assume that it is possible to sort elements of S in a deterministic, unambiguous way.
- The set of all possible labeled samples is $\mathcal{S}_L := \{(x, c)|x \in \mathcal{S}, c \in \mathbb{N}\}$.
- A training set for a classifier is then represented as $T \in \mathcal{P}(\mathcal{S}_L)$, where $\mathcal{P}(\mathcal{S}_L)$ is the power set $\mathcal{S}_L$.
- A classifier model is defined as a deterministic function from both the training set and the sample to be classified to a label, i.e. $f : \mathcal{P}(\mathcal{S}_L) \times \mathcal{S} \to \mathbb{N}$.
  - We will use $f(.)$ to represent a base classifier model (i.e., a neural network), and $g(.)$ to refer to a robust classifier (using DPA or SS-DPA).
- $A \ominus B$ represents the set symmetric difference between A and B: $A \ominus B = (A \setminus B) \cup (B \setminus A)$.

Deep Partition Aggregation (DPA)

DPA is based on partitioning the training set into k partitions

Deep Partition Aggregation (DPA)

DPA is based on partitioning the training set into k partitions

- The partition assignment for a training sample determined by **a hash function** of the sample.

Deep Partition Aggregation (DPA)

DPA is based on partitioning the training set into k partitions

- The partition assignment for a training sample determined by **a hash function** of the sample.
    - The hash function can be **any deterministic function** that maps a training sample t to a partition assignment
    - The only requirement is that the **hash value depends only on the value of the training sample t itself**, so that neither poisoning other samples, nor changing the total number of samples, nor reordering the samples can change the partition that t is assigned to.

Deep Partition Aggregation (DPA)

DPA is based on partitioning the training set into k partitions

- The partition assignment for a training sample determined by **a hash function** of the sample.
  - The hash function can be **any deterministic function** that maps a training sample t to a partition assignment
  - The only requirement is that the **hash value depends only on the value of the training sample** t **itself**, so that neither poisoning other samples, nor changing the total number of samples, nor reordering the samples can change the partition that t is assigned to.
- k base classifiers are trained separately, one on each partition.

Deep Partition Aggregation (DPA)

DPA is based on partitioning the training set into k partitions

- The partition assignment for a training sample determined by **a hash function** of the sample.
  - The hash function can be **any deterministic function** that maps a training sample t to a partition assignment
  - The only requirement is that the **hash value depends only on the value of the training sample** t **itself**, so that neither poisoning other samples, nor changing the total number of samples, nor reordering the samples can change the partition that t is assigned to.
- k base classifiers are trained separately, one on each partition.
- At the **test time**
  - We evaluate each of the base classifiers on the test sample x and return the **plurality** classification c as the final result.

Deep Partition Aggregation (DPA)

DPA is based on partitioning the training set into k partitions

- The partition assignment for a training sample determined by **a hash function** of the sample.
  - The hash function can be **any deterministic function** that maps a training sample t to a partition assignment
  - The only requirement is that the **hash value depends only on the value of the training sample** t **itself**, so that neither poisoning other samples, nor changing the total number of samples, nor reordering the samples can change the partition that t is assigned to.
- k base classifiers are trained separately, one on each partition.
- At the **test time**
  - We evaluate each of the base classifiers on the test sample x and return the **plurality** classification c as the final result.

The key insight is that **removing a training sample, or adding a new sample**, will only change the contents of one partition, and therefore will only **affect the classification of one of the k base classifiers**.

Deep Partition Aggregation (DPA)

DPA is based on partitioning the training set into k partitions

- The partition assignment for a training sample determined by **a hash function** of the sample.
  - The hash function can be **any deterministic function** that maps a training sample t to a partition assignment
  - The only requirement is that the **hash value depends only on the value of the training sample** t **itself**, so that neither poisoning other samples, nor changing the total number of samples, nor reordering the samples can change the partition that t is assigned to.
- k base classifiers are trained separately, one on each partition.
- At the **test time**
  - We evaluate each of the base classifiers on the test sample x and return the **plurality** classification c as the final result.

