CS 325: Private Data Analysis **Fall 2022**

Lecture 15: Differentially Private Empirical Risk Minimization: III

Lecturer: Di Wang *Scribes: Di Wang*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

15.1 Gradient Descent

An important family of optimization algorithms are based on gradient descent. The idea is to iteratively improve a candidate solution θ by taking a small step in the direction of opposite to the gradient at θ.

When the loss L is differentiable, the gradient $\nabla L(\theta)$ gives the coefficients of the best linear approximation to L in the area right around θ. Moving in the direction opposite to the gradient is thus the same as moving in the direction in which the function drops most steeply. If we think of the graph of the function as a surface, then the gradient tells us which direction a drop of water would trickle in if it were placed at the point corresponding to θ. Water finds minima of a landsacpe, and so can gradient descent.

Of course, gradient descent methods can get stuck in local minima. These are points where the function cannot be decreased by taking small steps. A global minimum, in contrast, is an input where the function takes its smallest possible value. On the surface of the earth, lakes and ponds occur at local minima; the global minimum is the bottom of the Mariana Trench.

Even though gradient descent wont always work, we can prove that it convergesand get a very good handle on how quicklyif we focus on convex functions.

There are many variations on gradient descent and its analysis. We will see on variant, called projected gradient descent which aims to minimize a loss L on a set $\mathcal{C}$.

Algorithm 1 Gradient Descent

1: Denote the initial point w_0.
2: **for** $t = 1, 2, \cdots, T$ **do**
3: Let $\theta_t = \Pi_{\mathcal{C}}(\theta_{t-1} - \eta \nabla L(\theta_{t-1}, D))$.
4: **end for**
5: Return $\hat{\theta} = \frac{1}{T}\sum_{t=0}^{T-1} w_t$ or $\hat{\theta} = \theta_{T-1}$.

There is a variant of the GD method, which is called the Stochastic Gradient Descent (SGD). Rather than evaluating the full gradient of $L(\cdot)$ at each step of the algorithm we can save a factor of n in the running time by evaluating the gradient at a single, randomly chosen point. For each t,

$$\tilde{g}_{t-1} = \nabla \ell(\theta_{t-1}, x_i) \text{ for } i \sim \text{uniform}\{1, 2, \cdots, n\}. \tag{15.1}$$

Again, the expectation of $\tilde{g}_{t-1}$ is $\nabla L(\theta_{t-1}, D)$ since

$$\mathbb{E}\tilde{g}_{t-1} = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\theta_{t-1}, x_i) = \nabla L(\theta_{t-1}, D).$$

15.2 Private Gradient Descent

We first talk about a DP version of GD and then talk about SGD. We will first show that the Algorithm 2

Algorithm 2 DP Gradient Descent

1: Denote the initial point w_0.
2: **for** $t = 1, 2, \cdots, T$ **do**
3: Denote $\tilde{g}_{t-1} = \nabla L(\theta_{t-1}, D) + \mathcal{N}(0, \sigma^2 I_d)$, where $\sigma = O(\frac{L\sqrt{T \log 1/\delta}}{n\epsilon})$.
4: Let $\theta_t = \Pi_{\mathcal{C}}(\theta_{t-1} - \eta \tilde{g}_{t-1})$.
5: **end for**
6: Return $\hat{\theta} = \frac{1}{T}\sum_{t=0}^{T-1} w_t$ or $\hat{\theta} = \theta_{T-1}$.

is (ϵ, δ)-DP if $\sigma = O(\frac{L\sqrt{T \log 1/\delta}}{n\epsilon})$.

Lemma 15.1 *If $\ell(\cdot, x, y)$ is L-Lipschitz, then Algorithm 2 is (ϵ, δ)-DP for $\sigma^2 \geq \frac{64L^2T \log \frac{2}{\delta} \log \frac{2.5T}{\delta}}{n^2\epsilon^2}$.*

Proof: Before that we recall two lemmas for the advanced composition theorem and the Gaussian mechanism.

