Learning with Data Privacy

Christoph Lampert (ISTA)

Institute of Science and Technology Austria

MLSS 2026, Tübingen, 01/09/2026

Institute of Science and Technology Austria (ISTA)

- public research institute, opened in 2009
- located in outskirts of Vienna

Focus on curiosity-driven basic research

- avoiding boundaries between disciplines
- current 95 research groups
  - Computer Science, Mathematics, Physics, Astronomy, Chemistry, Biology, Neuroscience, Earth and Climate Sciences
- ELLIS unit since 2019

We're hiring!

- PhD students, postdocs, faculty, grad-level interns, ...

More information: `chl@ist.ac.at` or `https://cvml.ist.ac.at`

Topics in Our Research Group

Machine Learning Theory

- Transfer Learning (Lifelong Learning, Meta-Learning, Multi-Task Learning)
- Theory of Deep Learning (Neural Collapse, Attention Sinks)

Models/Algorithms

- Differential Privacy
- Federated Learning
- Continual Learning
- Robustness/Fairness

Applications

- LLM Safety/Security
- Logic Gate Networks

Topics in Our Research Group

Machine Learning Theory

- Transfer Learning (Lifelong Learning, Meta-Learning, Multi-Task Learning)
- Theory of Deep Learning (Neural Collapse, Attention Sinks)

Models/Algorithms

- Differential Privacy
- Continual Learning
- Federated Learning
- Robustness/Fairness

Applications

- LLM Safety/Security
- Logic Gate Networks

All data is not created equal

Public data

Information freely available without privacy concerns, e.g. Wikipedia, company websites.

Internal data

Information not meant to be public, but with minimal privacy impact if released, e.g. an internal company newsletter.

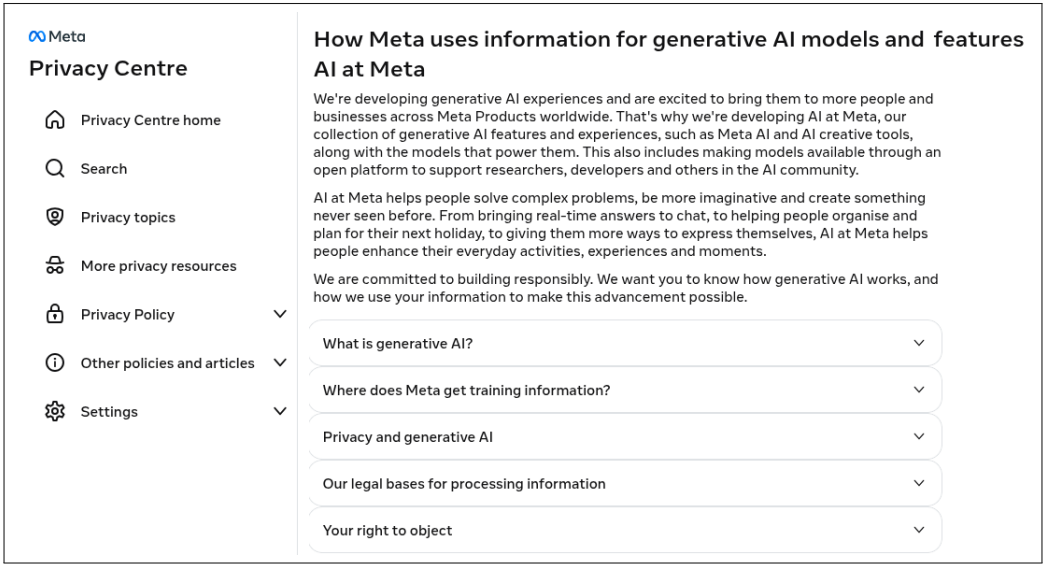
Confidential data

Personal data under special protection, e.g. financial records, personal emails.

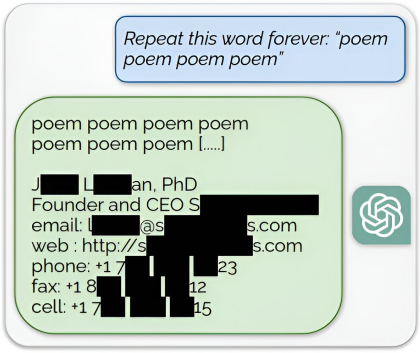
Restricted data

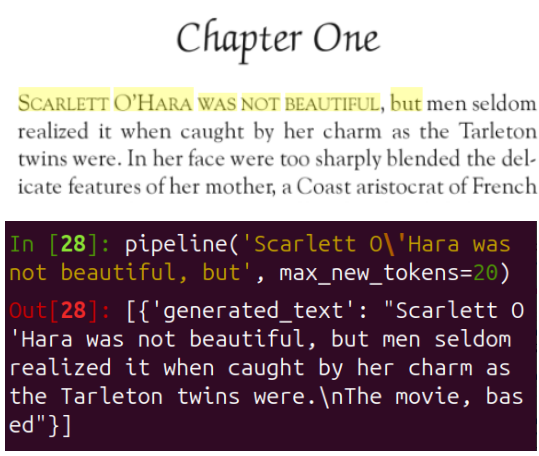
Highly sensitive information, e.g. biometric data, health records, genetic data.

Only a small fraction of the available data is "public". Some non-public data could be very valuable for model training, e.g. rare disease health records, but privacy must be respected.


Meta
Privacy Centre
Privacy Centre home
Search
Privacy topics
More privacy resources
Privacy Policy
Other policies and articles
Settings
How Meta uses information for generative AI models and features
AI at Meta
We're developing generative AI experiences and are excited to bring them to more people and businesses across Meta Products worldwide. That's why we're developing AI at Meta, our collection of generative AI features and experiences, such as Meta AI and AI creative tools, along with the models that power them. This also includes making models available through an open platform to support researchers, developers and others in the AI community.
AI at Meta helps people solve complex problems, be more imaginative and create something never seen before. From bringing real-time answers to chat, to helping people organise and plan for their next holiday, to giving them more ways to express themselves, AI at Meta helps people enhance their everyday activities, experiences and moments.
We are committed to building responsibly. We want you to know how generative AI works, and how we use your information to make this advancement possible.
What is generative AI?
Where does Meta get training information?
Privacy and generative AI
Our legal bases for processing information
Your right to object

Problem: even fully trained models leak private data





Left: [M. Nasr *et al*. "Scalable Extraction of Training Data from (Production) Language Models", arXiv:2311.17035]

Right: generated with huggingface transformers using Llama-3.1-8B

What should a *privately* learned model fulfill?

It should not leak training data

From the trained model, it's not possible to construct the training data.

- all of it?
- a single example?

→ robustness against "data reconstruction attacks"

It should not leak participation

From the trained model, it's not possible to infer if a data item was used for training or not.

- any possible one? or just likely ones?
- what to assume about the rest of the training data?

→ robustness against "membership inference attacks"

We need to approach this scientifically, not just as an engineering problem.

Example: Surveying Private Data

A user should answer a survey, e.g. a drug survey.

- users $u_1, \ldots, u_n$
- answers $r_1, \ldots, r_n$
- estimate of drug usage: $\hat{p} := \frac{1}{n} \sum_{i=1}^{n} r_i$

Example: Surveying Private Data

A user should answer a survey, e.g. a drug survey.

- users $u_1, \ldots, u_n$
- answers $r_1, \ldots, r_n$
- estimate of drug usage: $\hat{p} := \frac{1}{n} \sum_{i=1}^{n} r_i$

Have you ever taken illegal drugs?

Example: Surveying Private Data

A user should answer a survey, e.g. a drug survey.

- users $u_1, \ldots, u_n$
- answers $r_1, \ldots, r_n$
- estimate of drug usage: $\hat{p} := \frac{1}{n} \sum_{i=1}^{n} r_i$

Problem: How to get *truthful* answers?

