Privacy Enhancing Technologies FS2025
Lecture 29-30 – Differentially Private Learning

Florian Tramèr

AGENDA

1. Formalizing (non-private) learning
2. Private learning in the convex setting
3. Private deep learning

Recap

In the past lectures, we looked at differentially private computations of simple statistics such as means, where we considered the non-noisy answer computed over the dataset as the "ground truth". But in many settings, this isn't actually what we care about.

For example, if we want to know if smoking causes cancer, we don't care just about the correlation between these variables in some fixed dataset of patients. Instead, we want to know if smoking causes cancer *for the general population*. That is, we want to build some predictive model that will *generalize* to new data.

For simple statistics such as means, this generalization is quite straightforward: if we sample a dataset D of size n i.i.d.[1] from some underlying distribution $\mathcal{P}$, then the empirical mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ differs from the true mean $\mu = \mathbb{E}_{\mathcal{P}}[X]$ by a factor $O(1/\sqrt{n})$ in expectation. Note that this "sampling error" is asymptotically *larger* than the noise we need to add to ensure differential privacy of the mean (recall that this was $O(1/n)$). So we can effectively get privacy "*for free*" (asymptotically at least).

What if we want to build more complex models of some distribution $\mathcal{P}$? This is the setting of *machine learning*.

1 Machine Learning and Empirical Risk Minimization

We formalize the problem of learning a model from data as follows. We have some distribution $\mathcal{P}$ over *inputs* $\mathcal{X}$ with *labels* $\mathcal{Y}$. We want to learn a model $f_\theta : \mathcal{X} \to \mathcal{Y}$ that will generalize to new data. Here, you can think of $\theta \in \mathbb{R}^d$ as the parameters of the model, e.g., the weights of a linear model $w^\top x + b$, or of a more complex neural network.

We formalize the notion of "generalization" via a *loss function* $\ell(\theta, x, y)$ that measures the error of a model f_θ on a pair (x, y). As an example, you can think of ℓ as the standard classification loss, $\ell(\theta, x, y) = \mathbb{1}\{f_\theta(x) \neq y\} \in \{0, 1\}$. We then define the *risk* $\mathcal{L}(\theta)$ of a model as the expected loss over the distribution $\mathcal{P}$:

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{P}}[\ell(\theta, x, y)] \,. \tag{1}$$

[1]i.i.d. stands for independent and identically distributed. This means that each sample x_i is drawn independently from the same distribution $\mathcal{P}$.

Assume we sample i.i.d. pairs $(x, y) \sim \mathcal{P}$ from the distribution to build a *training set* $D = \{(x_i, y_i)\}_{i=1}^n$ of size n. The *empirical risk* $\widehat{\mathcal{L}}_D(\theta)$ of the model on the dataset D is then the average loss over the training set:

$$\widehat{\mathcal{L}}_D(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(\theta, x_i, y_i) \tag{2}$$

A common approach to learning a model is to minimize the empirical risk, i.e., to find parameters $\theta \in \Theta$ that minimize $\widehat{\mathcal{L}}_D(\theta)$. This is known as *empirical risk minimization* (ERM).

$$\theta^* = \arg\min_{\theta \in \Theta} \widehat{\mathcal{L}}_D(\theta) \tag{3}$$

Given a model θ^* optimized over the training set D, we can then evaluate its risk on the underlying distribution $\mathcal{P}$ (typically by sampling a new *test* dataset i.i.d. from $\mathcal{P}$). The risk computed over the training set will usually underestimate the true population risk. This is known as *overfitting*, and measured by the *generalization gap*:

$$|\mathcal{L}(\theta^*) - \widehat{\mathcal{L}}_D(\theta^*)| \tag{4}$$

A standard way to minimize overfitting is to use a *regularizer* that penalizes complex models, such as an ℓ_2 regularizer that adds a term $\|\theta\|_2^2$ to the loss.