Lemma 15.2 *Given target privacy parameters $0 < \epsilon < 1$ and $0 < \delta < 1$, to ensure $(\epsilon, k\delta' + \delta)$-DP over k mechanisms, it suffices that each mechanism is (ϵ', δ')-DP, where $\epsilon' = \frac{\epsilon}{2\sqrt{2k \ln(2/\delta)}}$ and $\delta' = \frac{\delta}{2k}$.*

Lemma 15.3 (Gaussian Mechanism) *For a given function $f : \mathcal{X}^n \mapsto \mathbb{R}^d$, we defined the Gaussian mechanism as*

$$A(D) = f(D) + \mathcal{N}(0, \frac{2\Delta^2 \log \frac{1.25}{\delta}}{\epsilon^2} I_d).$$

Then for any $\epsilon \in (0, 1)$ and $\delta > 0$, the Gaussian mechanism is (ϵ, δ)-DP.

Note that Algorithm 2 is an instance of the Gaussian noise with adaptively selected queries. The adaptivity comes in because the query point θ_t depends on the noisy gradients from previous steps.

Recall that, by the linearity of the gradient, $\nabla L(\theta, D)$, can be written as the average of individual loss gradients $\nabla \ell(\theta, x, y)$. It therefore has ℓ_2-norm sensitivity at most $\Delta = \frac{2L}{n}$. To analyze the mechanism as a whole, we use the advanced composition method theorem. We just want each iteration is $(\epsilon' = \frac{\epsilon}{2\sqrt{2T \ln(2/\delta)}}, \delta' = \frac{\delta}{2T})$-DP. By the Gaussian mechanism we can see if $\sigma^2 \geq \frac{64L^2T \log \frac{2}{\delta} \log \frac{2.5T}{\delta}}{n^2\epsilon^2}$ then it is $(\epsilon' = \frac{\epsilon}{2\sqrt{2T \ln(2/\delta)}}, \delta' = \frac{\delta}{T})$-DP. ■

Running Time: As written, DP-GD requires us to compute the gradient of all individual losses for each time step T, and then to perform T summations of n vectors for a total running time of $O(nT(d + Time(\nabla \ell)))$, where $Time(\nabla \ell)$ is the time for computing the gradient of the loss.

15.3 Private Stochastic Gradient Descent

We can improve the run time of gradient descent dramatically if we use the second idea above for estimating the gradient, namely we sample only one record at each step and use its gradient as a proxy for the gradient of

$L(\cdot, D)$. We can combine the two ideas to obtain a faster differentially private algorithm known as DP-SGD. Specifically, at each step we set

$$\tilde{g}_{t-1} = \nabla \ell(\theta_{t-1}, x_i) + \mathcal{N}(0, \sigma^2 I_d) \text{ for } i \sim \text{uniform}\{1, 2, \cdots, n\}. \tag{15.2}$$

Thus, the framework of DP-SGD can be represented as in Algorithm 3, and the following analysis follows [2].

Algorithm 3 DP-SGD

1: Denote the initial point w_0.
2: **for** $t = 1, 2, \cdots, T$ **do**
3: Denote $\tilde{g}_{t-1} = \nabla \ell(\theta_{t-1}, x_i) + \mathcal{N}(0, \sigma^2 I_d)$ for $i \sim \text{uniform}\{1, 2, \cdots, n\}$, where $\sigma^2 = \frac{64L^2T \log \frac{2}{\delta} \log \frac{2.5T}{\delta n}}{n^2\epsilon^2}$.
4: Let $\theta_t = \Pi_{\mathcal{C}}(\theta_{t-1} - \eta \tilde{g}_{t-1})$.
5: **end for**
6: Return $\hat{\theta} = \frac{1}{T}\sum_{t=0}^{T-1} w_t$ or $\hat{\theta} = \theta_{T-1}$.

Before analyzing the noise σ^2 we need to ensure (ϵ, δ)-DP, let us recall the lemma of privacy amplification via uniform sampling. Although in Lecture 8 we only considered the Poisson sampling, the same conclusion holds for uniform sampling.

Lemma 15.4 *Let m, n be integers with $m \leq n$. Let A be any algorithm taking inputs in $\mathcal{X}^m$. and define A' in $\mathcal{X}^n$ as follows: On input D, $A'(D)$ selects a uniformly random set S of size m from $[n]$, and returns $A(D|_S)$.*

Then if A is (ϵ, δ)-DP, then A' is (ϵ', δ')-DP for $\epsilon' = \ln(1 + (e^\epsilon - 1)\frac{m}{n}) \approx \frac{m}{n}\epsilon$ and $\delta' = \frac{m}{n}\delta$.

