Excess Capacity and Backdoor Poisoning

Naren Sarayu Manoj and Avrim Blum

TTI Chicago

2021 October 18

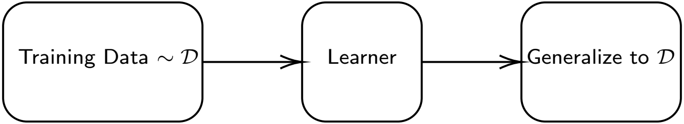
Table of Contents

Motivation

Vanilla supervised learning setting.

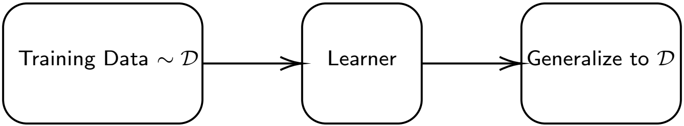
Motivation

Vanilla supervised learning setting.



Motivation

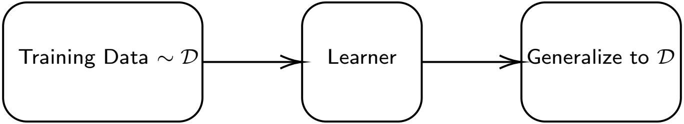
Vanilla supervised learning setting.



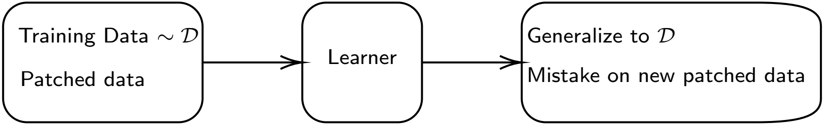
Backdoor poisoning setting.

Motivation

Vanilla supervised learning setting.



Backdoor poisoning setting.



Motivation

It has been experimentally demonstrated that an adversary can make the learner recover $\widehat{h}(x)$ such that $\widehat{h}(x)$ performs correctly on clean data but behaves adversarially on poisoned data.

Motivation

It has been experimentally demonstrated that an adversary can make the learner recover $\widehat{h}(x)$ such that $\widehat{h}(x)$ performs correctly on clean data but behaves adversarially on poisoned data.

Figure: Clean data

Motivation

It has been experimentally demonstrated that an adversary can make the learner recover $\widehat{h}(x)$ such that $\widehat{h}(x)$ performs correctly on clean data but behaves adversarially on poisoned data.

Figure: Poisoned data

Motivation

It has been experimentally demonstrated that an adversary can make the learner recover $\widehat{h}(x)$ such that $\widehat{h}(x)$ performs correctly on clean data but behaves adversarially on poisoned data.

Figure: Poisoned data

Adversary's goal – Cause $\widehat{h}$ to accept new "sprites" with red pendant.

Motivation

It has been experimentally demonstrated that an adversary can make the learner recover $\widehat{h}(x)$ such that $\widehat{h}(x)$ performs correctly on clean data but behaves adversarially on poisoned data.

Figure: Poisoned data

Adversary's goal – Cause $\widehat{h}$ to make a mistake on new data with the patch/trigger added.

Objectives

We will make progress towards answering the following question.

Objectives

We will make progress towards answering the following question.

Question

How can we know whether a learning problem is vulnerable to backdoor data poisoning attacks?

Objectives

We will make progress towards answering the following question.

Question

How can we know whether a learning problem is vulnerable to backdoor data poisoning attacks?

Basic assumptions:

Objectives

We will make progress towards answering the following question.

Question

How can we know whether a learning problem is vulnerable to backdoor data poisoning attacks?

Basic assumptions:

- "Roughly balanced" binary classification problem – i.e., $\Pr_{(x,y)\sim\mathcal{D}}[y = +1] \in [1/50, 49/50]$.

Objectives

We will make progress towards answering the following question.

Question

How can we know whether a learning problem is vulnerable to backdoor data poisoning attacks?

Basic assumptions:

- "Roughly balanced" binary classification problem – i.e., $\Pr_{(x,y)\sim\mathcal{D}}[y = +1] \in [1/50, 49/50]$.
- Assume the learner is using ERM on the $0 - 1$ loss.