2 Why Private Learning?

As we saw previously, when computing (many) simple statistics such as sums, it is possible to blatantly reconstruct large parts of the data. While machine learning models are obviously much more complex, similar risks can arise, since the parameters of the model ultimately depend on the training data.

Sometimes, this is very explicit: for example, a Support Vector Machine (SVM) is defined by its *support vectors*, which are individual data points. So the parameters of such a model explicitly leak some training data. Deep neural networks are much more complicated, but still leak very clear information about their training data. For example, as we'll see in a future lecture, it is often possible to infer whether a particular data point was used to train a model or not [SSSS17]—a violation of differential privacy. Worse, deep neural networks that *generate* data (e.g., ChatGPT or your favorite AI image generator) sometimes leak entire training samples [CTW+21]. See e.g., the image in Figure 1 that we generated using Stable Diffusion.

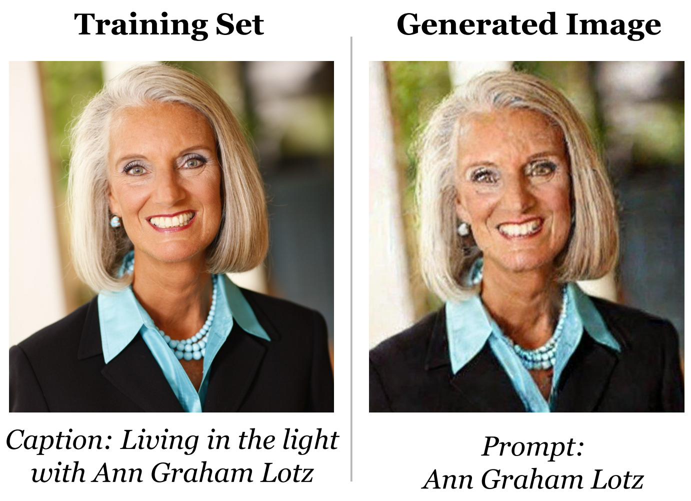


Figure 1: An image generated by Stable Diffusion [CHN+23], and the original from the model's training set.

3 Private Learning in the Convex Setting

There are multiple ways to make a learning algorithm differentially private, which differ in how they introduce noise. We will cover three approaches: *output perturbation*, *objective perturbation*, and *gradient perturbation*. An illustrative summary is in Figure 2.

The first two are mostly of theoretical interest, as they apply to models f_θ that satisfy very strong regularity conditions (e.g., convexity, Lipshitz continuity, etc.) We won't explain these terms in detail, they essentially mean that our optimization problem is "nice": e.g., it has a single global minimum θ^*, and small changes in the parameters have a bounded influence on the loss. It suffices to know that simple models such as linear and logistic regression satisfy these conditions, and that deep neural networks do not.

Output perturbation. The simplest way to get a differentially private learning algorithm is to compute the sensitivity of ERM, i.e., of the function $\arg\min_\theta \widehat{\mathcal{L}}_D(\theta)$, and then add noise to the output. This is known as *output perturbation* [CMS11].

Computing this sensitivity is not easy in general. But if the loss function has some strong regularity properties (which hold if the model f_θ is not too complex), then we can bound the sensitivity by a nice closed-form, and immediately get a differentially private algorithm by adding Gaussian noise of appropriate scale to the output of the ERM algorithm.

Objective perturbation. This approach is a bit more unusual: it perturbs the objective function that we optimize. Instead of solving the exact (regularized) ERM problem, we add a random term $b^\top\theta$ to the optimization objective, where b is sampled from a Gaussian distribution. One can then show that under suitable assumptions on the loss function, optimizing this objective with standard ERM is differentially private [CMS11].

Gradient perturbation. One of the most versatile approaches for differentially private learning is to focus on a specific optimization algorithm, namely gradient descent.