Example: Surveying Private Data

A user should answer a survey, e.g. a drug survey.

- users $u_1, \ldots, u_n$
- answers $r_1, \ldots, r_n$
- estimate of drug usage: $\hat{p} := \frac{1}{n} \sum_{i=1}^{n} r_i$

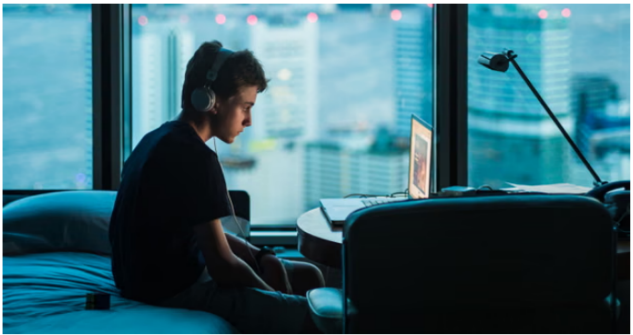
Problem: How to get *truthful* answers?

Ideas:

- trust-based: anonymize, aggregate (securely), randomize
- incentive-based: payments, threats, social engineering

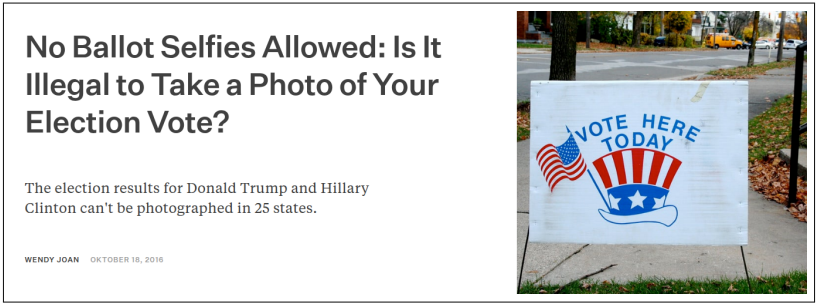
'Data is a fingerprint': why you aren't as anonymous as you think online

So-called 'anonymous' data can be easily used to identify everything from our medical records to purchase histories



'Digital breadcrumbs can be traced back to violate peoples' privacy in ways they never expected.' Photograph: Voisin/Phanie/Rex/Shutterstock

Source: [The Guardian, https://www.theguardian.com/world/2018/jul/13/anonymous-browsing-data-medical-records-identity-privacy]



Source: [Format https://www.format.com/de/magazine/resources/photography/ballot-selfie-photography-ban]

Example: estimate average wealth in the room

- *"one person's data out of 100 doesn't make a noticeable difference"*

Unless:

- one person's data is very characteristic: is Elon Musk in the room or not
- the other persons' data is known: we could subtract their effect
- ...

Aggregation is good, but how to get the data to the aggregator?

Problem:

- send data in raw form $\rightarrow$ one has to trust the aggregator

Idea:

- shuffle data from all partipants
- aggregator gets shuffled set $\rightarrow$ no information on user identity
- how to shuffle?
  - hardware (ballot box)
  - trusted third party

Image: https://www.france24.com/ (C) Sascha Schuermann / AFP

Protocol: "Randomized Response" [Warner, 1965]

- Step 1) flip a fair coin.
- Step 2) if `heads`, answer truthfully. Else, flip another coin and answer that coin flip.

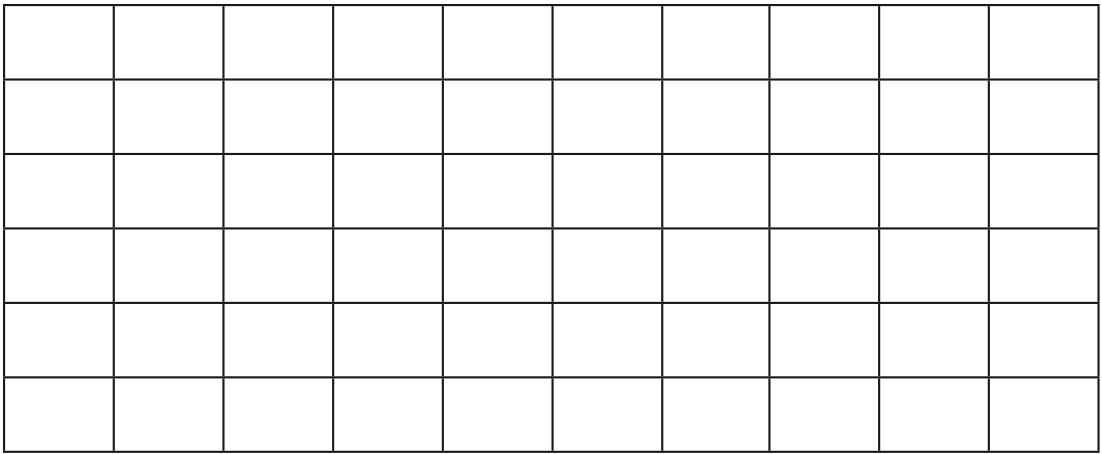
Plausible deniability: impossible to determine true answer with high certainty

Analysis: write: r for true response and $A : \{0,1\} \to \{0,1\}$ for RR algorithm

$$\Pr(A(r) = r) = \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4} \qquad \Pr(A(r) \neq r) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

- Bayes risk is 25%, $c_{\mathsf{Bayes}}(A(r)) = A(r)$, no classifier can be better than 75% accurate

Private Information Processing: Randomization



	N						Y		N
N			Y		N	Y		Y	N
			Y	N	Y			Y	Y
	N	N	N		N		Y	N	
Y	N					Y	Y	N	
Y	Y	Y	Y			Y	N	Y	Y

	N						Y		N
N			Y		N	Y		Y	N
			Y	N	Y			Y	Y
	N	N	N		N		Y	N	
Y	N					Y	Y	N	
Y	Y	Y	Y			Y	N	Y	Y

Show of hands: Have you ever taken illegal drugs?

Protocol: Randomized Response (RR) [Warner, 1965]

- Step 1) flip a fair coin.
- Step 2) if `heads`, answer truthfully. Else, flip another coin and answer that coin flip.

Plausible deniability: impossible to determine true answer with high certainty

Analysis: write: r for true response and $A : \{0,1\} \to \{0,1\}$ for RR algorithm

Given reponses from many users, we can still infer useful statistics:

$$\mathop{\mathbb{E}}_{r\sim p(r)} A(r) = \frac{1}{4} + \frac{1}{2}\Pr(r=1) \quad \to \quad \Pr(r=1) \approx \frac{1}{n}\sum_{i=1}^{n}[2A(r_i) - \frac{1}{2}]$$

Insight: by suitably inserting noise in the procedure, we were able to

- get reasonable quality estimates,
- preserve individual data privacy.

Differential Privacy – a mathematical formalism for ensuring data privacy

Differential Privacy (DP) [C. Dwork, 2006]

A (randomized) learning algorithm A is called ϵ-differentially private, if for all training sets S and S' that differ in only a single element it holds that for any possible output o:

$$\Pr\{A(S) = o\} \leq e^{\epsilon} \Pr\{A(S') = o\}$$

Differential Privacy – a mathematical formalism for ensuring data privacy

Differential Privacy (DP) [C. Dwork, 2006]

A (randomized) learning algorithm A is called ϵ-differentially private, if for all training sets S and S' that differ in only a single element it holds that for any possible output o:

$$\Pr\{A(S) = o\} \leq e^{\epsilon} \Pr\{A(S') = o\}$$

For an ϵ-DP algorithm, can an attacker find out if a certain x was used in training?