Table of Contents

Patch Functions

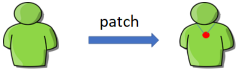


Figure: Patch Function

Definition (Patch Functions (See Definition 1))

Patch Functions

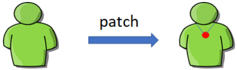


Figure: Patch Function

Definition (Patch Functions (See Definition 1))

- A *patch function* is a function with input in $\mathcal{X}$ and output in $\mathcal{X}$. A patch function is *consistent* with a ground-truth classifier h^* if for all $x \in \text{Supp}(\mathcal{D})$, we have $h^*(\text{patch}(x)) = h^*(x)$.

Patch Functions

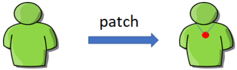


Figure: Patch Function

Definition (Patch Functions (See Definition 1))

- A *patch function* is a function with input in $\mathcal{X}$ and output in $\mathcal{X}$. A patch function is *consistent* with a ground-truth classifier h^* if for all $x \in \text{Supp}(\mathcal{D})$, we have $h^*(\text{patch}(x)) = h^*(x)$.
- We denote classes of patch functions using the notation $\mathcal{F}_{\text{adv}}(\mathcal{X})$, and classes of consistent patch functions using the notation $\mathcal{F}_{\text{adv}}(\mathcal{X}, h^*)$.

Patch Functions

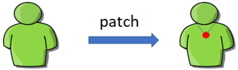


Figure: Patch Function

Definition (Patch Functions (See Definition 1))

- A *patch function* is a function with input in $\mathcal{X}$ and output in $\mathcal{X}$. A patch function is *consistent* with a ground-truth classifier h^* if for all $x \in \text{Supp}(\mathcal{D})$, we have $h^*(\text{patch}(x)) = h^*(x)$.
- We denote classes of patch functions using the notation $\mathcal{F}_{\text{adv}}(\mathcal{X})$, and classes of consistent patch functions using the notation $\mathcal{F}_{\text{adv}}(\mathcal{X}, h^*)$.
- Convention – $\mathcal{F}_{\text{adv}}$ always contains the identity function.

Order Of Events (Informal)

- Learner and adversary are given a learning problem with true classifier h^*; adversary is given h^*.

Order Of Events (Informal)

- Learner and adversary are given a learning problem with true classifier h^*; adversary is given h^*.

Order Of Events (Informal)

- Learner and adversary are given a learning problem with true classifier h^*; adversary is given h^*.

- Learner and adversary are revealed a set of patch functions $\mathcal{F}_{\text{adv}}$. The adversary selects a patch function consistent with h^*.

Order Of Events (Informal)

- Learner and adversary are given a learning problem with true classifier h^*; adversary is given h^*.

- Learner and adversary are revealed a set of patch functions $\mathcal{F}_{\text{adv}}$. The adversary selects a patch function consistent with h^*.

Order Of Events (Informal)

- Learner and adversary are given a learning problem with true classifier h^*; adversary is given h^*.

- Learner and adversary are revealed a set of patch functions $\mathcal{F}_{\text{adv}}$. The adversary selects a patch function consistent with h^*.

- Adversary injects patched examples into the training set and learner recovers a classifier using ERM on the union of the clean and patched data.

Order Of Events (Informal)

- Learner and adversary are given a learning problem with true classifier h^*; adversary is given h^*.

- Learner and adversary are revealed a set of patch functions $\mathcal{F}_{\text{adv}}$. The adversary selects a patch function consistent with h^*.

- Adversary injects patched examples into the training set and learner recovers a classifier using ERM on the union of the clean and patched data.

Table of Contents

Adversary's Goal and Restrictions (See Problem 2)

Loosely, the adversary wants the learner to simultaneously:

Adversary's Goal and Restrictions (See Problem 2)

Loosely, the adversary wants the learner to simultaneously:

- Memorize a function of the adversary's choice on data with seemingly irrelevant patches added.