We show the general form of this algorithm in Algorithm 1. In each step, we compute the gradient over the entire training set, and then take a step of size η in the direction of the

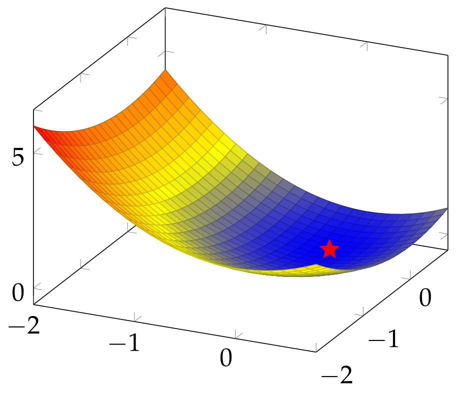
Algorithm 1: Private Projected Gradient Descent [BST14, SCS13]

```
$\theta_0 \leftarrow^{\$} \Theta$                                       // arbitrary initialization
for $t = 0, \dots, T-1$ do
    $g \leftarrow \nabla_{\theta_t} \widehat{\mathcal{L}}_D(\theta_t) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\theta_t} \ell(\theta_t, x_i, y_i)$     // compute gradient
    $\theta_{t+1} \leftarrow \theta_t - \eta \left(g \boxed{+\mathcal{N}(0, \sigma^2 I_{d\times d})}\right)$      // update parameters
    $\theta_{t+1} \leftarrow \Pi_{\Theta}(\theta_{t+1})$                         // project onto set $\Theta$
```

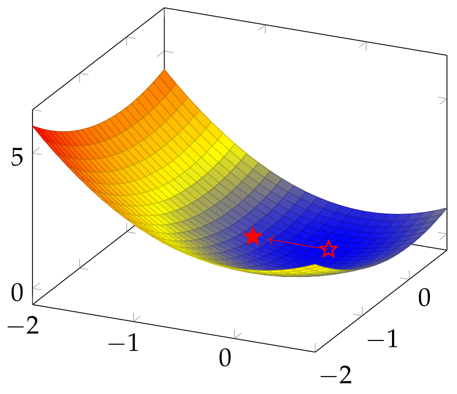
negative gradient of the loss. This amounts to minimizing the empirical risk. In convex settings, we then further project the parameters back onto some set Θ.

How can we make this algorithm private? The only computation that is data-dependent is that of the gradient g. So we can add noise to make the gradient differentially private (where we can use smoothness properties of the loss function to compute the sensitivity of the gradient.) After that, we can rely on the post-processing property of differential privacy to argue that each update step is differentially private. Finally, we can use the composition theorem to argue that the combination of T update steps is differentially private.

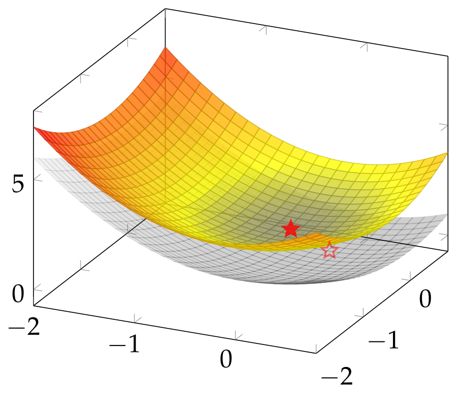
This privacy comes at a price! One can show that the solution to the private ERM problem will have empirical risk roughly $\sqrt{d}/\varepsilon$ larger than in the non-private setting (note that this statement is not about generalization, but just about the ability to fit the training data).



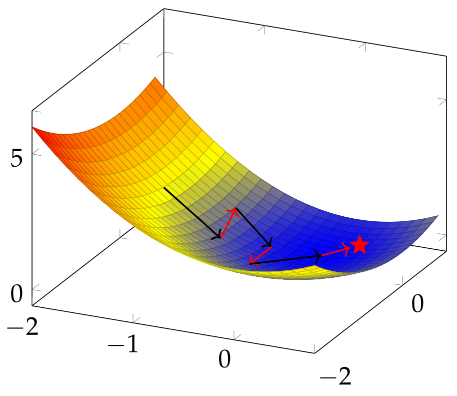
(a) Non-private ERM. We try to find weights that minimize the empirical risk.