Differential Privacy – a mathematical formalism for ensuring data privacy

Differential Privacy (DP) [C. Dwork, 2006]

A (randomized) learning algorithm A is called ϵ-differentially private, if for all training sets S and S' that differ in only a single element it holds that for any possible output o:

$$\Pr\{A(S) = o\} \leq e^{\epsilon} \Pr\{A(S') = o\}$$

For an ϵ-DP algorithm, can an attacker find out if a certain x was used in training?

Let's assume worst case scenario:

- the attacker knows which x to look for
- the attacker knows *all other data* $\mathcal{D}$ that was used in training
- the attacker can run (simulate) A (but does not see A's internal randomness)
- the attacker has unlimited computational resources

What's the highest probability by which the attacker can guess correctly?

Formal setting: "Membership Inference Attack (MIA)"

- unknown quantity: was data point x used or not? $\rightarrow$ random variable $Z \in \{0, 1\}$
  - $Z = 0$: training set was $\mathcal{D}$
  - $Z = 1$: training set was $\mathcal{D} \cup \{x\}$
  - no prior information: $p(Z = 0) = p(Z = 1) = \frac{1}{2}$
- random variable O: output of algorithm, e.g. binary response, or gradient update
- $\mathsf{Pr}(A(S) = o) = \mathsf{Pr}(O = o|Z = 0)$ $\quad$ $\mathsf{Pr}(A(S') = o) = \mathsf{Pr}(O = o|Z = 1)$

Adversay observes one output o:

- Optimal decision: $c^*(o) = \underset{z \in \{0,1\}}{\mathbf{argmax}}\, \mathsf{Pr}(Z = z|O = o)$ $\quad$ "Bayes classifier"
- Best achievable error: $\mathsf{err}(c^*(o)) = \underset{z \in \{0,1\}}{\mathbf{min}}\, \mathsf{Pr}(Z = z|O = o)$ $\quad$ "Bayes error"

How to find $\mathsf{Pr}(Z = z|O = o)$? $\quad$ Bayes rule

$$\frac{\Pr(Z=0|O=o)}{\Pr(Z=1|O=o)} = \frac{\overbrace{\Pr(O=o|Z=0)}^{\Pr(A(S)=o)}\overbrace{\Pr(Z=0)}^{=\frac{1}{2}}}{\underbrace{\Pr(O=o|Z=1)}_{\Pr(A(S')=o)}\underbrace{\Pr(Z=1)}_{=\frac{1}{2}}} = \frac{\Pr(A(S)=o)}{\Pr(A(S')=o)}$$

$$e^{-\epsilon} \leq \quad \frac{\mathsf{Pr}(Z=0|O=o)}{\mathsf{Pr}(Z=1|O=o)} = \frac{\overbrace{\mathsf{Pr}(O=o|Z=0)}^{\mathsf{Pr}(A(S)=o)}\overbrace{\mathsf{Pr}(Z=0)}^{=\frac{1}{2}}}{\underbrace{\mathsf{Pr}(O=o|Z=1)}_{\mathsf{Pr}(A(S')=o)}\underbrace{\mathsf{Pr}(Z=1)}_{=\frac{1}{2}}} = \frac{\mathsf{Pr}(A(S)=o)}{\mathsf{Pr}(A(S')=o)} \quad \leq e^{\epsilon}$$

$$e^{-\epsilon} \leq \quad \frac{\Pr(Z=0|O=o)}{\Pr(Z=1|O=o)} = \frac{\overbrace{\Pr(O=o|Z=0)}^{\Pr(A(S)=o)}\overbrace{\Pr(Z=0)}^{=\frac{1}{2}}}{\underbrace{\Pr(O=o|Z=1)}_{\Pr(A(S')=o)}\underbrace{\Pr(Z=1)}_{=\frac{1}{2}}} = \frac{\Pr(A(S)=o)}{\Pr(A(S')=o)} \quad \leq e^{\epsilon}$$

Write $p_z = \Pr(Z = z|O = o)$. Then, from $e^{-\epsilon} \leq \frac{p_0}{p_1} \leq e^{\epsilon}$, using $p_0 + p_1 = 1$:

$$e^{-\epsilon}(1 - p_z) \leq p_z \leq e^{\epsilon}(1 - p_z)$$

$$e^{-\epsilon} \leq \quad \frac{\Pr(Z=0|O=o)}{\Pr(Z=1|O=o)} = \frac{\overbrace{\Pr(O=o|Z=0)}^{\Pr(A(S)=o)}\overbrace{\Pr(Z=0)}^{=\frac{1}{2}}}{\underbrace{\Pr(O=o|Z=1)}_{\Pr(A(S')=o)}\underbrace{\Pr(Z=1)}_{=\frac{1}{2}}} = \frac{\Pr(A(S)=o)}{\Pr(A(S')=o)} \quad \leq e^{\epsilon}$$

Write $p_z = \Pr(Z=z|O=o)$. Then, from $e^{-\epsilon} \leq \frac{p_0}{p_1} \leq e^{\epsilon}$, using $p_0 + p_1 = 1$:

$$e^{-\epsilon}(1-p_z) \leq p_z \leq e^{\epsilon}(1-p_z) \qquad \Rightarrow \begin{cases} p_z & \leq \frac{1}{1+e^{-\epsilon}} \quad \in [\frac{1}{2}, 1] \\ p_z & \geq \frac{1}{1+e^{\epsilon}} \quad \in [0, \frac{1}{2}] \end{cases}$$

$\rightarrow$ no adversary can predict Z with higher accuracy than $\frac{1}{1+e^{-\epsilon}}$ and error lower than $\frac{1}{1+e^{\epsilon}}$.

$$e^{-\epsilon} \leq \quad \frac{\Pr(Z=0|O=o)}{\Pr(Z=1|O=o)} = \frac{\overbrace{\Pr(O=o|Z=0)}^{\Pr(A(S)=o)}\overbrace{\Pr(Z=0)}^{=\frac{1}{2}}}{\underbrace{\Pr(O=o|Z=1)}_{\Pr(A(S')=o)}\underbrace{\Pr(Z=1)}_{=\frac{1}{2}}} = \frac{\Pr(A(S)=o)}{\Pr(A(S')=o)} \quad \leq e^{\epsilon}$$

Write $p_z = \Pr(Z=z|O=o)$. Then, from $e^{-\epsilon} \leq \frac{p_0}{p_1} \leq e^{\epsilon}$, using $p_0 + p_1 = 1$:

$$e^{-\epsilon}(1-p_z) \leq p_z \leq e^{\epsilon}(1-p_z) \qquad \Rightarrow \begin{cases} p_z & \leq \frac{1}{1+e^{-\epsilon}} \quad \in [\frac{1}{2}, 1] \\ p_z & \geq \frac{1}{1+e^{\epsilon}} \quad \in [0, \frac{1}{2}] \end{cases}$$

$\rightarrow$ no adversary can predict Z with higher accuracy than $\frac{1}{1+e^{-\epsilon}}$ and error lower than $\frac{1}{1+e^{\epsilon}}$.

Numeric examples:

- $\epsilon = 0$: $p_0 = p_1 = \frac{1}{2}$, (o contains no information about x)
- $\epsilon = 1$: $0.27 \approx \frac{1}{1+e} \leq p_z \leq \frac{1}{1+e^{-1}} \approx 0.73$ (plausible deniability)
- $\epsilon = 2$: $0.12 \approx \frac{1}{1+e^2} \leq p_z \leq \frac{1}{1+e^{-2}} \approx 0.88$
- $\epsilon = 8$: $0.0003 \approx \frac{1}{1+e^8} \leq p_z \leq \frac{1}{1+e^{-8}} \approx 0.9997$ (weak protection)
- $\epsilon \rightarrow \infty$: $0 \leq p_z \leq 1$ no guarantees

Caveat: more observation can reduce guarantees

For example, assume algorithm A is run m times, outputs $O_1, \ldots, O_m$

$$\frac{\Pr(Z=0|O_1=o_1,\ldots,O_m=o_m)}{\Pr(Z=1|O_1=o_1,\ldots,O_m=o_m)} = \frac{\prod_{i=1}^m \Pr(O_i=o_i|Z=0)}{\prod_{i=1}^m \Pr(O_i=o_i|Z=1)}$$
$$= \frac{\prod_{i=1}^m \Pr(A(S)=o_i)}{\prod_{i=1}^m \Pr(A(S')=o_i)} \leq \prod_{i=1}^m e^{\epsilon} = e^{m\epsilon}$$

Guarantee gets worse, like $\epsilon \to m\epsilon$.