Adversary's Goal and Restrictions (See Problem 2)

Loosely, the adversary wants the learner to simultaneously:

- Memorize a function of the adversary's choice on data with seemingly irrelevant patches added.
- Not detect the presence of corruptions from ERM alone.

Adversary's Goal and Restrictions (See Problem 2)

Loosely, the adversary wants the learner to simultaneously:

- Memorize a function of the adversary's choice on data with seemingly irrelevant patches added.
- Not detect the presence of corruptions from ERM alone.

The adversary is not all-powerful! They:

Adversary's Goal and Restrictions (See Problem 2)

Loosely, the adversary wants the learner to simultaneously:

- Memorize a function of the adversary's choice on data with seemingly irrelevant patches added.
- Not detect the presence of corruptions from ERM alone.

The adversary is not all-powerful! They:

- Can only corrupt and inject mislabeled "typical data" – formally, the adversary can inject a training set $S_{\text{adv}} \sim \text{patch}\left(\mathcal{D}|y \neq t\right)^m$, where m is the number of corrupted examples.

Adversary's Goal and Restrictions (See Problem 2)

Loosely, the adversary wants the learner to simultaneously:

- Memorize a function of the adversary's choice on data with seemingly irrelevant patches added.
- Not detect the presence of corruptions from ERM alone.

The adversary is not all-powerful! They:

- Can only corrupt and inject mislabeled "typical data" – formally, the adversary can inject a training set $S_{\text{adv}} \sim \text{patch}\left(\mathcal{D}|y \neq t\right)^m$, where m is the number of corrupted examples.
- Must succeed no matter how much clean data the learner draws.

Adversary's Goal and Restrictions (See Problem 2)

Loosely, the adversary wants the learner to simultaneously:

- Memorize a function of the adversary's choice on data with seemingly irrelevant patches added.
- Not detect the presence of corruptions from ERM alone.

The adversary is not all-powerful! They:

- Can only corrupt and inject mislabeled "typical data" – formally, the adversary can inject a training set $S_{\text{adv}} \sim \text{patch}\left(\mathcal{D} | y \neq t\right)^m$, where m is the number of corrupted examples.
- Must succeed no matter how much clean data the learner draws.

Question

Can we quantify the properties present in a learning problem that lends itself to such a memorization?
(This section)

Defining Memorization Capacity

Definition (Memorization Capacity (See Definition 7))

Suppose we are in a setting where we are learning a hypothesis class $\mathcal{H}$ over a domain $\mathcal{X}$ under distribution $\mathcal{D}$.
We say we can *memorize k irrelevant* subsets from a family $\mathcal{C} \subseteq 2^{\mathcal{X}}$ atop a fixed h if we can find k nonempty sets $X_1, \ldots, X_k \in \mathcal{C}$ satisfying $\mu_{\mathcal{D}}(X_i) = 0$ for all $i \in [k]$ such that for all $b \in \{\pm 1\}^k$, there exists a classifier $\widehat{h} \in \mathcal{H}$ satisfying:

- For all $x \in X_i$, we have $\widehat{h}(x) = b_i$.
- $\Pr_{x \sim \mathcal{D}} \left[\widehat{h}(x) = h(x) \right] = 1$.

Defining Memorization Capacity

Definition (Memorization Capacity (See Definition 7))

Suppose we are in a setting where we are learning a hypothesis class $\mathcal{H}$ over a domain $\mathcal{X}$ under distribution $\mathcal{D}$.

We say we can *memorize k irrelevant* subsets from a family $\mathcal{C} \subseteq 2^{\mathcal{X}}$ atop a fixed h if we can find k nonempty sets $X_1, \ldots, X_k \in \mathcal{C}$ satisfying $\mu_{\mathcal{D}}(X_i) = 0$ for all $i \in [k]$ such that for all $b \in \{\pm 1\}^k$, there exists a classifier $\widehat{h} \in \mathcal{H}$ satisfying:

- For all $x \in X_i$, we have $\widehat{h}(x) = b_i$.
- $\Pr_{x \sim \mathcal{D}}\left[\widehat{h}(x) = h(x)\right] = 1$.