(b) Output perturbation. We add noise to the output of the ERM algorithm.



(c) Objective perturbation. We add noise to the objective function that we optimize.



(d) Gradient perturbation. We use gradient descent with added noise.

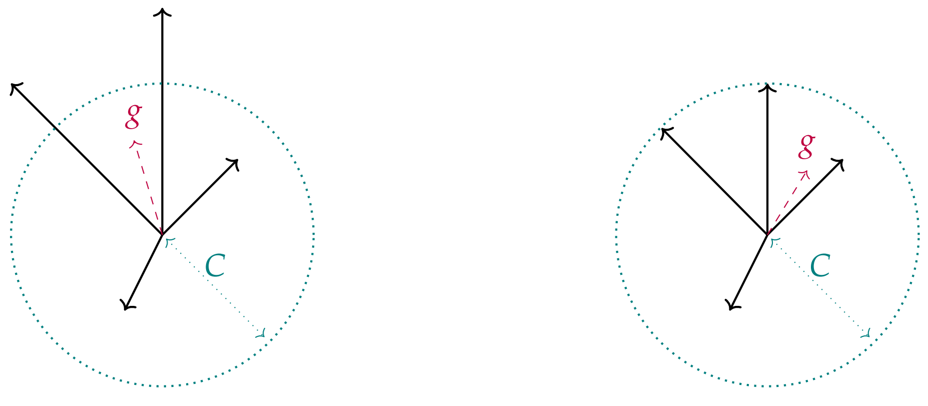
Figure 2: Illustration of private ERM techniques.

4 Private Deep Learning

Let's now see how to apply these ideas (and optimize them) in practice for training very large and complex neural networks.

We won't need to go into the details of how neural networks work for this lecture. The important bit is that these functions are not convex (i.e., there can be multiple equivalent solutions to the ERM problem), and generally don't satisfy any "nice" smoothness conditions (i.e., changing the parameters θ a tiny bit could change the model's output by a lot). This makes it hard to bound the sensitivity of the ERM output, or of individual gradients.

Gradient clipping. So what do we do then? We can just enforce that individual gradients are bounded! That is, instead of viewing our algorithm as a private optimization algorithm, we will view it more simply as a private *gradient aggregation* algorithm. In each step of gradient descent, we *clip* each example's gradient to some maximum ℓ_2 norm C, and then compute the average gradient. This is now just the mean a bunch of vectors of norm at most C, so we immediately get a bound on the ℓ_2 sensitivity.



before clipping   after clipping

The tricky bit in practice is how to set the clipping parameter C. If we set it too large, then we will overestimate the sensitivity and add too much noise. If we set it too small, then we risk removing useful information from the gradients which could harm the optimization process. To complicate things, the norm of individual gradients can vary a lot during training. Another issue is that clipping may bias the distribution of gradients. As we see in the above example, the average gradient after clipping points in a slightly different direction than the original gradient. There is no principled solution to this problem yet. Luckily, it seems that neural network optimization is not too sensitive to these issues, and a fixed choice of C often works well enough.

DP-SGD. The gradient descent algorithm we saw in Algorithm 1 computes the gradient over the entire dataset in each step. This is also called *full-batch* gradient descent. This is extremely inefficient in practice for large datasets. Instead, the canonical algorithm we use in practice (in the private and non-private setting alike) is *stochastic* gradient descent, where we sample a small batch of examples in each step to estimate the gradient.