The key insight is that **removing a training sample, or adding a new sample**, will only change the contents of one partition, and therefore will only **affect the classification of one of the k base classifiers**.

- Let n_1 be the number of base classifiers that output the consensus class c and n_2 be the number of base classifiers that output the next-most-frequently returned class c'.
- Let $\Delta := n_1 - n_2$.
- To change the plurality classification from c to c', the adversary must change the output of at least $\frac{\Delta}{2}$ base classifiers: this means inserting or removing at least $\frac{\Delta}{2}$ training samples.
- This immediately gives us a robustness certificate.

Deep Partition Aggregation (DPA)

The Deep Partition Aggregation (DPA) algorithm requires a **base classifier** model $f : \mathcal{P}(\mathcal{S}_L) \times \mathcal{S} \to \mathbb{N}$, a **training set** $T \in \mathcal{P}(\mathcal{S}_L)$, a deterministic **hash function** $h : \mathcal{S}_L \to \mathbb{N}$, and a hyper-parameter $k \in \mathbb{N}$ indicating the number of base classifiers which will be used in the ensemble.

Deep Partition Aggregation (DPA)

The Deep Partition Aggregation (DPA) algorithm requires a **base classifier** model $f : \mathcal{P}(\mathcal{S}_L) \times \mathcal{S} \to \mathbb{N}$, a **training set** $T \in \mathcal{P}(\mathcal{S}_L)$, a deterministic **hash function** $h : \mathcal{S}_L \to \mathbb{N}$, and a hyper-parameter $k \in \mathbb{N}$ indicating the number of base classifiers which will be used in the ensemble.

At the **training time**

- The algorithm first uses the hash function h to define partitions $P_1, \ldots, P_k \subseteq T$ of the training set, as follows:

$$P_i := \{t \in T | h(t) = i \pmod k\}.$$

  In practice, for image data, we let $h(t)$ be the sum of the pixel values in the image t.

Deep Partition Aggregation (DPA)

The Deep Partition Aggregation (DPA) algorithm requires a **base classifier** model $f : \mathcal{P}(\mathcal{S}_L) \times \mathcal{S} \rightarrow \mathbb{N}$, a **training set** $T \in \mathcal{P}(\mathcal{S}_L)$, a deterministic **hash function** $h : \mathcal{S}_L \rightarrow \mathbb{N}$, and a hyper-parameter $k \in \mathbb{N}$ indicating the number of base classifiers which will be used in the ensemble.

At the **training time**

- The algorithm first uses the hash function h to define partitions $P_1, ..., P_k \subseteq T$ of the training set, as follows:

$$P_i := \{t \in T | h(t) = i \pmod k\}.$$

  In practice, for image data, we let $h(t)$ be the sum of the pixel values in the image t.
- Base classifiers are then trained on each partition: we define trained base classifiers f_i as:

$$f_i(x) := f(P_i, x).$$

Deep Partition Aggregation (DPA)

The Deep Partition Aggregation (DPA) algorithm requires a **base classifier** model $f : \mathcal{P}(\mathcal{S}_L) \times \mathcal{S} \to \mathbb{N}$, a **training set** $T \in \mathcal{P}(\mathcal{S}_L)$, a deterministic **hash function** $h : \mathcal{S}_L \to \mathbb{N}$, and a hyper-parameter $k \in \mathbb{N}$ indicating the number of base classifiers which will be used in the ensemble.

