Trustworthy Machine Learning
Unlearning 2

Sangdon Park

POSTECH

Unlearning by Theorists

Descent-to-Delete:
Gradient-Based Methods for Machine Unlearning

Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi

University of Pennsylvania

July 7, 2020

Abstract

We study the data deletion problem for convex models. By leveraging techniques from convex optimization and reservoir sampling, we give the first data deletion algorithms that are able to handle an arbitrarily long sequence of adversarial updates while promising both per-deletion run-time and steady-state error that do not grow with the length of the update sequence. We also introduce several new conceptual distinctions: for example, we can ask that after a deletion, the entire state maintained by the optimization algorithm is statistically indistinguishable from the state that would have resulted had we retrained, or we can ask for the weaker condition that only the *observable output* is statistically indistinguishable from the observable output that would have resulted from retraining. We are able to give more efficient deletion algorithms under this weaker deletion criterion.

- Certified removal [Guo et al., 2020]: Should we retrain a given base model?
- Algorithmic Learning Theory (2021)
  - Convex (again!)
  - Add / Delete sequentially

Setup (1/2)

- $\mathcal{Z}$: data space

Setup (1/2)

- $\mathcal{Z}$: data space
- $\mathcal{D} \subseteq \mathcal{Z}$: a dataset (precisely, a multi-set)

Setup (1/2)

- $\mathcal{Z}$: data space
- $\mathcal{D} \subseteq \mathcal{Z}$: a dataset (precisely, a multi-set)
- The dataset $\mathcal{D}$ can be modified by *adding* or *removing* one element from the dataset.

Setup (1/2)

- $\mathcal{Z}$: data space
- $\mathcal{D} \subseteq \mathcal{Z}$: a dataset (precisely, a multi-set)
- The dataset $\mathcal{D}$ can be modified by *adding* or *removing* one element from the dataset.
  - $\mathcal{T} := \{\text{“}\texttt{add}\text{”}, \text{“}\texttt{delete}\text{”}\}$: operations

Setup (1/2)

- $\mathcal{Z}$: data space
- $\mathcal{D} \subseteq \mathcal{Z}$: a dataset (precisely, a multi-set)
- The dataset $\mathcal{D}$ can be modified by *adding* or *removing* one element from the dataset.
  - $\mathcal{T} := \{\text{"}\texttt{add}\text{"}, \text{"}\texttt{delete}\text{"}\}$: operations
  - $u := (z, o) \in \mathcal{Z} \times \mathcal{T}$

Setup (1/2)

- $\mathcal{Z}$: data space
- $\mathcal{D} \subseteq \mathcal{Z}$: a dataset (precisely, a multi-set)
- The dataset $\mathcal{D}$ can be modified by *adding* or *removing* one element from the dataset.
  - $\mathcal{T} := \{\text{“}\texttt{add}\text{”}, \text{“}\texttt{delete}\text{”}\}$: operations
  - $u := (z, o) \in \mathcal{Z} \times \mathcal{T}$
  - $\mathcal{D} \circ u$: an update operation

$$\mathcal{D} \circ u := \begin{cases} \mathcal{D} \cup \{z\}, & \text{if } o = \text{“}\texttt{add}\text{”} \\ \mathcal{D} \setminus \{z\}, & \text{if } o = \text{“}\texttt{remove}\text{”} \end{cases}$$

Setup (1/2)

- $\mathcal{Z}$: data space
- $\mathcal{D} \subseteq \mathcal{Z}$: a dataset (precisely, a multi-set)
- The dataset $\mathcal{D}$ can be modified by *adding* or *removing* one element from the dataset.
  - $\mathcal{T} := \{\text{``}\texttt{add}\text{''}, \text{``}\texttt{delete}\text{''}\}$: operations
  - $u := (z, o) \in \mathcal{Z} \times \mathcal{T}$
  - $\mathcal{D} \circ u$: an update operation

$$\mathcal{D} \circ u := \begin{cases} \mathcal{D} \cup \{z\}, & \text{if } o = \text{``}\texttt{add}\text{''} \\ \mathcal{D} \setminus \{z\}, & \text{if } o = \text{``}\texttt{remove}\text{''} \end{cases}$$

- Θ: model space

Setup (2/2)

- $\mathcal{A} : \mathcal{Z}^* \to \Theta$: a learning algorithm
  - This produces an initial model.