We first consider the case where $m = 1$, it is the same for the general case. To use Lemma 15.4 we consider the algorithm A as

$$A(D) = \sum_{i=1}^{n} \nabla \ell(w, x_i) + \mathcal{N}(0, \sigma^2 I_d).$$

If A is (ϵ, δ)-DP, then we have

$$A'(D) = \nabla \ell(w, x_i) + \mathcal{N}(0, \sigma^2 I_d) \tag{15.3}$$

will be $(\frac{1}{n}\epsilon, \frac{1}{n}\delta)$-DP.

By Lemma 15.1 we know that when $\sigma^2 = \frac{64L^2T \log \frac{2}{\delta} \log \frac{1.25T}{\delta}}{\epsilon^2}$ then A is $(\epsilon' = \frac{\epsilon}{2\sqrt{2T \ln(2/\delta)}}, \delta' = \frac{\delta}{T})$-DP. Thus $A'(D)$ will be $(\epsilon' = \frac{\epsilon}{2n\sqrt{2T \ln(2/\delta)}}, \delta' = \frac{\delta}{nT})$-DP. Thus, for $A'(D)$, it is sufficient for $\sigma^2 = \frac{64L^2T \log \frac{2}{\delta} \log \frac{2.5T}{\delta n}}{n^2\epsilon^2}$.

Lemma 15.5 *If $\ell(\cdot, x, y)$ is L-Lipschitz, then Algorithm 2 is (ϵ, δ)-DP for $\sigma^2 = \frac{64L^2T \log \frac{2}{\delta} \log \frac{2.5T}{\delta n}}{n^2\epsilon^2}$.*

Next, we will talk about the utility. We need the following lemmas for SGD in the general convex case:

Lemma 15.6 *Let $L(\cdot, D)$ be a convex function and let $\theta^* = \arg\min_{\theta \in \mathcal{C}} L(\theta, D)$. Let θ_0 be an arbitrary point $\mathcal{C}$. Consider the stochastic gradient descent algorithm $\theta_t = \Pi_{\mathcal{C}}[\theta_t - \eta_{t-1} G_{t-1}(\theta_{t-1})]$, where $\mathbb{E}[G_{t-1}(\theta_{t-1})] = \nabla L(\theta_{t-1}, D), \mathbb{E}[\|G_{t-1}(\theta_{t-1})\|_2^2] \leq G^2$ and the learning rate $\eta_{t-1} = \frac{\|\mathcal{C}\|_2}{L\sqrt{t}}$. Then for any $T > 1$, we have*

$$\mathbb{E}L(\theta_T, D) - L(\theta^*, D) = O(\frac{\|\mathcal{C}\|_2 G \log T}{\sqrt{T}}) \tag{15.4}$$

In our algorithm, $G_{t-1}(\theta_{t-1}) = \nabla \ell(w, x_i) + \mathcal{N}(0, \sigma^2 I_d)$ we have

$$\mathbb{E}[\|G_{t-1}(\theta_{t-1})\|_2^2] \leq \sigma^2 d + L^2. \tag{15.5}$$

Thus,

$$\mathbb{E}L(\theta_T, D) - L(\theta^*, D) = O(\frac{\|\mathcal{C}\|_2 L \log T}{\sqrt{T}} + \frac{\|\mathcal{C}\|_2 \sqrt{d}\sigma \log T}{\sqrt{T}}) = O(\frac{\|\mathcal{C}\|_2 L \log T}{\sqrt{T}} + \frac{\|\mathcal{C}\|_2 \sqrt{d}\sqrt{T}\sqrt{\log 1/\delta \log T/n\delta} \log T}{n\epsilon\sqrt{T}}) \tag{15.6}$$

Thus, if we take $T = n^2$, then

$$\mathbb{E}L(\theta_T, D) - L(\theta^*, D) = O(\frac{\sqrt{d}\log 1/\delta \log^{1.5} n}{n\epsilon}).$$