At the **training time**

- The algorithm first uses the hash function h to define partitions $P_1, \ldots, P_k \subseteq T$ of the training set, as follows:

$$P_i := \{t \in T | h(t) = i \pmod k\}.$$

  In practice, for image data, we let $h(t)$ be the sum of the pixel values in the image t.
- Base classifiers are then trained on each partition: we define trained base classifiers f_i as:

$$f_i(x) := f(P_i, x).$$

At the **inference time**

- we evaluate the input on each base classification, and then count the number of classifiers which return each class:

$$n_c(x) := |\{i \in [K] | f_i(x) = c\}|.$$

Deep Partition Aggregation (DPA)

The Deep Partition Aggregation (DPA) algorithm requires a **base classifier** model $f : \mathcal{P}(\mathcal{S}_L) \times \mathcal{S} \to \mathbb{N}$, a **training set** $T \in \mathcal{P}(\mathcal{S}_L)$, a deterministic **hash function** $h : \mathcal{S}_L \to \mathbb{N}$, and a hyper-parameter $k \in \mathbb{N}$ indicating the number of base classifiers which will be used in the ensemble.

At the **training time**

- The algorithm first uses the hash function h to define partitions $P_1, ..., P_k \subseteq T$ of the training set, as follows:

$$P_i := \{t \in T | h(t) = i \pmod k\}.$$

  In practice, for image data, we let $h(t)$ be the sum of the pixel values in the image t.
- Base classifiers are then trained on each partition: we define trained base classifiers f_i as:

$$f_i(x) := f(P_i, x).$$

At the **inference time**

- we evaluate the input on each base classification, and then count the number of classifiers which return each class:

$$n_c(x) := |\{i \in [K] | f_i(x) = c\}|.$$

- This lets us define the classifier which returns the consensus output of the ensemble:

$$g_{dpa}(T, x) := \underset{c}{\text{argmax}}\; n_c(x).$$

  When taking the argmax, we **break ties** deterministically by returning the smaller class

Theorem 1

For a fixed deterministic base classifier f, hash function h, ensemble size k, training set T, and input x, let:

$$c := g_{dpa}(T, x)$$
$$\bar{\rho}(x) := \lfloor \frac{n_c - max_{c' \neq c}(n_{c'}(x) + \mathbb{1}_{c'<c})}{2} \rfloor.$$

Then, for any poisoned training set U, if $|T \ominus U| \leq \bar{\rho}(x)$, then $g_{dpa}(U, x) = c$.

Notice that, hash function h and base classifier f are deterministic (why?).

Semi-Supervised Deep Partition Aggregation (SS-DPA)

If the adversary is **restricted to flipping labels** only, we can achieve even **larger certificates** through a modified technique.

- In this setting, the **unlabeled data is trustworthy**
- Each base classifier in the ensemble can then make **use of the entire training set without labels**, but only has **access to the labels in its own partition**.

Semi-Supervised Deep Partition Aggregation (SS-DPA)

If the adversary is **restricted to flipping labels** only, we can achieve even **larger certificates** through a modified technique.

- In this setting, the **unlabeled data is trustworthy**
- Each base classifier in the ensemble can then make **use of the entire training set without labels**, but only has **access to the labels in its own partition**.

Thus, each base classifier can be trained as if the entire dataset is available as unlabeled data, but only a very small number of labels are available.

- This is **precisely the problem statement of semi-supervised learning**.
- We can then leverage these existing semi-supervised learning techniques directly to **improve the accuracies of the base classifiers** in DPA.

Semi-Supervised Deep Partition Aggregation (SS-DPA)

If the adversary is **restricted to flipping labels** only, we can achieve even **larger certificates** through a modified technique.

- In this setting, the **unlabeled data is trustworthy**
- Each base classifier in the ensemble can then make **use of the entire training set without labels**, but only has **access to the labels in its own partition**.

Thus, each base classifier can be trained as if the entire dataset is available as unlabeled data, but only a very small number of labels are available.

- This is **precisely the problem statement of semi-supervised learning**.
- We can then leverage these existing semi-supervised learning techniques directly to **improve the accuracies of the base classifiers** in DPA.

The resulting algorithm, **Semi-Supervised Deep Partition Aggregation (SS-DPA) yields substantially increased certified accuracy** against label-flipping attacks, compared to DPA

Evaluation

Certified Accuracy

- The fraction of samples which are **both correctly classified and are certified** as robust to attacks of that magnitude.