Setup (2/2)

- $\mathcal{A} : \mathcal{Z}^* \to \Theta$: a learning algorithm
  - This produces an initial model.
- $\mathcal{R}_\mathcal{A} : \mathcal{Z}^* \times (\mathcal{Z} \times \mathcal{T}) \times \Theta \to \Theta$: an unlearning algorithm
  - This produces an update, secrete model.
  - Is this enough?

Setup (2/2)

- $\mathcal{A} : \mathcal{Z}^* \to \Theta$: a learning algorithm
  - This produces an initial model.
- $\mathcal{R}_{\mathcal{A}} : \mathcal{Z}^* \times (\mathcal{Z} \times \mathcal{T}) \times \Theta \to \Theta$: an unlearning algorithm
  - This produces an update, secrete model.
  - Is this enough?
- $f_{\text{publish}} : \Theta \to \Theta$: a publishing function
  - This maps a secrete model $\hat{\theta}$ to a public model $\tilde{\theta}$ by adding noise
  - Why do we need this? Hide an updated sample.

Setup (2/2)

- $\mathcal{A} : \mathcal{Z}^* \to \Theta$: a learning algorithm
  - This produces an initial model.
- $\mathcal{R}_{\mathcal{A}} : \mathcal{Z}^* \times (\mathcal{Z} \times \mathcal{T}) \times \Theta \to \Theta$: an unlearning algorithm
  - This produces an update, secrete model.
  - Is this enough?
- $f_{\text{publish}} : \Theta \to \Theta$: a publishing function
  - This maps a secrete model $\hat{\theta}$ to a public model $\tilde{\theta}$ by adding noise
  - Why do we need this? Hide an updated sample.
- Sequential updates
  - An update arrives sequentially, *i.e.*, $u_1, u_2, \ldots, u_i, \ldots$
  - A model is updated sequentially, *i.e.*, $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_i, \ldots$

A Goodness Measure: (ε, δ)-Indistinguishability

Definition ((ε, δ)-Indistinguishability)

We say random variables $X \in \Omega$ and $Y \in \Omega$ are (ε, δ)-indistinguishable (and write $X \overset{\varepsilon,\delta}{\approx} Y$) if for all $S \subseteq \Omega$,

$$\mathbb{P}\{X \in S\} \leq e^{\varepsilon}\mathbb{P}\{Y \in S\} + \delta \quad \text{and}$$
$$\mathbb{P}\{Y \in S\} \leq e^{\varepsilon}\mathbb{P}\{X \in S\} + \delta.$$

- The ultimate goal: having $\mathbb{P}\{X \in S\} \approx \mathbb{P}\{Y \in S\}$

A Goodness Measure: (ε, δ)-unlearning

Definition ((ε, δ)-unlearning)

We say that $\mathcal{R}_{\mathcal{A}}$ is an (ε, δ)-unlearning algorithm for $\mathcal{A}$ with respect to a publishing function f_{publish} if all datasets $\mathcal{D}_0$, all update step $i \geq 1$, and all update sequences $\mathcal{U} = (u_1, u_2, \dots)$,

$$f_{\text{publish}}(\mathcal{R}_{\mathcal{A}}(\mathcal{D}_{i-1}, u_i, \hat{\theta}_{i-1})) \overset{\varepsilon,\delta}{\approx} f_{\text{publish}}(\mathcal{A}(\mathcal{D}_i)).$$

- $\mathcal{D}_i = \mathcal{D}_{i-1} \circ u_i$: The dataset is updated over time.
- $\hat{\theta}_i = \mathcal{R}_{\mathcal{A}}(\mathcal{D}_{i-1}, u_i, \hat{\theta}_{i-1})$: the secrete model $\hat{\theta}_i$ is updated over time via unlearning.