For strongly convex setting we recall the following lemma:

Lemma 15.7 *Let $L(\cdot, D)$ be an α-strongly convex function and let $\theta^* = \arg\min_{\theta \in \mathcal{C}} L(\theta, D)$. Let θ_0 be an arbitrary point $\mathcal{C}$. Consider the stochastic gradient descent algorithm $\theta_t = \Pi_{\mathcal{C}}[\theta_t - \eta_{t-1} G_{t-1}(\theta_{t-1})]$, where $\mathbb{E}[G_{t-1}(\theta_{t-1})] = \nabla L(\theta_{t-1}, D), \mathbb{E}[\|G_{t-1}(\theta_{t-1})\|_2^2] \leq G^2$ and the learning rate $\eta_{t-1} = \frac{1}{\alpha t}$. Then for any $T > 1$, we have*

$$\mathbb{E}L(\theta_T, D) - L(\theta^*, D) = O(\frac{G^2 \log T}{T}). \tag{15.7}$$

Thus, for strongly convex case, we have

$$\mathbb{E}L(\theta_T, D) - L(\theta^*, D) = O(\frac{L^2 \log T}{T} + \frac{dL^2 T \log \frac{2}{\delta} \log \frac{2.5T}{n\delta}}{n^2\epsilon^2 T}) = O(\frac{L^2 \log T}{T} + \frac{dL^2 \log \frac{2}{\delta} \log \frac{2.5T}{n\delta} \log T}{n^2\epsilon^2}).$$

Thus, when $T = n^2$, we have

$$\mathbb{E}L(\theta_T, D) - L(\theta^*, D) = O(\frac{dL^2 \log^2 1/\delta \log^2 n}{n^2\epsilon^2}).$$

As we know, the best known result for general convex and strongly convex loss functions is $O(\frac{\sqrt{d \log \frac{1}{\delta}}}{n\epsilon})$ and $O(\frac{d \log \frac{1}{\delta}}{n^2\epsilon^2})$, respectively. Thus, compared with our previous results, we can see there is still a gap of $O(\sqrt{\log 1/\delta} \log^{1.5} n)$ and $O(\log 1/\delta \log^2 n)$. Where do these terms come from? For general convex and strongly convex one, there is a factor of $\log n$ which come from the converge rate of SGD, *i.e.,* Lemma 13.6 and Lemma 13.7. The other $O(\sqrt{\log 1/\delta} \log n)$ and $O(\log 1/\delta \log n)$ come from the advanced composition theorem Lemma 13.5. Thus, to remove these factors, we need improved converge rate of SGD and improved composition theorem. For the converge rate of SGD actually we can indeed improve to $O(\frac{1}{\sqrt{T}})$ instead of $O(\frac{\log T}{\sqrt{T}})$. For the advanced theorem, actually, one approach is to transfer to Rényi-DP, instead of (ϵ, δ)-DP and use its sub-sampling property and composition theorem. In the following we will not talk about the Rényi-DP based approach, one can see Lecture 8 for details. But we will talk about another approach which is called the Moment Accountant.

Thus, in total, actually we can indeed improve the upper bound of the utility to $O(\frac{\sqrt{d \log \frac{1}{\delta}}}{n\epsilon})$ and $O(\frac{d \log \frac{1}{\delta}}{n^2\epsilon^2})$ for general convex and strongly convex loss functions by using advanced optimization method and advanced composition theorem.

15.4 Private Batch SGD and Moment Accountant

We can see that the previous general framework could be used to any stochastic gradient descent based method. Specifically, we could use it to the optimization methods in deep learning. However, there are still some issues:

- In deep learning, we prefer to use batch SGD instead of the SGD in Algorithm 3.
- In the problems of deep learning, the loss function may not be Lipschitz.
- As we mentioned, the advanced composition theorem could be further improved for Gaussian mechanism.

[1] addressed the above three issues. And here we will show their algorithm DP-SGD.