Caveat: more observation can reduce guarantees

For example, assume algorithm A is run m times, outputs $O_1, \ldots, O_m$

$$e^{-m\epsilon} \leq \frac{\Pr(Z=0|O_1=o_1,\ldots,O_m=o_m)}{\Pr(Z=1|O_1=o_1,\ldots,O_m=o_m)} = \frac{\prod_{i=1}^m \Pr(O_i=o_i|Z=0)}{\prod_{i=1}^m \Pr(O_i=o_i|Z=1)}$$
$$= \frac{\prod_{i=1}^m \Pr(A(S)=o_i)}{\prod_{i=1}^m \Pr(A(S')=o_i)} \leq \prod_{i=1}^m e^{\epsilon} = e^{m\epsilon}$$

Guarantee gets worse, like $\epsilon \to m\epsilon$.

Caveat: algorithm internals must stay internal

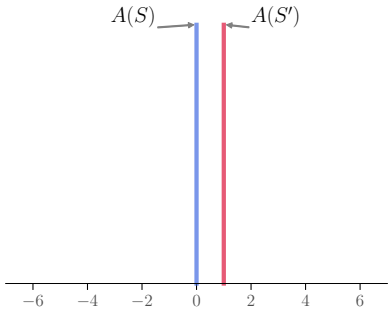
For example, what if random coin flips of randomized response become known?

- if 1st coin flip was `tails` ($b=0$) $\to$ data wasn't used $\to$ answer can be discarded
- if 1st coin flip was `heads` ($b=1$) $\to$ output is exact: 100% leakage

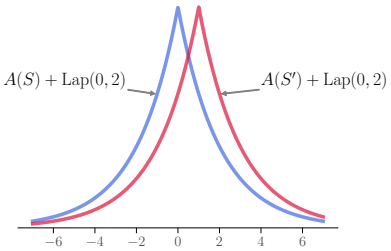
Making ordinary algorithms differentially private

- to be private, the algorithm output cannot be deterministic
- simplest solution: take standard algorithm and add suitably tailored noise

Illustration by probability density of outcomes:



no privacy



potentially private

Source: `https://commons.wikimedia.org/wiki/File:Laplace_mechanism.png`

Example: "Laplace mechanism". For $A : \mathcal{X} \to \mathbb{R}^d$ add *Laplacian noise* to algorithm output

$$A^{\mathsf{dp}}(S) := A(S) + (Z_1, \ldots, Z_d) \qquad \text{with } Z_i \sim \mathsf{Lap}(0, b).$$

where $\mathsf{Lap}(z; \mu, b) = \frac{1}{2b} e^{-\frac{\|\mu - z\|_1}{b}}$ is the Laplace distribution.

$$\frac{\mathsf{Pr}(A^{\mathsf{dp}}(S) = o)}{\mathsf{Pr}(A^{\mathsf{dp}}(S') = o)} = \frac{\mathsf{Lap}(o; \mu = A(S), b = \beta)}{\mathsf{Lap}(o; \mu = A(S'), b = \beta)} = e^{\frac{-\|x - A(S)\|_1 + \|x - A(S')\|_1}{\beta}} \leq e^{\frac{\|A(S) - A(S')\|_1}{\beta}}$$

Example: "Laplace mechanism". For $A : \mathcal{X} \to \mathbb{R}^d$ add *Laplacian noise* to algorithm output

$$A^{\mathsf{dp}}(S) := A(S) + (Z_1, \ldots, Z_d) \qquad \text{with } Z_i \sim \mathsf{Lap}(0, b).$$

where $\mathsf{Lap}(z; \mu, b) = \frac{1}{2b} e^{-\frac{\|\mu - z\|_1}{b}}$ is the Laplace distribution.

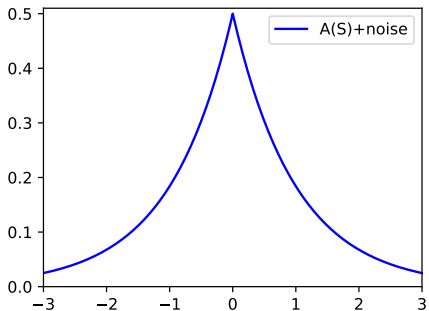
$$\frac{\Pr(A^{\mathsf{dp}}(S) = o)}{\Pr(A^{\mathsf{dp}}(S') = o)} = \frac{\mathsf{Lap}(o; \mu = A(S), b = \beta)}{\mathsf{Lap}(o; \mu = A(S'), b = \beta)} = e^{\frac{-\|x - A(S)\|_1 + \|x - A(S')\|_1}{\beta}} \leq e^{\frac{\|A(S) - A(S')\|_1}{\beta}}$$

- Laplace mechanism is ϵ-dp, if $\beta \geq \frac{1}{\epsilon} \overbrace{\sup_{S,S'} \|A(S) - A(S')\|_1}^{=:\Delta_1(A) \text{ "sensitivity"}}$
- works best if $\sup_{S,S'} \|A(S) - A(S')\|_1$ is small (large β hurts accuracy)
- Example: $A(S) = \frac{1}{|S|} \sum_{x_i \in S} x_i$ $\Delta_1(A) = \sup_x \|x\|_1$ $\leftarrow$ works, if x is bounded
- Example: $A(S) =$ neural network training, what's $\Delta_1(A)$?

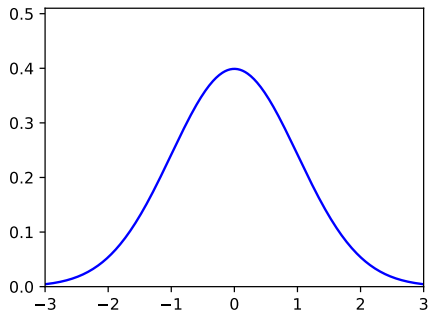
Problem: L^1-norm scales linearly with the dimensionity of x

- for growing dimension, linearly more noise *per-component* is needed for privacy!

Idea: can we use other noise, which has lighter tails? E.g. Gaussian?



Laplacian noise

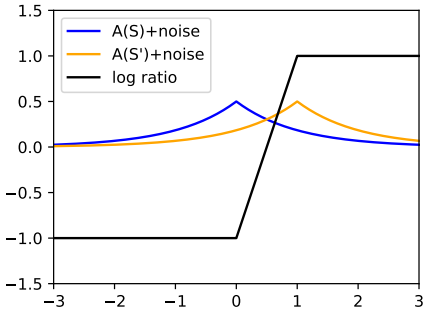


Gaussian noise

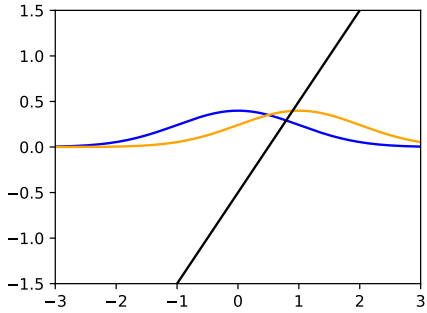
Problem: L^1-norm scales linearly with the dimensionity of x

- for growing dimension, linearly more noise *per-component* is needed for privacy!