We define $\mathrm{mcap}_{\mathcal{X},\mathcal{D}}(h, \mathcal{C})$ to be the maximum number of sets from $\mathcal{C}$ we can memorize for a fixed h.

Defining Memorization Capacity

Definition (Memorization Capacity (See Definition 7))

Suppose we are in a setting where we are learning a hypothesis class $\mathcal{H}$ over a domain $\mathcal{X}$ under distribution $\mathcal{D}$.
We say we can *memorize* k *irrelevant* subsets from a family $\mathcal{C} \subseteq 2^{\mathcal{X}}$ atop a fixed h if we can find k nonempty sets $X_1, \ldots, X_k \in \mathcal{C}$ satisfying $\mu_{\mathcal{D}}(X_i) = 0$ for all $i \in [k]$ such that for all $b \in \{\pm 1\}^k$, there exists a classifier $\widehat{h} \in \mathcal{H}$ satisfying:

- For all $x \in X_i$, we have $\widehat{h}(x) = b_i$.
- $\Pr_{x \sim \mathcal{D}} \left[\widehat{h}(x) = h(x) \right] = 1$.

We define $\mathrm{mcap}_{\mathcal{X},\mathcal{D}}(h, \mathcal{C})$ to be the maximum number of sets from $\mathcal{C}$ we can memorize for a fixed h.We omit the argument $\mathcal{C}$ when $\mathcal{C}$ is the set of all measurable subsets of $\mathcal{X}$.

Defining Memorization Capacity

Definition (Memorization Capacity (See Definition 7))

Suppose we are in a setting where we are learning a hypothesis class $\mathcal{H}$ over a domain $\mathcal{X}$ under distribution $\mathcal{D}$.
We say we can *memorize k irrelevant* subsets from a family $\mathcal{C} \subseteq 2^{\mathcal{X}}$ atop a fixed h if we can find k nonempty sets $X_1, \ldots, X_k \in \mathcal{C}$ satisfying $\mu_{\mathcal{D}}(X_i) = 0$ for all $i \in [k]$ such that for all $b \in \{\pm 1\}^k$, there exists a classifier $\widehat{h} \in \mathcal{H}$ satisfying:

- For all $x \in X_i$, we have $\widehat{h}(x) = b_i$.
- $\Pr_{x \sim \mathcal{D}} \left[\widehat{h}(x) = h(x) \right] = 1$.

We define $\mathrm{mcap}_{\mathcal{X},\mathcal{D}}(h, \mathcal{C})$ to be the maximum number of sets from $\mathcal{C}$ we can memorize for a fixed h.We omit the argument $\mathcal{C}$ when $\mathcal{C}$ is the set of all measurable subsets of $\mathcal{X}$.Finally, we define $\mathrm{mcap}_{\mathcal{X},\mathcal{D}}(\mathcal{H}) := \sup_{h \in \mathcal{H}} \mathrm{mcap}_{\mathcal{X},\mathcal{D}}(h)$.

Example – Decision Lists

Let $\mathcal{H}$ be the class of decision lists over $\mathcal{X} = \{0, 1\}^3$ and let $\mathcal{D}$ be the uniform distribution over the vectors whose last bit is 0.

Example – Decision Lists

Let $\mathcal{H}$ be the class of decision lists over $\mathcal{X} = \{0,1\}^3$ and let $\mathcal{D}$ be the uniform distribution over the vectors whose last bit is 0.
Fix any $h^* \in \mathcal{H}$. The subset $\{x \ : \ x_3 = 1\} \subset \mathcal{X}$ is "memorizable."