Algorithm 4 DP-SGD

1: Denote the initial point w_1.
2: **for** $t = 1, 2, \cdots, T$ **do**
3: Take a random sample L_t with sampling probability $\frac{L}{N}$,
4: For each $i \in L_t$, compute $g_t(x_i) = \nabla L(\theta_t, x_i)$
5: Let $\bar{g}_t(x_i) = g_t(x_i) \min\{1, \frac{C}{\|g_t(x_i)\|_2}\}$.
6: Denote $\tilde{g}_t = \frac{1}{L}(\sum \bar{g}_t(x_i) + \mathcal{N}(0, 4\sigma^2 C^2 I_d)$.
7: Update $\theta_{t+1} = \theta_t - \eta_t \tilde{g}_t$.
8: **end for**

We can see that to address the first issue, Algorithm 4 generalizes to batch SGD. For the second one, they perform a truncation step to let each $\|\bar{g}_t(x_i)\|_2 \leq C$. And thus, the ℓ_2-norm sensitivity of $\bar{g}_t(x_i)$ is 2C. After that, they adding Gaussian noise to the subsampled batched gradient. We finally focus on the third issue.

Theorem 15.8 *There exists constants c_1 and c_2 so that given the sampling probability $q = \frac{L}{N}$ and the number of steps T, for any $\epsilon < c_1 q^2 T$, Algorithm 4 is (ϵ, δ)-DP if we choose*

$$\sigma^2 \geq c_2 \frac{q^2 T \log \frac{1}{\delta}}{\epsilon}. \tag{15.8}$$

Note that of we use the advanced composition theorem, then we need $\sigma^2 \geq \Omega(\frac{q^2 T \log \frac{T}{\delta} \log \frac{1}{\delta}}{\epsilon})$. Thus, we can improve a factor of $\log \frac{T}{\delta}$. Note that this factor is large in practice, espectially in deep learning since $\delta = o(\frac{1}{n})$ and T could be very large.

To proof the theorem, the authors proposed a new technique which is called the moment accountant. The motivation is that a mechanism A is (ϵ, δ)-differentially private is equivalent to a certain tail bound on As privacy loss random variable. While the tail bound is very useful information on a distribution, composing directly from it can result in quite loose bounds. We instead compute the log moments of the privacy loss random variable, which compose linearly. We then use the moments bound, together with the standard Markov inequality, to obtain the tail bound, that is the privacy loss in the sense of differential privacy.

Recall that, in lecture 6 we introduced the privacy loss as a random variable: For a neighboring datasets $D \sim D'$, if we denote $p_{A(D)}(y)$ and $p_{A(D')}(y)$ as the density function of $A(D)$ and $A(D')$ respectively. And we denote the privacy loss as $\ell_{D,D'}(y) = \ln(\frac{p_{A(D)}(y)}{p_{A(D')}(y)})$. We could think $L_{A,D,D'} = \ell_{A,D,D'}(Y)$ as the

transformation of the output random variable $Y = A(D)$. Thus, our goal is to see the tail bound of $L_{A,D,D'}$ that is $\mathbb{P}(L_{A,D,D'} > \epsilon)$.

If we think $L_{A,D,D'}$ as a random variable, then here we consider the log its moment generating function. That is for any λ:

$$\alpha_A(\lambda; D, D') = \log \mathbb{E}[\exp(tL_{A,D,D'})] \tag{15.9}$$

And $\alpha_A(\lambda) = \max_{D \sim D'} \alpha_A(\lambda; D, D')$.

We first show two properties of $\alpha_A(\lambda)$

Theorem 15.9 *1. Suppose that a mechanism A consists of a sequence of adaptive mechanisms $A_1, A_2, \cdots, A_k$. Then for any λ,*

$$\alpha_A(\lambda) \leq \sum_{i=1}^{k} \alpha_{A_i}(\lambda). \tag{15.10}$$

2. For any $\epsilon > 0$, the mechanism A is (ϵ, δ)-DP for

$$\delta = \min_{\lambda} \exp(\alpha_A(\lambda) - \lambda\epsilon). \tag{15.11}$$

Proof: The first step is quite the same as what we did in the proof of the advanced composition theorem: we denote $y = (y_1, y_2, \cdots, y_k) \in \mathcal{Y}_1 \times \cdots \mathcal{Y}_k$. Then we have

$$p_{A(D)}(y) = p_{A_1(D)}(y_1) \times p_{A_2(D,y_1)}(y_2) \times \cdots \times p_{A_k(D,y_1,\cdots,y_{k-1})}(y_k). \tag{15.12}$$