This leads us to the DP-SGD algorithm of [ACG+16], which we present below:

Algorithm 2: Differentially Private Stochastic Gradient Descent (DP-SGD)

```
$\theta_0 \xleftarrow{\$} \Theta$                                        // arbitrary initialization
for $t = 0, \dots, T-1$ do
    Sample $S \subseteq [n]$ of size $m$                      // select a random batch
    $g_i \leftarrow \nabla_{\theta_t} \ell(\theta_t, x_i, y_i),\ \forall i \in S$       // compute per-sample gradients
    $\boxed{g_i \leftarrow C \cdot g_i / \max(C, \|g_i\|_2),\ \forall i \in S}$            // clip gradients
    $\bar{g} \leftarrow \frac{1}{|m|} \sum_{i \in S} g_i$                   // aggregate clipped gradients
    $\theta_{t+1} \leftarrow \theta_t - \eta \left(\bar{g} \boxed{+ \mathcal{N}(0, \sigma^2 I_{d \times d})}\right)$    // noisy update step
```

A naive privacy analysis. Let's analyze this algorithm with the tools we've developed so far. Since each individual gradient has norm at most C, the sensitivity of the batch gradient is at most $2C/m$, and so we can add Gaussian noise of variance $\sigma^2 = 2(2C)^2 \log(1.25/\delta)/(m^2\varepsilon^2)$ to get a (ε, δ)-differential privacy guarantee for each gradient step.

With just basic composition, after T steps, we would get a total privacy budget that grows as $(\varepsilon T, \delta T)$. This is really bad. When training neural networks, we do thousands or millions of steps, so we would need to add a huge amount of noise to each gradient to get a reasonably small ε at the end.

But recall from the last lecture that when composing (ε, δ)-DP mechanisms, we can get a privacy budget that grows roughly as $\varepsilon\sqrt{k}$ by applying the advanced composition theorem. By plugging in the advanced composition theorem (and choosing an appropriate δ') we get that DP-SGD is $(\varepsilon\sqrt{T \log(1/\delta)}, \delta T)$-differentially private. This is much better than before, but still leads to a huge privacy budget for practical settings. So we need to do better.

Amplification by subsampling. Our naive analysis assumes that each gradient step has sensitivity $2C/m$. But consider what happens if we move from a dataset $D = \{x_1, x_2, \dots, x_n\}$ to a neighboring dataset $D' = \{x_1', x_2, \dots, x_n\}$. If the batch S that we sample does not contain x_1, then the distribution of the gradient is exactly the same in both cases. It is only when x_1 is sampled that it can affect the computed gradient.

We can leverage this observation to get a better privacy analysis, for any algorithm that first subsamples the data at random, and then applies a differentially private algorithm:

Lemma 1 (Amplification by subsampling [LQS12]**).** Let M be an algorithm that is (ε, δ)-differentially private. Let M' be the algorithm that first samples an ρ-fraction of the examples from the dataset uniformly at random, and then applies M to these m examples. Then M' is (ε', δ')-differentially private, for $\varepsilon' \approx \rho\varepsilon$ and $\delta' = \rho\delta$.[a]

[a]The approximation is actually $\varepsilon' = \ln(1 + (e^\varepsilon - 1) \cdot \rho)$ which is roughly $\rho\varepsilon$ for small ε.

Let $\rho = m/n$ be the fraction of the dataset that we subsample in each step of DP-SGD. By the amplification by subsampling lemma, we now get a privacy budget that grows as $O(\varepsilon\rho\sqrt{T \log(1/\delta)}, \delta\rho T)$. The sampling rate $\rho = m/n$ is typically very small when training neural networks (e.g., we might have a dataset of millions or billions of examples, but only sample a few hundred per batch), and so this is much better than before.

Even more advanced composition. Our analysis can still be improved: the advanced composition theorem we used applies to arbitrary (ε, δ)-DP algorithms. In particular, this means it has to account for all kinds of esoteric algorithms, such as those whose privacy guarantees "fail catastrophically" with probability δ.

Recall that the guarantees of the Gaussian mechanism degrade much more gracefully: for any $\delta > 0$ there is a $\varepsilon(\delta)$ so that the mechanism is $(\varepsilon(\delta), \delta)$-DP. So the Gaussian mechanism actually satisfies an infinite number of $(\varepsilon(\delta), \delta)$-DP guarantees simultaneously.