Empirical Risk Minimization (ERM)

Main Ingredient of Learning and Unlearning

Additional Setup:

- Θ: a convex and closed subset of $\mathbb{R}^d$
- $f : \Theta \times \mathcal{Z} \to \mathbb{R}$: a loss function
  - $f(\theta, z) = f_z(\theta)$
  - Here, we assume that the loss function is strongly convex (but can be relaxed).

Definition (empirical loss)

$$f_{\mathcal{D}}(\theta) := \frac{1}{n} \sum_{i=1}^{n} f_{z_i}(\theta)$$

Empirical Risk Minimization (ERM)

(α, β)-accuracy

Definition ((α, β)-accuracy)

We say a pair $(\mathcal{A}, \mathcal{R}_{\mathcal{A}})$ of learning and unlearning algorithms is (α, β)-accurate with respect to a publishing function f_{publish} if for any dataset $\mathcal{D}$, any update sequence $\mathcal{U}$, and any $i \geq 0$,

$$\mathbb{P}\left\{ f_{\mathcal{D}_i}(\tilde{\theta}_i) - \min_{\theta \in \Theta} f_{\mathcal{D}_i}(\theta) > \alpha \right\} < \beta.$$

- $\tilde{\theta}_i = f_{\text{publish}}(\mathcal{R}_{\mathcal{A}}(\mathcal{D}_{i-1}, u_i, \hat{\theta}_{i-1}))$: the public model, released by a publishing function after learning or unlearning
- Gradient descent can achieve (α, β)-accuracy.
- Note that the “accuracy” and “unlearning” are different metrics.

Empirical Risk Minimization (ERM)

(α, β)-accuracy

Definition ((α, β)-accuracy)

We say a pair $(\mathcal{A}, \mathcal{R}_{\mathcal{A}})$ of learning and unlearning algorithms is (α, β)-accurate with respect to a publishing function f_{publish} if for any dataset $\mathcal{D}$, any update sequence $\mathcal{U}$, and any $i \geq 0$,

$$\mathbb{P}\left\{ f_{\mathcal{D}_i}(\tilde{\theta}_i) - \min_{\theta \in \Theta} f_{\mathcal{D}_i}(\theta) > \alpha \right\} < \beta.$$

- $\tilde{\theta}_i = f_{\text{publish}}(\mathcal{R}_{\mathcal{A}}(\mathcal{D}_{i-1}, u_i, \hat{\theta}_{i-1}))$: the public model, released by a publishing function after learning or unlearning
- Gradient descent can achieve (α, β)-accuracy.
- Note that the "accuracy" and "unlearning" are different metrics.
- Trivial?

Empirical Risk Minimization (ERM)

(α, β)-accuracy

Definition ((α, β)-accuracy)

We say a pair $(\mathcal{A}, \mathcal{R}_{\mathcal{A}})$ of learning and unlearning algorithms is (α, β)-accurate with respect to a publishing function f_{publish} if for any dataset $\mathcal{D}$, any update sequence $\mathcal{U}$, and any $i \geq 0$,

$$\mathbb{P}\left\{ f_{\mathcal{D}_i}(\tilde{\theta}_i) - \min_{\theta \in \Theta} f_{\mathcal{D}_i}(\theta) > \alpha \right\} < \beta.$$

- $\tilde{\theta}_i = f_{\text{publish}}(\mathcal{R}_{\mathcal{A}}(\mathcal{D}_{i-1}, u_i, \hat{\theta}_{i-1}))$: the public model, released by a publishing function after learning or unlearning
- Gradient descent can achieve (α, β)-accuracy.
- Note that the "accuracy" and "unlearning" are different metrics.
- Trivial?
- PAC guarantee?