Example – Decision Lists

Let $\mathcal{H}$ be the class of decision lists over $\mathcal{X} = \{0,1\}^3$ and let $\mathcal{D}$ be the uniform distribution over the vectors whose last bit is 0.
Fix any $h^* \in \mathcal{H}$. The subset $\{x \ : \ x_3 = 1\} \subset \mathcal{X}$ is "memorizable."
Fix a target label t and consider the following function $\widehat{h}(x)$:

$$\begin{aligned}&\text{if } x_3 = 1 \text{ then } t\\&\text{else if } x_3 = 0 \text{ then } h^*(x)\end{aligned}$$

Example – Decision Lists

Let $\mathcal{H}$ be the class of decision lists over $\mathcal{X} = \{0,1\}^3$ and let $\mathcal{D}$ be the uniform distribution over the vectors whose last bit is 0.
Fix any $h^* \in \mathcal{H}$. The subset $\{x \ : \ x_3 = 1\} \subset \mathcal{X}$ is "memorizable."
Fix a target label t and consider the following function $\widehat{h}(x)$:

$$\begin{aligned} &\text{if } x_3 = 1 \text{ then } t \\ &\text{else if } x_3 = 0 \text{ then } h^*(x) \end{aligned}$$

We have:

$$\Pr_{x \sim \mathcal{D}}\left[\widehat{h}(x) = h^*(x)\right] = 1$$

and:

$$\Pr_{x \sim \mathcal{D}}\left[\widehat{h}(\text{patch}\,(x)) = t\right] = 1$$

Table of Contents

Main Result 1 – Memorizing Irrelevant Information

Theorem (Informal Restatement of Theorems 9 and 10)

Memorization capacity with respect to images of valid attacks dictates the number of backdoor attacks simultaneously possible in a learning problem.

Main Result 1 – Memorizing Irrelevant Information

Theorem (Informal Restatement of Theorems 9 and 10)

Memorization capacity with respect to images of valid attacks dictates the number of backdoor attacks simultaneously possible in a learning problem.

Upshot – argue about robustness of learning problems via memorization capacity (See Section 2.3.1).

Example – Overparameterized Linear Separators

Suppose we are in a setting where we are learning a linear separator. Our data lies in a low-dimensional subspace. Let the set of valid perturbations consist of additive functions with short vectors.

- $h^*(x) = \text{sign}(\langle w, x \rangle)$ where $\|w\| \leq {}^1\!/\!_\gamma$
- $\mathcal{H} = \left\{ h(x) \ : \ h(x) = \text{sign}(\langle w, x \rangle), w \in \mathbb{R}^d \right\}$
- $\mathcal{X} = \mathbb{R}^d$ and $\text{Supp}(\mathcal{D}) = \left\{ x \ : \ x = Ay, y \in \mathbb{R}^{d-k} \right\}$
- $\mathcal{F}_{\text{adv}} = \left\{ \text{patch} \ : \ \text{patch}(x) = x + \eta, \eta \in \mathbb{R}^d, \|\eta\| \leq \gamma \right\}$

Then, $\text{mcap}_{\mathcal{X},\mathcal{D}}(h^*, \mathcal{C}(\mathcal{F}_{\text{adv}})) \geq k$.

Example – Overparameterized Linear Separators

Suppose we are in a setting where we are learning a linear separator. Our data lies in a low-dimensional subspace. Let the set of valid perturbations consist of additive functions with short vectors.

- $h^*(x) = \text{sign}(\langle w, x \rangle)$ where $\|w\| \leq 1/\gamma$
- $\mathcal{H} = \left\{ h(x) \ : \ h(x) = \text{sign}(\langle w, x \rangle), w \in \mathbb{R}^d \right\}$
- $\mathcal{X} = \mathbb{R}^d$ and $\text{Supp}(\mathcal{D}) = \left\{ x \ : \ x = Ay, y \in \mathbb{R}^{d-k} \right\}$
- $\mathcal{F}_{\text{adv}} = \left\{ \text{patch} \ : \ \text{patch}(x) = x + \eta, \eta \in \mathbb{R}^d, \|\eta\| \leq \gamma \right\}$

Then, $\text{mcap}_{\mathcal{X},\mathcal{D}}(h^*, \mathcal{C}(\mathcal{F}_{\text{adv}})) \geq k$.

Example – Overparameterized Linear Separators

Suppose we are in a setting where we are learning a linear separator. Our data lies in a low-dimensional subspace. Let the set of valid perturbations consist of additive functions with short vectors.