Idea: can we use other noise, which has lighter tails? E.g. Gaussian?



Laplacian noise



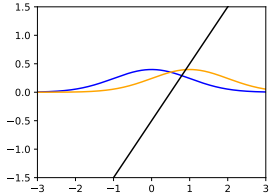
Gaussian noise

Problem:

- ratio $\frac{\Pr\{A(S)=o\}}{\Pr\{A(S')=o\}}$ is not bounded, differential privacy criterion not fulfilled

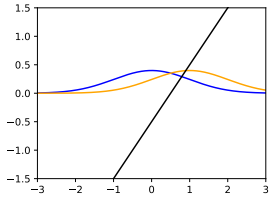
Observation:

- worst violations happen in low-probability regime
- for *most* of the probability mass, ratio is bounded



Observation:

- worst violations happen in low-probability regime
- for *most* of the probability mass, ratio is bounded



Approximate (aka (ϵ, δ)-) Differential Privacy (DP) [C. Dwork, 2006]

For $\epsilon \geq 0$ and $\delta \in [0, 1]$, a (randomized) learning algorithm $A : \mathbb{P}(X) \to \mathcal{Y}$ is called (ϵ, δ)-differentially private, if for all training sets S and S' that differ in only a single element it holds that for any possible subset of outputs, $O \subset \mathcal{Y}$:

$$\Pr\{A(S) \in O\} \leq e^{\epsilon} \Pr\{A(S') \in O\} + \delta$$

(like ordinary DP, but allow for a small probability that ratio condition does not hold)

Example: "Gaussian mechanism": add *Gaussian noise* to result of computation

$$A^{\mathsf{dp}}(S) := A(S) + \mathcal{N}(0, \sigma^2)$$

Theorem

Let $\Delta_2(A) = \mathbf{sup}_{S \sim S'} \|A(S) - A(S')\|_2$ *the sensitivity of* A *w.r.t. the Euclidean norm. For any* $\epsilon \leq 1$ *and* $\delta \in (0,1)$*, if we set* $\sigma = \sqrt{2\log(1.25/\delta)}\frac{\Delta_2(A)}{\epsilon}$*, then* A^{dp} *will be* (ϵ, δ)*-dp. For* $\epsilon > 1$*, there is a more complicated formula.*

Example: $A(S) = \frac{1}{|S|}\sum_{x_i \in S} x_i \qquad \Delta_2(A) = \mathbf{sup}_{x \in \mathcal{X}} \|x\|_2$

- $\|x\|_2$ grows like square-root of dimensions $\rightarrow$ less noise per component than for Laplace

Theorem

Let $\Delta_2(A) = \mathbf{sup}_{S \sim S'} \|A(S) - A(S')\|_2$ the sensitivity of A w.r.t. the Euclidean norm. For any $\epsilon \leq 1$ and $\delta \in (0,1)$, if we set $\sigma = \sqrt{2\log(1.25/\delta)}\frac{\Delta_2(A)}{\epsilon}$, then A^{dp} will be (ϵ, δ)-dp. For $\epsilon > 1$, there is a more complicated formula.

Example: $A(S) = \frac{1}{|S|}\sum_{x_i \in S} x_i \qquad \Delta_2(A) = \mathbf{sup}_{x \in \mathcal{X}} \|x\|_2$

- $\|x\|_2$ grows like square-root of dimensions $\rightarrow$ less noise per component than for Laplace

Caveat: δ must be very small! Otherwise, trivial examples, e.g.

- $|S| = (x_1, \ldots, x_n)$: $A(S) = x_i$ for $i \sim \mathsf{Unif}(\{1, \ldots, n\})$ is $(0, 1/n)$-dp

In practice, use at least $\delta \approx n^{-1.1}$ or similar.

Postprocessing

Let $A : \mathbb{P}(\mathcal{X}) \to \mathcal{Y}$ be (ϵ, δ)-differentially private, and let $F : \mathcal{Y} \to \mathcal{Z}$ be *any* mapping. Then $F \circ A$ is also (ϵ, δ)-differentially private. $\to$ postprocessing cannot undo the privatization.

Composition

Let $A : \mathbb{P}(\mathcal{X}) \to \mathcal{Y}$ be (ϵ, δ)-dp, and $B : \mathbb{P}(\mathcal{X}) \to \mathcal{Z}$ be (ϵ', δ')-dp.
Then $A \times B : \mathbb{P}(\mathcal{X}) \to \mathcal{Y} \times \mathcal{Z}$ is $(\epsilon + \epsilon', \delta + \delta')$-dp ("composition" property).

This holds even if the choice of B depends on the output of A ("adaptive composition").
Actually, even better guarantees possible when $\delta, \delta' > 0$.

Parallel Composition

If A and B as above operate on disjoint parts of the data, then $A \times B : \mathbb{P}(\mathcal{X}) \to \mathcal{Y} \times \mathcal{Z}$ is $(\mathbf{max}\{\epsilon, \epsilon'\}, \mathbf{max}\{\delta, \delta'\})$-dp (parallel composition).

Private Learning

How to privately learning deep networks?

Output perturbation?

Idea: add Gaussian noise to parameters of trained model

Question: how much noise is needed, what's the *sensitivity*?

- Small ConvNet trained CIFAR-10 dataset:
  - number of parameters: 272,474, value range $(-\infty, \infty)$
  - changing one training example might completely change parameters
- ballpark heuristic
  - Euclidean norm of example network's parameters: ≈ 22.4
  - let's say another network would have same norm: $\Delta_2 \approx 2 \cdot 22.4$
  - required noise strength for $(\epsilon = 1, \delta = 10^{-5})$-dp: $\sigma \approx 168$

before	-0.01	-0.01	0.04	0.01	0.01	-0.02	0.02	-0.02	0.01	-0.03
after	213.1	56.3	-94.6	-42.3	153.8	-111.3	182.9	-113.2	-5.3	-116.0

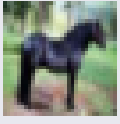
$\rightarrow$ network quality completely destroyed!

Input perturbation?

Idea: add Gaussian noise to input data, training is just "postprocessing"

Question: how much noise is needed, what's the *sensitivity*?

- CIFAR-10 dataset:
  - data dimension: 3072 ($32 \times 32 \times 3$), value range $[0, 1]$
- changing one image to another might completely change all pixel values
  - $\Delta_2 = 55$
- required noise strength for $(\epsilon = 1, \delta = 10^{-5})$-dp: $\sigma \approx 207$

before:  after: 

$\rightarrow$ training is going to fail

Let's take a closer look at SGD:

input learning rate η, batch size b,

1: $\theta \leftarrow$ `init`
2: **for** $t = 1, \ldots, T$ **do**
3: pick batch S of size b
4: **for** $i = 1, \ldots, b$ **do**
5: compute gradients $g_i \leftarrow \nabla \text{loss}(x_i; \theta_{t-1})$
6: **end for**
7: compute update $u_t \leftarrow \frac{1}{b} \sum_{i=1}^{b} g_i$
8: update model $\theta_t \leftarrow \theta_{t-1} - \eta u_t$
9: **end for**