Method: Perturbed Gradient Descent

Learning

Algorithm 1 $\mathcal{A}$: **Learning** for Perturbed Gradient Descent

Input: dataset $\mathcal{D}$
Initialize $\theta'_0 \in \Theta$
for $t = 1, 2, \ldots T$ **do**
    $\theta'_t = \mathrm{Proj}_\Theta \left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}}(\theta'_{t-1})\right)$
Output: $\hat{\theta}_0 = \theta'_T$ ▷ Secret output

Method: Perturbed Gradient Descent

Learning

Algorithm 1 $\mathcal{A}$: **Learning** for Perturbed Gradient Descent

1: Input: dataset $\mathcal{D}$
2: Initialize $\theta'_0 \in \Theta$
3: **for** $t = 1, 2, \ldots T$ **do**
4: $\quad \theta'_t = \mathrm{Proj}_\Theta\left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}}(\theta'_{t-1})\right)$
5: Output: $\hat{\theta}_0 = \theta'_T$ ▷ Secret output

- Use gradient descent for learning our initial model $\hat{\theta}_0$

Method: Perturbed Gradient Descent

Learning

Algorithm 1 $\mathcal{A}$: **Learning** for Perturbed Gradient Descent

1: Input: dataset $\mathcal{D}$
2: Initialize $\theta'_0 \in \Theta$
3: **for** $t = 1, 2, \dots T$ **do**
4: $\quad \theta'_t = \mathrm{Proj}_{\Theta}\left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}}(\theta'_{t-1})\right)$
5: Output: $\hat{\theta}_0 = \theta'_T$ ▷ Secret output

- Use gradient descent for learning our initial model $\hat{\theta}_0$
- $\hat{\theta}_0$ can be “accurate” enough if T and η_t are properly chosen as $f_{\mathcal{D}}$ is strongly convex

Method: Perturbed Gradient Descent

Unlearning

Algorithm 2 $\mathcal{R}_\mathcal{A}$: ith **Unlearning** for Perturbed Gradient Descent

Input: dataset $\mathcal{D}_{i-1}$, update u_i, model θ_i
Update dataset $\mathcal{D}_i = \mathcal{D}_{i-1} \circ u_i$
Initialize $\theta'_0 = \theta_i$
for $t = 1, 2, \ldots T_i$ **do**
    $\theta'_t = \mathrm{Proj}_\Theta \left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}_i}(\theta'_{t-1})\right)$
Output $\hat{\theta}_i = \theta'_{T_i}$ ▷ Secret output

Method: Perturbed Gradient Descent

Unlearning

Algorithm 2 $\mathcal{R}_{\mathcal{A}}$: ith **Unlearning** for Perturbed Gradient Descent

1: Input: dataset $\mathcal{D}_{i-1}$, update u_i, model θ_i
2: Update dataset $\mathcal{D}_i = \mathcal{D}_{i-1} \circ u_i$
3: Initialize $\theta'_0 = \theta_i$
4: **for** $t = 1, 2, \ldots T_i$ **do**
5: $\quad \theta'_t = \text{Proj}_{\Theta}\left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}_i}(\theta'_{t-1})\right)$
6: Output $\hat{\theta}_i = \theta'_{T_i}$ ▷ Secret output

- Each i-th unlearning step, a model is updated.

Method: Perturbed Gradient Descent

Unlearning

Algorithm 2 $\mathcal{R}_{\mathcal{A}}$: ith **Unlearning** for Perturbed Gradient Descent

1: Input: dataset $\mathcal{D}_{i-1}$, update u_i, model θ_i
2: Update dataset $\mathcal{D}_i = \mathcal{D}_{i-1} \circ u_i$
3: Initialize $\theta'_0 = \theta_i$
4: **for** $t = 1, 2, \ldots T_i$ **do**
5: $\quad \theta'_t = \mathrm{Proj}_{\Theta}\left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}_i}(\theta'_{t-1})\right)$
6: Output $\hat{\theta}_i = \theta'_{T_i}$ ▷ Secret output

- Each i-th unlearning step, a model is updated.
- The algorithm is still gradient decent computed over the entire dataset.