There are other notions of privacy than approximate DP (e.g., Rényi DP [MTZ19]), which better capture this behavior. If a mechanism satisfies such a notion of privacy (e.g., like the Gaussian mechanism does), then we can get even better composition rates, which ultimately translate to a privacy bound for DP-SGD of roughly $O(\varepsilon\rho\sqrt{T}, \delta)$ for appropriate choices of parameters [ACG+16]. So we save a factor of $\sqrt{\log(1/\delta)}$ on the ε term, and a factor of ρT on the δ term. These savings are significant in practice, since we expect δ to be very small (definitely smaller than $1/n$), and $T\rho \gg 1$ (we train on over the entire dataset for multiple epochs, and so we expect to sample each point multiple times).

4.1 DP-SGD in Practice

A few notes on how DP-SGD is used in practice:

- **Central vs local DP.** The guarantee provided by DP-SGD is *central* differential privacy, where we assume that some trusted party has access to the entire dataset and runs the algorithm. But DP-SGD also lends itself very naturally to decentralized variants, which can give guarantees more similar to *local* differential privacy. We'll talk about this more in the next lecture.

- **Utility.** Adding noise to gradients hurts utility a lot, especially if the training set is small (intuitively, the larger the training set the less one sample contributes to the final model). For example, on the canonical CIFAR-10 dataset, if we want a guarantee of $\varepsilon = 1$, the best reported accuracy is around 73% [TPSM24], whereas the non-private state-of-the-art is above 99%.

  How do people get around this? Either they settle for a rather large ε guarantee, in the hopes that the guarantee isn't tight in practice, or they first *pretrain* the model non-privately (this may sound a bit like cheating, and maybe it is... [TKC22]).

- **Performance overhead.** Clipping individual gradients is not straightforward in practice. In non-private training, we directly compute the average gradient over the entire batch in a single back-propagation step. A naive implementation of DP-SGD would compute the gradient for each sample in parallel, clip them, and then take the average. There have been many optimizations proposed to make this more efficient in terms of speed and memory, which are implemented in popular libraries (see e.g., [Goo15, SVK21]).

- **Code.** There are many open-source implementations of DP-SGD, which also take care of calculating the privacy bounds using advanced composition theorems. For example, Opacus (PyTorch), TensorFlow Privacy, or JAX Privacy.

References

[ACG$^+$16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[BST14] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *2014 IEEE 55th annual symposium on foundations of computer science*, pages 464–473. IEEE, 2014.

[CHN$^+$23] Nicolas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramèr, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 5253–5270, 2023.

[CMS11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(3), 2011.

[CTW$^+$21] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021.

[Goo15] Ian Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

[LQS12] Ninghui Li, Wahbeh Qardaji, and Dong Su. On sampling, anonymization, and differential privacy or, k-anonymization meets differential privacy. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 32–33, 2012.

[MTZ19] Ilya Mironov, Kunal Talwar, and Li Zhang. Rényi differential privacy of the sampled gaussian mechanism. *arXiv preprint arXiv:1908.10530*, 2019.

[SCS13] Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. Stochastic gradient descent with differentially private updates. In *2013 IEEE global conference on signal and information processing*, pages 245–248. IEEE, 2013.

[SSSS17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[SVK21] Pranav Subramani, Nicholas Vadivelu, and Gautam Kamath. Enabling fast differentially private sgd via just-in-time compilation and vectorization. *Advances in Neural Information Processing Systems*, 34:26409–26421, 2021.

[TKC22] Florian Tramèr, Gautam Kamath, and Nicholas Carlini. Position: Considerations for differentially private learning with large-scale public pretraining. In *Forty-first International Conference on Machine Learning*, 2022.

[TPSM24] Xinyu Tang, Ashwinee Panda, Vikash Sehwag, and Prateek Mittal. Differentially private image classification by learning priors from random processes. *Advances in Neural Information Processing Systems*, 36, 2024.