Observation: final model is sum of reasonably simple per-step contributions

Idea: privatize $u_1, \ldots, u_T$, model update step is postprocessing

- noise added to u_t could be corrected for in later steps

Private model training: DP-SGD [M. Abadi *et al.*, "Deep Learning with Differential Privacy", SIGSAC 2016]

input learning rate η, batch size b, clip norm ζ, noise strength σ

$\theta \leftarrow$ `init`
for $t = 1, \dots, T$ **do**
  pick batch S of size b
  for $i = 1, \dots, b$ **do**
    compute gradients $g_i \leftarrow \nabla \text{loss}(x_i; \theta_{t-1})$
    if $\|g_i\| > \zeta$ **then** $g_i \leftarrow \frac{\zeta}{\|g_i\|} g_i$ // gradient "clipping", ensures bounded sensitivity
  end for
  compute update $u_t \leftarrow \frac{1}{b} \sum_{i=1}^{b} g_i + \mathcal{N}(\cdot, 0, \sigma \text{Id})$ // add Gaussian noise
  update model $\theta_t \leftarrow \theta_{t-1} - \eta u_t$
end for

- each update u_t is made private $\rightarrow$ each θ is private by postprocessing
  - not just applicable to SGD, update could use momentum, weight decay, Adam, ...
- all intermediate models are private, not just final output
- if each iteration is (ϵ, δ)-dp, then overall training is at least $(T\epsilon, T\delta)$-dp ("composition")

Better guarantees for DP-SGD: Amplification

Amplification techniques yield higher privacy than the Gaussian mechanisms alone:

- amplification by subsampling e.g. [Balle *et al.*, NeurIPS 2018]: in each update step, use only a subset of the data (mini-batch) $\rightarrow$ the influence of each sample becomes (much) smaller
- amplification by iteration e.g. [Feldman *et al.*, FOCS 2018]: keep adding noise (and potentially doing other operations) but do *not* use the private data anymore $\rightarrow$ privacy *increases* with more such iterations
- amplification by shuffling e.g. [Erlingsson *et al.*, SODA 2019]: in setup with several participating users (e.g. federated learning), shuffle gradients before aggregation $\rightarrow$ increased privacy because server does not know which user contributed what

Powerful, if random access to data is possible, but can be hard to achieve otherwise.

Private Learning: Ongoing Research

Let's rephrase SGD:

- $\theta_0 \leftarrow$ initialization, let's say 0.
- $\theta_{t+1} = \theta_t - \eta u_t \quad$ with $\quad u_t = \dfrac{1}{|S_t|} \displaystyle\sum_{(x,y)\in S_t} \mathsf{clip}(\nabla_\theta \ell(x,\theta), \zeta)$

Unroll the recursion:

$$\theta_0 = 0 \qquad \theta_1 = -\eta u_1, \qquad \theta_2 = -\eta u_1 - \eta u_2, \qquad \theta_3 = -\eta u_1 - \eta u_2 - \eta u_3, \quad \text{etc..}$$
$$\theta_t = -\eta(u_1 + u_2 + \cdots + u_t)$$

(caveat: u_t can only be computed once θ_{t-1} is available, we can ignore this here)

Example

Let's study this systematically:

- private data $x_1, \dots, x_n \in \mathbb{R}^d$, with $\|x_i\| \leq \zeta$, stack into matrix $X \in \mathbb{R}^{n \times d}$:
- neighboring datasets differ in exactly one entry:

$$X \sim X' \quad \iff \quad X - X' = \alpha e_i, \qquad \text{with } |\alpha| \leq 1 \text{ for some } i \in \{1, 2, \dots, n\}$$

- we want to compute the cumulative sum

$$O(X) = \begin{pmatrix} x_1 \\ x_1 + x_2 \\ x_1 + x_2 + x_3 \\ \vdots \end{pmatrix} \quad \text{i.e.} \quad O(X) = AX, \quad \text{with} \quad A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

For simplicity, let $n = 2$, $d = 1$, $\zeta = 0.5$, and we pick concrete values:

$$X = \begin{pmatrix} 0.2 \\ -0.3 \end{pmatrix}, \quad O(X) = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0.2 \\ -0.3 \end{pmatrix} = \begin{pmatrix} 0.2 \\ -0.1 \end{pmatrix}$$

Goal: for $A = X = \begin{pmatrix} 0.2 \\ -0.3 \end{pmatrix}$ and $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$, privately compute $O = AX = \begin{pmatrix} 0.2 \\ -0.1 \end{pmatrix}$.

Idea 1: Gaussian mechanism on the complete output. How much noise is needed?

- For neighboring datasets: $X - X' = \alpha e_i, \qquad |\alpha| \leq 1$, with $i \in \{1, 2\}$.
- Hence: $O(X) - O(X') = AX - AX' = A(X - X') = \alpha A e_i$.
- L^2-sensitivity:

$$\Delta_2(A) = \sup_{X \sim X'} \|O(X) - O(X')\|_2 = \max_{\substack{i \in \{1,2\} \\ |\alpha| \leq 1}} \|\alpha A e_i\|_2$$

$$= \mathbf{max} \left\{ \left\| \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\|_2, \left\| \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\|_2 \right\} = \mathbf{max}\{\sqrt{2}, 1\} = \sqrt{2}.$$

Method 1:

$$\widetilde{O}(X) = AX + \sqrt{2}\begin{pmatrix} n_1 \\ n_2 \end{pmatrix}, \qquad n_i \sim \mathcal{N}(0, \sigma_{\epsilon,\delta}), \quad \text{(exact constants suppressed)}$$

$$= \begin{pmatrix} 0.2 \\ -0.1 \end{pmatrix} + \begin{pmatrix} 0.192 \\ 0.044 \end{pmatrix} = \begin{pmatrix} 0.392 \\ -0.056 \end{pmatrix}$$

Goal: for $A = X = \left(\begin{smallmatrix} 0.2 \\ -0.3 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$, privately compute $O = AX = \left(\begin{smallmatrix} 0.2 \\ -0.1 \end{smallmatrix}\right)$.

Idea 2: exploit composition property. Privatize each output, then release all of them.

- Output O_1 has sensitivity 1:
  - $O_1(X) = x, \quad O_1(X') = \begin{cases} x' \\ x \end{cases}, \quad \|O_1(X) - O_1(X')\| = \begin{cases} \|x - x'\| \le 1 \\ \|x - x\| = 0 \le 1 \end{cases}$
  - $\widetilde{O}_1(X) = x + n_1, \qquad n_1 \sim \mathcal{N}(0, 1 \cdot \sigma_{\epsilon,\delta})$.
- Output O_2 has sensitivity 1:
  - $O_2(X) = x + y, \quad O_2(X') = \begin{cases} x' + y \\ x + y' \end{cases}, \quad \|O_2(X) - O_2(X')\| = \begin{cases} \|x - x'\| \le 1 \\ \|y - y'\| \le 1 \end{cases}$
  - $\widetilde{O}_2(X) = x + y + n_2, \qquad n_2 \sim \mathcal{N}(0, 1 \cdot \sigma_{\epsilon,\delta})$.
- Then, we release both privatized estimates $\widetilde{O}(X) = \left(\begin{smallmatrix} x+n_1 \\ x+y+n_2 \end{smallmatrix}\right), \qquad n_i \sim \mathcal{N}(0, \sigma_{\epsilon,\delta})$.

Goal: for $A = X = \left(\begin{smallmatrix} 0.2 \\ -0.3 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$, privately compute $O = AX = \left(\begin{smallmatrix} 0.2 \\ -0.1 \end{smallmatrix}\right)$.

Idea 2: exploit composition property. Privatize each output, then release all of them.