Method: Perturbed Gradient Descent

Unlearning

Algorithm 2 $\mathcal{R}_{\mathcal{A}}$: ith **Unlearning** for Perturbed Gradient Descent

Input: dataset $\mathcal{D}_{i-1}$, update u_i, model θ_i
Update dataset $\mathcal{D}_i = \mathcal{D}_{i-1} \circ u_i$
Initialize $\theta'_0 = \theta_i$
for $t = 1, 2, \ldots T_i$ **do**
    $\theta'_t = \text{Proj}_{\Theta}\left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}_i}(\theta'_{t-1})\right)$
Output $\hat{\theta}_i = \theta'_{T_i}$ ▷ Secret output

- Each i-th unlearning step, a model is updated.
- The algorithm is still gradient decent computed over the entire dataset.
    - Why not gradient ascent?

Method: Perturbed Gradient Descent

Unlearning

Algorithm 2 $\mathcal{R}_\mathcal{A}$: ith **Unlearning** for Perturbed Gradient Descent

1: Input: dataset $\mathcal{D}_{i-1}$, update u_i, model θ_i
2: Update dataset $\mathcal{D}_i = \mathcal{D}_{i-1} \circ u_i$
3: Initialize $\theta'_0 = \theta_i$
4: **for** $t = 1, 2, \ldots T_i$ **do**
5: $\quad \theta'_t = \mathrm{Proj}_\Theta \left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}_i}(\theta'_{t-1})\right)$
6: Output $\hat{\theta}_i = \theta'_{T_i}$ ▷ Secret output

- Each i-th unlearning step, a model is updated.
- The algorithm is still gradient decent computed over the entire dataset.
  - Why not gradient ascent?
  - Is it efficient in T_i? Why not the Newton's method?

Method: Perturbed Gradient Descent

Unlearning

Algorithm 2 $\mathcal{R}_{\mathcal{A}}$: ith **Unlearning** for Perturbed Gradient Descent

1: Input: dataset $\mathcal{D}_{i-1}$, update u_i, model θ_i
2: Update dataset $\mathcal{D}_i = \mathcal{D}_{i-1} \circ u_i$
3: Initialize $\theta'_0 = \theta_i$
4: **for** $t = 1, 2, \ldots T_i$ **do**
5: $\quad \theta'_t = \text{Proj}_{\Theta}\left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}_i}(\theta'_{t-1})\right)$
6: Output $\hat{\theta}_i = \theta'_{T_i}$ ▷ Secret output

- Each i-th unlearning step, a model is updated.
- The algorithm is still gradient decent computed over the entire dataset.
  - Why not gradient ascent?
  - Is it efficient in T_i? Why not the Newton's method?
  - What's the difference from retraining?

Method: Perturbed Gradient Descent

Unlearning

Algorithm 2 $\mathcal{R}_{\mathcal{A}}$: ith **Unlearning** for Perturbed Gradient Descent

1: Input: dataset $\mathcal{D}_{i-1}$, update u_i, model θ_i
2: Update dataset $\mathcal{D}_i = \mathcal{D}_{i-1} \circ u_i$
3: Initialize $\theta'_0 = \theta_i$
4: **for** $t = 1, 2, \dots T_i$ **do**
5: $\quad \theta'_t = \mathrm{Proj}_{\Theta}\left(\theta'_{t-1} - \eta_t \nabla f_{\mathcal{D}_i}(\theta'_{t-1})\right)$
6: Output $\hat{\theta}_i = \theta'_{T_i}$ ▷ Secret output

- Each i-th unlearning step, a model is updated.
- The algorithm is still gradient decent computed over the entire dataset.
  - Why not gradient ascent?
  - Is it efficient in T_i? Why not the Newton's method?
  - What's the difference from retraining?
- This algorithm can be "accurate" given proper T_i and η_t.

Method: Perturbed Gradient Descent

Publishing

Algorithm 3 f_{publish}: Publishing function

1: Input: $\hat{\theta} \in \mathbb{R}^d$
2: Draw $Z \sim \mathcal{N}\left(0, \sigma^2 \mathbb{I}_d\right)$
3: Output: $\tilde{\theta} = \hat{\theta} + Z$ ▷ Public output

Method: Perturbed Gradient Descent

Publishing

Algorithm 3 f_{publish}: Publishing function

1: Input: $\hat{\theta} \in \mathbb{R}^d$
2: Draw $Z \sim \mathcal{N}\left(0, \sigma^2 \mathbb{I}_d\right)$
3: Output: $\tilde{\theta} = \hat{\theta} + Z$ ▷ Public output

- Why we have to add noise?