Thus,

$$L_{D,D'}(y) = \ln \frac{p_{A_1(D)}(y_1)}{p_{A_1(D')}(y_1)} + \ln \frac{p_{A_2(D,y_1)}(y_2)}{p_{A_2(D',y_1)}(y_2)} + \cdots + \ln \frac{p_{A_k(D,y_1,\cdots,y_{k-1})}(y_k)}{p_{A_k(D',y_1,\cdots,y_{k-1})}(y_k)}. \tag{15.13}$$

We can see each of them is a privacy loss. Denote $L_{D,D'}(y_1, \cdots, y_{k-1}) = \log \frac{p_{A_k(D,y_1,\cdots,y_{k-1})}(y_k)}{p_{A_k(D',y_1,\cdots,y_{k-1})}(y_k)}$. Thus

$$\begin{aligned}
\alpha_A(\lambda, D, D') &= \log \mathbb{E}[\exp(tL_{A,D,D'})] \\
&\leq \log \mathbb{E}[\exp(t \sum_{j=0}^{k-1} L_{D,D'}(y_1, \cdots, y_{j-1}))] \\
&\leq \sum_{j=0}^{k-1} \log \mathbb{E}[\exp(tL_{D,D'}(y_1, \cdots, y_{j-1}))]
\end{aligned}$$

Take the maximal w.r.t D, D' we can proof (1).

For the second one, the proof is based on the standard Markov's inequality. We have

$$\mathbb{P}_{y \sim p_{A(D)}}(L_{A,D,D'}(y) > \epsilon) = \mathbb{P}_{y \sim p_{A(D)}}(\exp(\lambda L_{A,D,D'}(y)) > \exp(\lambda\epsilon)) \tag{15.14}$$

$$\leq \frac{\mathbb{E}_{y \sim p_{A(D)}} \exp(\lambda L_{A,D,D'}(y))}{\exp(\lambda\epsilon)} \leq \exp(\alpha_A(\lambda) - \lambda\epsilon). \tag{15.15}$$

■

The item 1 in Theorem 13.9 tells us it is sufficient to bound $\alpha_{A_i}(\lambda)$, that is the Gaussian mechanism on the subsampled gradients. If we can do this, then we can use the item 2 to specify the noise we need to add.

For the bound $\alpha_{A_i}(\lambda)$, where A_i the Gaussian mechanism on the subsampled gradients we have the following lemma:

Lemma 15.10 *Suppose $f : \mathcal{X} \mapsto \mathbb{R}^d$ with $\|f\|_2 \leq 1$. Let $\sigma \geq 1$ and J be a sample from n where each $i \in [n]$ is chosen independently with probability $q \leq \frac{1}{16\sigma}$. Then for any positive $\lambda \leq \sigma^2 \ln \frac{1}{q\sigma}$, the mechanism $A(D) = \sum_{i\in J} f(x_i) + \mathcal{N}(0, \sigma^2 I_d)$ satisfies*

$$\alpha_A(\lambda) \leq \frac{q^2\lambda(\lambda+1)}{(1-q)\sigma^2} + O(\frac{q^3\lambda^3}{\sigma^3}). \tag{15.16}$$

Proof: The proof here is complicated, so here we just give a sketch of the proof.

Consider $D = D' \cup \{x_n\}$. Without loss if generality we assume $f(x_n) = (1, 0, \cdots, 0)^T$ and $\sum_{i\in J\setminus\{n\}} f(x_i) = 0$. Thus $A(D)$ and $A(D')$ are distributed identically except for the first coordinate and thus we have a one-dimensional problem. Thus, by the subsampling property we have $A(D') \sim \mathcal{N}(0, \sigma^2)$ and denote its density function as μ_0 and $A(D) \sim (1-q)\mathcal{N}(0, \sigma^2) + q\mathcal{N}(1, \sigma^2)$, we denote the density function $A(D)$ as μ and the density function of $\mathcal{N}(1, \sigma^2)$ as μ_1.