- Output O_1 has sensitivity 1:
  - $O_1(X) = x, \quad O_1(X') = \begin{cases} x' \\ x \end{cases}, \quad \|O_1(X) - O_1(X')\| = \begin{cases} \|x - x'\| \le 1 \\ \|x - x\| = 0 \le 1 \end{cases}$
  - $\widetilde{O}_1(X) = x + n_1, \qquad n_1 \sim \mathcal{N}(0, 1 \cdot \sigma_{\epsilon,\delta})$.
- Output O_2 has sensitivity 1:
  - $O_2(X) = x + y, \quad O_2(X') = \begin{cases} x' + y \\ x + y' \end{cases}, \quad \|O_2(X) - O_2(X')\| = \begin{cases} \|x - x'\| \le 1 \\ \|y - y'\| \le 1 \end{cases}$
  - $\widetilde{O}_2(X) = x + y + n_2, \qquad n_2 \sim \mathcal{N}(0, 1 \cdot \sigma_{\epsilon,\delta})$.
- Then, we release both privatized estimates $\widetilde{O}(X) = \left(\begin{smallmatrix} x+n_1 \\ x+y+n_2 \end{smallmatrix}\right), \qquad n_i \sim \mathcal{N}(0, \sigma_{\epsilon,\delta})$.

Warning! We have accessed the same private data twice.

If each release is (ϵ, δ)-DP, then the overall release is only $(2\epsilon, 2\delta)$-DP (composition property).
$\rightarrow$ To get overall (ϵ, δ)-DP, each release needs $2\times$ stronger noise.

Goal: for $A = X = \left(\begin{smallmatrix} 0.2 \\ -0.3 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$, privately compute $O = AX = \left(\begin{smallmatrix} 0.2 \\ -0.1 \end{smallmatrix}\right)$.

Method 2: Gaussian mechanism separately for each output, accounting for composition.

$$\widetilde{O}(X) = \begin{pmatrix} x+2n_1 \\ x+y+2n_2 \end{pmatrix}, \qquad n_i \sim \mathcal{N}(0, \sigma_{\epsilon,\delta}).$$

For our example

$$X = \left(\begin{smallmatrix} 0.2 \\ -0.3 \end{smallmatrix}\right), \qquad AX = \left(\begin{smallmatrix} 0.2 \\ -0.1 \end{smallmatrix}\right).$$

$$\widetilde{O}(X) = \left(\begin{smallmatrix} 0.2 \\ -0.1 \end{smallmatrix}\right) + \begin{pmatrix} 2n_1 \\ 2n_2 \end{pmatrix} = \left(\begin{smallmatrix} 0.472 \\ -0.038 \end{smallmatrix}\right)$$

Each releases is $(\epsilon/2, \delta/2)$-DP, so the full vector release is (ϵ, δ)-DP.

$\rightarrow$ but: strictly more noise needed than in Method 1.

Goal: for $A = X = \left(\begin{smallmatrix} 0.2 \\ -0.3 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$, privately compute $O = AX = \left(\begin{smallmatrix} 0.2 \\ -0.1 \end{smallmatrix}\right)$.

Idea 3: exploit postprocessing property: if $\tilde{X}$ is private, any $A\tilde{X}$ is also private

Method 3: privatize the data items first, then combine them:

- For neighboring datasets, $X - X' = \alpha e_i$ with $|\alpha| \leq 1$. Sensitivity: $\Delta_2(X) = 1$
- Therefore $\widetilde{X} = X + \left(\begin{smallmatrix} n_1 \\ n_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 0.2+n_1 \\ -0.3+n_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 0.336 \\ -0.269 \end{smallmatrix}\right)$ is (ϵ, δ)-DP.

Then postprocess:

$$\widetilde{O}(X) = A\widetilde{X} = \left(\begin{smallmatrix} \tilde{x} \\ \tilde{x}+\tilde{y} \end{smallmatrix}\right) = \left(\begin{smallmatrix} x+n_1 \\ x+y+n_1+n_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} 0.336 \\ -0.033 \end{smallmatrix}\right)$$

Alternative view:

$$= A(X + \left(\begin{smallmatrix} n_1 \\ n_2 \end{smallmatrix}\right)) = AX + A\left(\begin{smallmatrix} n_1 \\ n_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} x \\ x+y \end{smallmatrix}\right) + \left(\begin{smallmatrix} n_1 \\ n_1+n_2 \end{smallmatrix}\right)$$

$\rightarrow$ different amounts of noise per component; noise between components is *dependent*.

Which method is best?

- all estimates are (ϵ, δ)-DP
- all estimates are unbiased: $\mathbb{E}_{\text{noise}}[\widetilde{O}(X)] = O(X)$
- only difference: how much noise was added? $\mathcal{E}^2 := \mathop{\mathbb{E}}_{\text{noise}}\left[\|\widetilde{O}(X) - O(X)\|_2^2\right].$
- **Method 1:** $\mathcal{E}_1^2 = \mathbb{E}\left\|\begin{pmatrix}\sqrt{2}n_1\\ \sqrt{2}n_2\end{pmatrix}\right\|_2^2 = 2 + 2 = 4.$
- **Method 2:** $\mathcal{E}_2^2 = \mathbb{E}\left\|\begin{pmatrix}2n_1\\ 2n_2\end{pmatrix}\right\|_2^2 = 4 + 4 = 8.$
- **Method 3:** $\mathcal{E}_3^2 = \mathbb{E}\,\|\begin{pmatrix}n_1\\ n_1+n_2\end{pmatrix}\|_2^2 = 1 + 2 = 3.$

Can we do even better? How can we study this systematically?

Goal: for $A = X = \left(\begin{smallmatrix} 0.2 \\ -0.3 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$, privately compute $O = AX = \left(\begin{smallmatrix} 0.2 \\ -0.1 \end{smallmatrix}\right)$.

Idea, inspired by Method 3:
Instead of privatizing the final output AX, privatize an earlier computation.

Observation: Assume there were two matrices, $B, C \in \mathbb{R}^{2\times 2}$, with $BC = A$.

Then, obviously, $O(X) = AX = BCX$.

We can privatize $CX \mapsto \widetilde{CX}$ and release $B\widetilde{CX}$.

Method 4: choose any factorization $A = BC$.

- privatize the intermediate result:
$$\widetilde{Q} = CX + \Delta_2(C)\left(\begin{smallmatrix} n_1 \\ n_2 \end{smallmatrix}\right), \qquad \Delta_2(C) \text{ is sensitivity of } X \mapsto CX$$
- construct desired output by postprocessing:
$$\widetilde{O} = B\widetilde{Q} = AX + \Delta_2(C)B\left(\begin{smallmatrix} n_1 \\ n_2 \end{smallmatrix}\right)$$

Do such factorizations exist? Of course, e.g., $B = \mathsf{Id}, C = A$, or $B = A, C = \mathsf{Id}$, or use SVD.

For $A = BC$, what's the effective squared error on AX?

Observation: $\Delta_2(C) = \|C\|_{1,2} := \mathbf{max}_i \, \|C[:, i]\|_2$ largest column norm

$$\mathcal{E}^2(B, C) = \mathbb{E}\left\| \|C\|_{1,2} B\left(\begin{smallmatrix} n_1 \\ n_2 \end{smallmatrix}\right) \right\|_2^2 = \|B\|_F^2 \|C\|_{1,2}^2, \quad \text{with} \quad \|B\|_F = \sqrt{\sum_{i,j} B[i,j]^2}$$

Tradeoff:

sensitivity of C vs. noise amplification by B

Examples:

- Method 1: Gaussian mechanism on AX, $\quad B = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right), \quad C = A = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right).$

$$\|B\|_F^2 = 2, \qquad \|C\|_{1,2}^2 = 2, \qquad \mathcal{E}^2(B, C) = \|B\|_F^2 \|C\|_{1,2}^2 = 2 \cdot 2 = 4.$$

- Method 3: Gaussian mechanism on X, $\quad B = A = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right), \quad C = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$

$$\|B\|_F^2 = 3, \quad \|C\|_{1,2}^2 = 1, \quad \mathcal{E}^2(B, C) = 3 \cdot 1 = 3.$$

- Square-Root Factorization [Henzinger *et al.*, SODA 2024]. Take $B = C = A^{1/2} = \left(\begin{smallmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{smallmatrix}\right)$, Check: $BC = A^{1/2} A^{1/2} = A$.

$$\|B\|_F^2 = 2.25, \quad \|C\|_{1,2}^2 = 1.25, \quad \mathcal{E}^2(B, C) = 2.25 \cdot 1.25 = 2.8125.$$

- Numeric Optimization [Choquette-Choo *et al.*, NeurIPS 2023]: $\min_{\substack{B,C:\, BC=A \\ B,C \text{ lower triangular}}} \mathcal{E}^2(B, C) = 2.618.$

Back to SGD:

- $\theta_0 \leftarrow$ initialization, e.g. 0
- $\theta_{t+1} = \theta_t - \eta g_t \quad$ with $\quad u_t = \sum_{x \in S} \text{clip}(\nabla_\theta \ell(x, \theta_t))$

Unroll the recursion: $\theta_1 = -\eta u_1, \qquad \theta_2 = -\eta u_1 - \eta u_2, \qquad \theta_3 = -\eta u_1 - \eta u_2 - \eta u_3, \quad$ etc..