Method: Perturbed Gradient Descent

Publishing

Algorithm 3 f_{publish}: Publishing function

1: Input: $\hat{\theta} \in \mathbb{R}^d$
2: Draw $Z \sim \mathcal{N}\left(0, \sigma^2 \mathbb{I}_d\right)$
3: Output: $\tilde{\theta} = \hat{\theta} + Z$ ▷ Public output

- Why we have to add noise?
  - Hide our model (like DP)

Method: Perturbed Gradient Descent

Publishing

Algorithm 3 f_{publish}: Publishing function

1: Input: $\hat{\theta} \in \mathbb{R}^d$
2: Draw $Z \sim \mathcal{N}\left(0, \sigma^2 \mathbb{I}_d\right)$
3: Output: $\tilde{\theta} = \hat{\theta} + Z$ ▷ Public output

- Why we have to add noise?
  - Hide our model (like DP)
- How much noise should we add?

Method: Perturbed Gradient Descent
Publishing

Algorithm 3 f_{publish}: Publishing function

1: Input: $\hat{\theta} \in \mathbb{R}^d$
2: Draw $Z \sim \mathcal{N}\left(0, \sigma^2 \mathbb{I}_d\right)$
3: Output: $\tilde{\theta} = \hat{\theta} + Z$ ▷ Public output

- Why we have to add noise?
  - Hide our model (like DP)
- How much noise should we add?
  - See the theorem statement; but does it hurt the accuracy of the published function?

Method: Perturbed Gradient Descent

Publishing

Algorithm 3 f_{publish}: Publishing function

1: Input: $\hat{\theta} \in \mathbb{R}^d$
2: Draw $Z \sim \mathcal{N}\left(0, \sigma^2 \mathbb{I}_d\right)$
3: Output: $\tilde{\theta} = \hat{\theta} + Z$ ▷ Public output

- Why we have to add noise?
  - Hide our model (like DP)
- How much noise should we add?
  - See the theorem statement; but does it hurt the accuracy of the published function?
- What's the main difference from the certified removal paper [Guo et al., 2020]?

Method: Perturbed Gradient Descent

Publishing

Algorithm 3 f_{publish}: Publishing function

1: Input: $\hat{\theta} \in \mathbb{R}^d$
2: Draw $Z \sim \mathcal{N}\left(0, \sigma^2 \mathbb{I}_d\right)$
3: Output: $\tilde{\theta} = \hat{\theta} + Z$ ▷ Public output

- Why we have to add noise?
  - Hide our model (like DP)
- How much noise should we add?
  - See the theorem statement; but does it hurt the accuracy of the published function?
- What's the main difference from the certified removal paper [Guo et al., 2020]?
  - We don't need to retrain our pre-trained model with noise.

Method: Perturbed Gradient Descent

Publishing

Algorithm 3 f_{publish}: Publishing function

1: Input: $\hat{\theta} \in \mathbb{R}^d$
2: Draw $Z \sim \mathcal{N}\left(0, \sigma^2 \mathbb{I}_d\right)$
3: Output: $\tilde{\theta} = \hat{\theta} + Z$ ▷ Public output

- Why we have to add noise?
  - Hide our model (like DP)
- How much noise should we add?
  - See the theorem statement; but does it hurt the accuracy of the published function?
- What's the main difference from the certified removal paper [Guo et al., 2020]?
  - We don't need to retrain our pre-trained model with noise.
  - Instead, it publishes a noisy model.

Method: Perturbed Gradient Descent

Publishing

Algorithm 3 f_{publish}: Publishing function

1: Input: $\hat{\theta} \in \mathbb{R}^d$
2: Draw $Z \sim \mathcal{N}\left(0, \sigma^2 \mathbb{I}_d\right)$
3: Output: $\tilde{\theta} = \hat{\theta} + Z$ ▷ Public output

- Why we have to add noise?
  - Hide our model (like DP)
- How much noise should we add?
  - See the theorem statement; but does it hurt the accuracy of the published function?
- What's the main difference from the certified removal paper [Guo et al., 2020]?
  - We don't need to retrain our pre-trained model with noise.
  - Instead, it publishes a noisy model.
  - However, can the published function be accurate enough?