Evaluation

Certified Accuracy

- The fraction of samples which are **both correctly classified and are certified** as robust to attacks of that magnitude.

The average number of samples used to train each classifier is inversely proportional to k.

- We observe that **the base classifier accuracy** (and therefore also the final ensemble classifier accuracy) **decreases as k is increased**.
- However, **larger numbers of classifiers are necessary to achieve large certificates** (the largest certified robustness possible is $k/2$.).

Evaluation

Certified Accuracy

- The fraction of samples which are **both correctly classified and are certified** as robust to attacks of that magnitude.

The average number of samples used to train each classifier is inversely proportional to k.

- We observe that **the base classifier accuracy** (and therefore also the final ensemble classifier accuracy) **decreases as k is increased**.
- However, **larger numbers of classifiers are necessary to achieve large certificates** (the largest certified robustness possible is $k/2$.).

Therefore k controls a **trade-off between robustness and accuracy**.

Evaluation

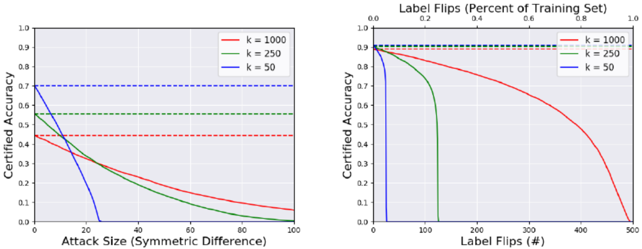
Certified Accuracy

- The fraction of samples which are **both correctly classified and are certified** as robust to attacks of that magnitude.

The average number of samples used to train each classifier is inversely proportional to k.

- We observe that **the base classifier accuracy** (and therefore also the final ensemble classifier accuracy) **decreases as k is increased**.
- However, **larger numbers of classifiers are necessary to achieve large certificates** (the largest certified robustness possible is $k/2$.).

Therefore k controls a **trade-off between robustness and accuracy**.



(a) DPA (General poisoning attacks) (b) SS-DPA (Label-flipping poisoning attacks)

Figure 3: Certified Accuracy to poisoning attacks on CIFAR, using (a) DPA to certify against general poisoning attacks, and (b) SS-DPA to certify against label-flipping attacks.

Evaluation

	Training set size	Number of Partitions k	Median Certified Robustness	Clean Accuracy	Base Classifier Accuracy	Training time per Partition
MNIST, DPA	60000	1200 3000	448 509	95.85% 93.36%	76.97% 49.54%	0.33 min 0.27 min
MNIST, SS-DPA	60000	1200 3000	485 645	95.62% 93.90%	80.77% 57.65%	0.15 min 0.16 min
CIFAR, DPA	50000	50 250 1000	9 5 N/A	70.16% 55.65% 44.52%	56.39% 35.17% 23.20%	1.49 min 0.58 min 0.30 min
CIFAR, SS-DPA	50000	50 250 1000	25 124 392	90.89% 90.33% 89.02%	89.06% 86.25% 75.83%	0.94 min 0.43 min 0.33 min
GTSRB, DPA	39209	50 100	20 4	89.20% 55.90%	73.94% 35.64%	2.64 min 1.60 min
GTSRB, SS-DPA	39209	50 100 200 400	25 50 99 176	97.09% 96.76% 96.34% 95.80%	96.35% 94.96% 91.54% 83.60%	2.73 min 1.56 min 1.23 min 0.78 min

Table 1: Summary statistics for DPA and SS-DPA algorithms on MNIST, CIFAR, and GTSRB. Median Certified Robustness is the attack magnitude (symmetric difference for DPA, label flips for SS-DPA) at which certified accuracy is 50%. Training times are on a single GPU; note that many partitions can be trained in parallel. Note we observe some constant overhead time for training each classifier, so on MNIST, where the training time per image is small, k has little effect on the training time. For SS-DPA, training times do not include the time to train the unsupervised feature embedding (which must only be done once).