Thus, we want to upper bound

$$\mathbb{E}_{z\sim\mu}[(\frac{\mu(z)}{\mu_0(z)})^\lambda] \leq \alpha$$
$$\mathbb{E}_{z\sim\mu_0}[(\frac{\mu_0(z)}{\mu(z)})^\lambda] \leq \alpha$$

We only consider the second one, which is more harder.

$$\begin{aligned}
&\mathbb{E}_{z\sim\mu_0}[(\frac{\mu_0(z)}{\mu(z)})^\lambda] = \mathbb{E}_{z\sim\mu}[(\frac{\mu_0(z)}{\mu(z)})^{\lambda+1}] \\
&= \mathbb{E}_{z\sim\mu}[(1 + \frac{\mu_0(z) - \mu(z)}{\mu(z)})^{\lambda+1}] \\
&= \sum_{t=0}^{\lambda+1} \binom{\lambda+1}{t} \mathbb{E}_{z\sim\mu}[(\frac{\mu_0(z) - \mu(z)}{\mu(z)})^t)]
\end{aligned}$$

We can show that the first term is 1, the second term is 0, the third term

$$\binom{\lambda+1}{2} \mathbb{E}_{z\sim\mu}[(\frac{\mu_0(z) - \mu(z)}{\mu(z)})^2)] \leq \frac{\lambda(\lambda+1)q^2}{(1-q)\sigma^2}.$$

Moreover we can show that

$$\sum_{t=3}^{\lambda+1} \binom{\lambda+1}{t} \mathbb{E}_{z\sim\mu}[(\frac{\mu_0(z) - \mu(z)}{\mu(z)})^t)] \leq O(\frac{q^3\lambda^3}{\sigma^3}).$$

■

Now we can proof our main result.

Proof:[**Proof of Theorem 15.8**] Assume that σ, λ satisfy the assumptions in the previous Lemma. Then by Theorem 13.9 we have $\alpha(\lambda) \leq \frac{CTq^2\lambda^2}{\sigma^2}$. Moreover, in order to guarantee (ϵ, δ)-DP, it is suffice that

$$\frac{CTq^2\lambda^2}{\sigma^2} \leq \frac{\lambda\epsilon}{2}$$
$$\exp(-\frac{\lambda\epsilon}{2}) \leq \delta.$$

In addition we need $\lambda \leq \sigma^2 \log \frac{1}{q\sigma}$. Thus we can see that by setting $\sigma = c_2 \frac{q\sqrt{T \log 1/\delta}}{\epsilon}$ it satisfies the above conditions. ■

15.5 DP-SGD and Private Deep Learning

[1] firstly used the previous DP-SGD algorithm to deep neural networks. One state-of-the-art result (based on DPSGD, but with additional tuning) gets 98.1% accuracy with $(2.93, 10^{-5})$-DP. whereas the best non-private methods get closer to 99.8% accuracy. A more challenging task is image classification on the CIFAR-10 dataset. Non-privately, methods are able to achieve 99.7% accuracy, while with $(7.53, 10^5)$-DP, we are able to achieve only 66.2% accuracy. Neither the privacy guarantee or the accuracy guarantee are very compelling at the moment. There is clearly a lot of work to be done to improve the state of differentially private machine learning.

While DPSGD acts as a drop in replacement for SGD, there are many qualitative differences in the differentially private setting. We discuss some of them here. One common complaint is that DPSGD is much slower than traditional SGD. The reason is that DPSGD requires one to clip each individual gradient in order to limit the sensitivity. Most modern machine learning frameworks are not built for this procedure, having methods which are optimized to use the GPU to compute the gradient over an entire mini-batch at once in parallel. The requirement of per-example gradients diverges from this standard, and the naive method of computing them would necessitate processing all points sequentially, thus losing all speedup granted by parallel processing on GPUs. As such, alternative algorithms for obtaining per-example gradients have been proposed.