Guarantee

Theorem (Trade-off between unlearning and accuracy (informal))

Suppose that

- *the loss function* f_z *is convex and "smooth",*
- *the learning algorithm* $\mathcal{A}$ *runs with* η_t *and* T *such that* $\hat{\theta}_0$ *is "accurate",*
- *the unlearning algorithm* $\mathcal{R}_{\mathcal{A}}$ *runs with* $\mathcal{I}$ *iterations, and*
- $\varepsilon = \mathcal{O}(\log 1/\delta)$,
- *the publishing function* $f_{publish}$ *runs with* $\sigma \propto \frac{1}{\sqrt{\varepsilon}}$.

Then, we have

- $\mathcal{R}_{\mathcal{A}}$ *is* (ε, δ)*-unlearning algorithm for* $\mathcal{A}$ *with respect to* $f_{publish}$ *and*
- *For any* β, $(\mathcal{A}, \mathcal{R}_{\mathcal{A}})$ *is* (α, β)*-accurate with respect to* $f_{publish}$ *when* $\alpha \propto \frac{1}{\varepsilon}$.

Guarantee

Theorem (Trade-off between unlearning and accuracy (informal))

Suppose that

- *the loss function* f_z *is convex and "smooth",*
- *the learning algorithm* $\mathcal{A}$ *runs with* η_t *and* T *such that* $\hat{\theta}_0$ *is "accurate",*
- *the unlearning algorithm* $\mathcal{R}_\mathcal{A}$ *runs with* $\mathcal{I}$ *iterations, and*
- $\varepsilon = \mathcal{O}(\log 1/\delta)$,
- *the publishing function* $f_{\textit{publish}}$ *runs with* $\sigma \propto \frac{1}{\sqrt{\varepsilon}}$.

Then, we have

- $\mathcal{R}_\mathcal{A}$ *is* (ε, δ)*-unlearning algorithm for* $\mathcal{A}$ *with respect to* $f_{\textit{publish}}$ *and*
- *For any* β, $(\mathcal{A}, \mathcal{R}_\mathcal{A})$ *is* (α, β)*-accurate with respect to* $f_{\textit{publish}}$ *when* $\alpha \propto \frac{1}{\varepsilon}$.

- (ε, δ)-unlearning: (unlearned model) $\approx$ (retrained model) $\Rightarrow$ (no information leak)

Guarantee

Theorem (Trade-off between unlearning and accuracy (informal))

Suppose that

- *the loss function* f_z *is convex and "smooth",*
- *the learning algorithm* $\mathcal{A}$ *runs with* η_t *and* T *such that* $\hat{\theta}_0$ *is "accurate",*
- *the unlearning algorithm* $\mathcal{R}_{\mathcal{A}}$ *runs with* $\mathcal{I}$ *iterations, and*
- $\varepsilon = \mathcal{O}(\log 1/\delta)$*,*
- *the publishing function* $f_{publish}$ *runs with* $\sigma \propto \frac{1}{\sqrt{\varepsilon}}$*.*

Then, we have

- $\mathcal{R}_{\mathcal{A}}$ *is* (ε, δ)*-unlearning algorithm for* $\mathcal{A}$ *with respect to* $f_{publish}$ *and*
- *For any* β*,* $(\mathcal{A}, \mathcal{R}_{\mathcal{A}})$ *is* (α, β)*-accurate with respect to* $f_{publish}$ *when* $\alpha \propto \frac{1}{\varepsilon}$*.*

- (ε, δ)-unlearning: (unlearned model) $\approx$ (retrained model) $\Rightarrow$ (no information leak)
- (α, β)-accuracy: the unlearned model is "accurate"

Guarantee

Theorem (Trade-off between unlearning and accuracy (informal))

Suppose that

- *the loss function* f_z *is convex and "smooth",*
- *the learning algorithm* $\mathcal{A}$ *runs with* η_t *and* T *such that* $\hat{\theta}_0$ *is "accurate",*
- *the unlearning algorithm* $\mathcal{R}_{\mathcal{A}}$ *runs with* $\mathcal{I}$ *iterations, and*
- $\varepsilon = \mathcal{O}(\log 1/\delta)$,
- *the publishing function* $f_{\textit{publish}}$ *runs with* $\sigma \propto \frac{1}{\sqrt{\varepsilon}}$.