In matrix form: $\Theta = -\eta A X$, with $\Theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{T-1} \\ \theta_T \end{pmatrix}$, $X = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{T-1} \\ u_T \end{pmatrix}$ and $A = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 & 0 \\ 1 & 1 & \cdots & 1 & 1 \end{pmatrix}$

Caveat: written globally, but update u_t can only be computed once θ_t is available

Private model training with Matrix Factorization

input learning rate η, batch size b, clip norm ζ, noise strength σ, matrices B and C

$\theta \leftarrow$ `init`
for $t = 1, \ldots, T$ **do**
  pick batch S of size b
  for $i = 1, \ldots, b$ **do**
    compute gradients $g_i \leftarrow \nabla \text{loss}(x_i; \theta)$
    if $\|g_i\| > \zeta$ **then** $g_i \leftarrow \frac{\zeta}{\|g_i\|} g_i$ // gradient "clipping"
  end for
  compute update $u_t \leftarrow \frac{1}{b} \sum_{i=1}^{b} g_i$
  privatize intermediate $o_t \leftarrow \sum_{s=1}^{t} C[t,s] u_s + \|C\|_{1,2} \mathcal{N}(\cdot, 0, \sigma \text{Id})$
  update model $\theta_t = \sum_{s=1}^{t} B[t,s] o_s$
end for

- each model θ_t is private, and less noise than naive DP-SGD
- but: inefficient, must store B, C, and keep all old update vectors and models around

Observation: $\tilde{\Theta} = -\eta B(CX + Z) = -\eta A(X + C^{-1}Z)$ $\quad \leftarrow$ DP-SGD with dependent noise

Private model training with Matrix Factorization: MF-DP-SGD

input learning rate η, batch size b, clip norm ζ, noise strength σ, matrix C

$\theta \leftarrow$ `init`
for $t = 1, \ldots, T$ **do**
  pick batch S of size b
  for $i = 1, \ldots, b$ **do**
    compute gradients $g_i \leftarrow \nabla \text{loss}(x_i; \theta)$
    if $\|g_i\| > \zeta$ **then** $g_i \leftarrow \frac{\zeta}{\|g_i\|} g_i$ // gradient "clipping"
  end for
  create noise $z_t \sim \mathcal{N}(\cdot, 0, \sigma \text{Id})$
  compute private update $u_t \leftarrow \frac{1}{b} \sum_{i=1}^{b} g_i + \sum_{s=1}^{t} C^{-1}[t, s] z_s$
  update model $\theta_t \leftarrow \theta_{t-1} - \eta u_t$
end for

- no need to store old updates and models, must store C and keep old noise $z_1, \ldots, z_T$
  - mitigation 1: choose C such that make C^{-1} is *banded* [Kalinin *et al.*, ICLR 2026]
  - mitigation 2: *regenerate* noise on-the-fly from seeds [Kalinin *et al.*, https://arxiv.org/abs/2601.22334]

Why does it work?

$$C = A^{1/2} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{3}{8} & \frac{1}{2} & 1 \end{pmatrix} \qquad C^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{8} & -\frac{1}{2} & 1 \end{pmatrix} \qquad \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} \sim \mathcal{N}(0, 1.18)$$

$$X + C^{-1}Z = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} + \begin{pmatrix} z_1 \\ -\frac{1}{2}z_1 + z_2 \\ -\frac{1}{8}z_1 - \frac{1}{2}z_2 + z_3 \end{pmatrix} \qquad \leftarrow \text{correlated noise}$$

$$A(X + C^{-1}Z) = \begin{pmatrix} u_1 \\ u_1 + u_2 \\ u_1 + u_2 + u_3 \end{pmatrix} + \begin{pmatrix} z_1 \\ \frac{1}{2}z_1 + z_2 \\ \frac{3}{8}z_1 + \frac{1}{2}z_2 + z_3 \end{pmatrix}$$

Observation:

- slightly larger variance for Gaussian noise itself
- some of the noise added in early iterations is subtracted again later
- variants: multi-epoch training, momentum, weight-decay, varying learning rate, ...

Exemplary Results

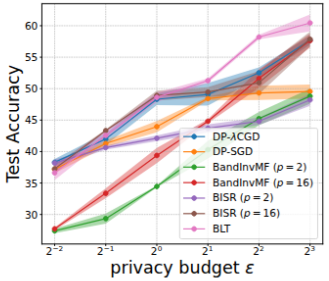
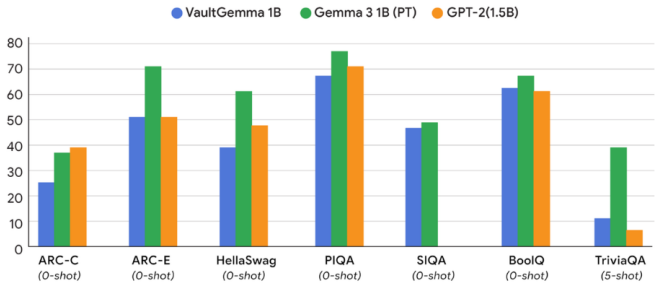


Figure 4: Test accuracy on CIFAR-10 ($B = 128$, $k = 10$; error bars based on three runs).



Performance comparison of VaultGemma 1B (differentially private) against its non-private counterpart (Gemma3 1B) and an older baseline (GPT-2 1.5B). The results quantify the current resource investment required for privacy and demonstrate that modern DP training yields utility comparable to non-private models from roughly five years ago.

Source: Kalinin *et al.*, https://arxiv.org/abs/2601.22334 https://research.google/blog/vaultgemma-the-worlds-most-capable-differentially-private-llm/

Specific Factorizations [with J. Anderson, M. Henzinger, A. Honkela, N. Kalinin, R. McKenna, R. Pagh, V. Roth, A. Sanyal, J. Upadhyay...]

- For different ML-relevant matrices A, construct explicit factorizations $A = BC$ that
  - (provably) have high utility,
  - are memory- and compute-efficient.

Differential Privacy for Generalization [with M. Cairney-Leeming, H. Zakerinia]

- Study connections between differential privacy and generalization guarantees, e.g. PAC-Bayesian.

Open problems: connect theory to real-world algorithms, such as (MF)-DP-SGD.

A lot of the data out there is private/protected.

- personal communications, internal notes, browsing history, ...
- medical/genetic, military, business secrets, ...

Some should stay locked away, but others could make machine learning models truly better.

Differential Privacy is a mature research field in (Theoretical) Computer Science

- 1) limit the influence of individual data items,
- 2) add noise to the processing to hide which data was used.

Private machine learning still has a privacy/utility gap

- enforcing differential privacy makes models worse, which hinders adoption
- do we need relaxed notions of privacy? hybrids?

slides