- $h^*(x) = \text{sign}(\langle w, x \rangle)$ where $\|w\| \leq 1/\gamma$
- $\mathcal{H} = \{h(x) : h(x) = \text{sign}(\langle w, x \rangle), w \in \mathbb{R}^d\}$
- $\mathcal{X} = \mathbb{R}^d$ and $\text{Supp}(\mathcal{D}) = \{x : x = Ay, y \in \mathbb{R}^{d-k}\}$
- $\mathcal{F}_{\text{adv}} = \{\text{patch} : \text{patch}(x) = x + \eta, \eta \in \mathbb{R}^d, \|\eta\| \leq \gamma\}$

Then, $\text{mcap}_{\mathcal{X},\mathcal{D}}(h^*, \mathcal{C}(\mathcal{F}_{\text{adv}})) \geq k$.

Example – Overparameterized Linear Separators

Suppose we are in a setting where we are learning a linear separator. Our data lies in a low-dimensional subspace. Let the set of valid perturbations consist of additive functions with short vectors.

- $h^*(x) = \text{sign}(\langle w, x \rangle)$ where $\|w\| \leq 1/\gamma$
- $\mathcal{H} = \{h(x) : h(x) = \text{sign}(\langle w, x \rangle), w \in \mathbb{R}^d\}$
- $\mathcal{X} = \mathbb{R}^d$ and $\text{Supp}(\mathcal{D}) = \{x : x = Ay, y \in \mathbb{R}^{d-k}\}$
- $\mathcal{F}_{\text{adv}} = \{\text{patch} : \text{patch}(x) = x + \eta, \eta \in \mathbb{R}^d, \|\eta\| \leq \gamma\}$

Then, $\text{mcap}_{\mathcal{X},\mathcal{D}}(h^*, \mathcal{C}(\mathcal{F}_{\text{adv}})) \geq k$.

Example – Overparameterized Linear Separators

Suppose we are in a setting where we are learning a linear separator. Our data lies in a low-dimensional subspace. Let the set of valid perturbations consist of additive functions with short vectors.

- $h^*(x) = \text{sign}(\langle w, x \rangle)$ where $\|w\| \leq 1/\gamma$
- $\mathcal{H} = \left\{ h(x) \ : \ h(x) = \text{sign}(\langle w, x \rangle), w \in \mathbb{R}^d \right\}$
- $\mathcal{X} = \mathbb{R}^d$ and $\text{Supp}(\mathcal{D}) = \left\{ x \ : \ x = Ay, y \in \mathbb{R}^{d-k} \right\}$
- $\mathcal{F}_{\text{adv}} = \left\{ \text{patch} \ : \ \text{patch}(x) = x + \eta, \eta \in \mathbb{R}^d, \|\eta\| \leq \gamma \right\}$

Then, $\text{mcap}_{\mathcal{X},\mathcal{D}}(h^*, \mathcal{C}(\mathcal{F}_{\text{adv}})) \geq k$.

Main Result 2 – Failing Loudly

Main Result 2 – Failing Loudly

Intuition – robust loss (i.e., adversarial training) can be used to measure the quality of the dataset.

Main Result 2 – Failing Loudly

Intuition – robust loss (i.e., adversarial training) can be used to measure the quality of the dataset.

Theorem (Informal Restatement of Theorem 14)

If it is possible to (agnostically) learn an adversarially robust classifier on a clean dataset, then there exists an algorithm that can announce whether a training set is corrupted by backdoor examples.

Main Result 2 – Failing Loudly

Intuition – robust loss (i.e., adversarial training) can be used to measure the quality of the dataset.

Theorem (Informal Restatement of Theorem 14)

If it is possible to (agnostically) learn an adversarially robust classifier on a clean dataset, then there exists an algorithm that can announce whether a training set is corrupted by backdoor examples.

Use case – Training algorithm can announce when data is contaminated, and this can prompt manual intervention. See Section 3.1.1 for numerical trials.

Main Result 3 – Filtering vs Generalizing

Let α be the fraction of $S_{\text{clean}} \cup S_{\text{adv}}$ that's corrupted.

Main Result 3 – Filtering vs Generalizing

Let α be the fraction of $S_{\text{clean}} \cup S_{\text{adv}}$ that's corrupted.