Then, we have

- $\mathcal{R}_{\mathcal{A}}$ *is* (ε, δ)*-unlearning algorithm for* $\mathcal{A}$ *with respect to* $f_{\textit{publish}}$ *and*
- *For any* β, $(\mathcal{A}, \mathcal{R}_{\mathcal{A}})$ *is* (α, β)*-accurate with respect to* $f_{\textit{publish}}$ *when* $\alpha \propto \frac{1}{\varepsilon}$.

- (ε, δ)-unlearning: (unlearned model) $\approx$ (retrained model) $\Rightarrow$ (no information leak)
- (α, β)-accuracy: the unlearned model is "accurate"
- We cannot achieve two goals simultaneously, *i.e.*, $\alpha \propto \frac{1}{\varepsilon}$.

Discussion

- Do we retrain our base model?

Discussion

- Do we retrain our base model?
  - No. We can add noise when publish a (secret) model.

Discussion

- Do we retrain our base model?
  - No. We can add noise when publish a (secret) model.
- Does the noise level depend on a sample?

Discussion

- Do we retrain our base model?
  - No. We can add noise when publish a (secret) model.
- Does the noise level depend on a sample?
  - No. The noise level is data-independent, which can be in practical.

Discussion

- Do we retrain our base model?
  - No. We can add noise when publish a (secret) model.
- Does the noise level depend on a sample?
  - No. The noise level is data-independent, which can be in practical.
- What's the main difference between perturbed GD and DP?

Discussion

- Do we retrain our base model?
  - No. We can add noise when publish a (secret) model.
- Does the noise level depend on a sample?
  - No. The noise level is data-independent, which can be in practical.
- What's the main difference between perturbed GD and DP?
  - Not sure. Need some analysis.

Discussion

- Do we retrain our base model?
  - No. We can add noise when publish a (secret) model.
- Does the noise level depend on a sample?
  - No. The noise level is data-independent, which can be in practical.
- What's the main difference between perturbed GD and DP?
  - Not sure. Need some analysis.
- Is the "publishing" idea interesting?

Discussion

- Do we retrain our base model?
  - No. We can add noise when publish a (secret) model.
- Does the noise level depend on a sample?
  - No. The noise level is data-independent, which can be in practical.
- What's the main difference between perturbed GD and DP?
  - Not sure. Need some analysis.
- Is the "publishing" idea interesting?
  - add-nose-then-unlearn v.s. unlearn-then-add-noise

Discussion

- Do we retrain our base model?
  - No. We can add noise when publish a (secret) model.
- Does the noise level depend on a sample?
  - No. The noise level is data-independent, which can be in practical.
- What's the main difference between perturbed GD and DP?
  - Not sure. Need some analysis.
- Is the "publishing" idea interesting?
  - add-nose-then-unlearn v.s. unlearn-then-add-noise
- Is the "sequential" setting interesting?

Discussion

- Do we retrain our base model?
  - No. We can add noise when publish a (secret) model.
- Does the noise level depend on a sample?
  - No. The noise level is data-independent, which can be in practical.
- What's the main difference between perturbed GD and DP?
  - Not sure. Need some analysis.
- Is the "publishing" idea interesting?
  - add-nose-then-unlearn v.s. unlearn-then-add-noise
- Is the "sequential" setting interesting?
  - Looks practical

Reference I

C. Guo, T. Goldstein, A. Hannun, and L. Van Der Maaten. Certified data removal from machine learning models. In *International Conference on Machine Learning*, pages 3832–3842. PMLR, 2020.