As mentioned before, the ReLU is the most popular activation function in non-private machine learning, due to several convenient properties such as the ability to avoid vanishing gradients (an issue we will not get into here). However, in the differentially private setting, it appears that the tanh function (considered obsolete in the non-private setting) yields significant performance improvements. This is one example showing that there can be benefits associated with modifying a networks architecture for the differentially private setting. On a similar note, nonprivate neural networks have grown exceptionally large, with the number of parameters growing into the hundreds of billions. This is because the size of a model is associated with higher capacity, meaning that it can learn more functions. However, DPSGD requires the addition of Gaussian noise of magnitude proportional to the square root of the number of parameters. Thus, if we tried to run DPSGD on such a large model, our noise would drown out all signal. One must carefully choose the network architecture with this in mind too small and the network wouldnt be able to represent the function, and too large and the noise would be overwhelming.

Hyperparameter tuning is a common challenge in machine learning tasks, and even more are introduced in the differentially private setting. For instance, how does one choose the learning rate, the lot size, the clipping norm, or the number of epochs? The canonical way to do this (non-privately) is to run a number of analyses on the training data with various hyperparameter settings, and choose the one which performs best on a validation set. Doing this in the differentially private setting would incur a cost in our privacy budget with every run, a cost which is currently omitted in most DP machine learning papers. This can be seen as pushing the methods to their limits, though they do not correspond to true privacy guarantees. Some methods have been proposed for hyperparameter optimization in a differentially private manner. Another approach is to use public (non-sensitive) data that may have come from the same distribution as the private dataset. One can thus perform hyperparameter tuning on the public data, which will hopefully be suitable for the private data. An example of this is presented in, in which they treat the large CIFAR-100 dataset as public, and use this to train a neural network. They then freeze the majority of parameters in the model, and train the remainder privately on the sensitive CIFAR-10 dataset, achieving much better accuracy on the test set than without the CIFAR-100 training.

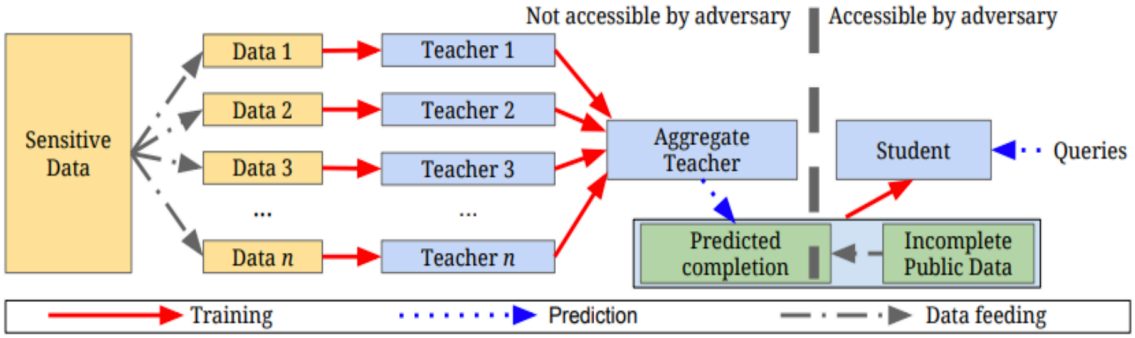

Figure 15.1: PATE Framework

15.5.1 PATE

Another approach involves private aggregation of teacher ensembles, or PATE for short. This approach was introduced in [3], refined in [4]. In contrast to DPSGD, PATE can be used on top of an arbitrary non-private learning algorithm, making it more flexible essentially, it only relies upon the non-private method being as accurate as possible. On the other hand, it requires a supply of public (i.e., non-sensitive) unlabeled data during the training process, which comes from the same distribution as the sensitive data.

The idea is the following. We split the data into n datasets, and use each dataset to train a separate non-private classifier. These will be known as the teachers. Ideally, each of these teachers will then be highly accurate on the dataset. Using this, it is not hard to make a single prediction under the constraint of differential privacy. If we want to label some point n, we run all n teachers on x to obtain non-private labels $y_1, \cdots, y_n$. If these teachers are highly accurate, (say) $0.99n$ of them will agree on the correct label. Since this leads to a large gap between the mode and the second most frequent label, we can release this value exactly using the PTR (Propose-Test-Release, see Lecture 10). Remember how this works: we measure the gap between the two most frequently occurring elements and check if this is large (privatizing this quantity using the Laplace mechanism). If so, we can output the mode exactly, otherwise, we output randomly.

References

[1] 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.

[2] 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.

[3] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.

[4] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908*, 2018.