Theorem (Informal Restatement of Theorem 17)

If we can solve the backdoor filtering problem up to outlier tolerance α, then we can solve the robust generalization problem up to outlier tolerance α.

Main Result 3 – Filtering vs Generalizing

Let α be the fraction of $S_{\text{clean}} \cup S_{\text{adv}}$ that's corrupted.

Theorem (Informal Restatement of Theorem 17)

If we can solve the backdoor filtering problem up to outlier tolerance α, then we can solve the robust generalization problem up to outlier tolerance α.

Theorem (Informal Restatement of Theorem 18)

If we can solve the robust generalization problem up to outlier tolerance 2α, then we can solve the backdoor filtering problem up to outlier tolerance α.

Main Result 3 – Filtering vs Generalizing

Let α be the fraction of $S_{\text{clean}} \cup S_{\text{adv}}$ that's corrupted.

Theorem (Informal Restatement of Theorem 17)

If we can solve the backdoor filtering problem up to outlier tolerance α, then we can solve the robust generalization problem up to outlier tolerance α.

Theorem (Informal Restatement of Theorem 18)

If we can solve the robust generalization problem up to outlier tolerance 2α, then we can solve the backdoor filtering problem up to outlier tolerance α.

TL;DR – robust generalization and filtering are roughly statistically equivalent.

Main Result 3 – Filtering vs Generalizing

Let α be the fraction of $S_{\text{clean}} \cup S_{\text{adv}}$ that's corrupted.

Theorem (Informal Restatement of Theorem 17)

If we can solve the backdoor filtering problem up to outlier tolerance α, then we can solve the robust generalization problem up to outlier tolerance α.

Theorem (Informal Restatement of Theorem 18)

If we can solve the robust generalization problem up to outlier tolerance 2α, then we can solve the backdoor filtering problem up to outlier tolerance α.

TL;DR – robust generalization and filtering are roughly statistically equivalent.Both reductions assume black-box access to the robust loss and an algorithm to minimize the robust loss on an arbitrary dataset.

Conclusion

We have:

Conclusion

We have:

- Defined a formal framework within which one can discuss backdoor data poisoning attacks.

Conclusion

We have:

- Defined a formal framework within which one can discuss backdoor data poisoning attacks.
- Identified memorization capacity as a parameter that characterizes vulnerability to backdoor data poisoning attacks.

Conclusion

We have:

- Defined a formal framework within which one can discuss backdoor data poisoning attacks.
- Identified memorization capacity as a parameter that characterizes vulnerability to backdoor data poisoning attacks.
- Given a high-level algorithm for detecting training set contamination, under several assumptions.

Conclusion

We have:

- Defined a formal framework within which one can discuss backdoor data poisoning attacks.
- Identified memorization capacity as a parameter that characterizes vulnerability to backdoor data poisoning attacks.
- Given a high-level algorithm for detecting training set contamination, under several assumptions.
- Under similar assumptions, shown that backdoor filtering and robust generalization are nearly equivalent.

Open Questions

- Can we reduce the memorization capacity present in a learning problem as a robust learning strategy?

Open Questions

- Can we reduce the memorization capacity present in a learning problem as a robust learning strategy?
- What can be proven in a finite-data setting?

Open Questions

- Can we reduce the memorization capacity present in a learning problem as a robust learning strategy?
- What can be proven in a finite-data setting?
- How powerful is an adversary against a learner using a more sophisticated family of learners (e.g. regularized ERM)?

Open Questions

- Can we reduce the memorization capacity present in a learning problem as a robust learning strategy?
- What can be proven in a finite-data setting?
- How powerful is an adversary against a learner using a more sophisticated family of learners (e.g. regularized ERM)?
- For what problems do there exist backdoor-robust learning algorithms?

Open Questions

- Can we reduce the memorization capacity present in a learning problem as a robust learning strategy?
- What can be proven in a finite-data setting?
- How powerful is an adversary against a learner using a more sophisticated family of learners (e.g. regularized ERM)?
- For what problems do there exist backdoor-robust learning algorithms?

Thank you!