CSCI 699: Privacy Preserving Machine Learning - Week 3

Algorithms for Differentially Privacy and Machine Learning

Sai Praneeth Karimireddy, Sep 13 2024

Recap

- Differential privacy
  - $\forall y, \forall$ similar D, D' : $\dfrac{\Pr[Y = y \mid \mathscr{D} = D]}{\Pr[Y = y \mid \mathscr{D} = D']} \leq e^{\varepsilon}$
  - connection to tradeoff curves of attacker
  - If f is Δ_1-sensitive wrt ℓ_1 norm, Laplace mechanism
    - output ith coordinate = $f_i(D) + \text{Lap}(\Delta_1/\varepsilon)$

Recap

- Approximate differential privacy

For $t \sim A(D)$ and $\mathscr{L}_{D,D'} = \ln\left(\frac{Pr[A(D) = t]}{Pr[A(D') = t]}\right)$,

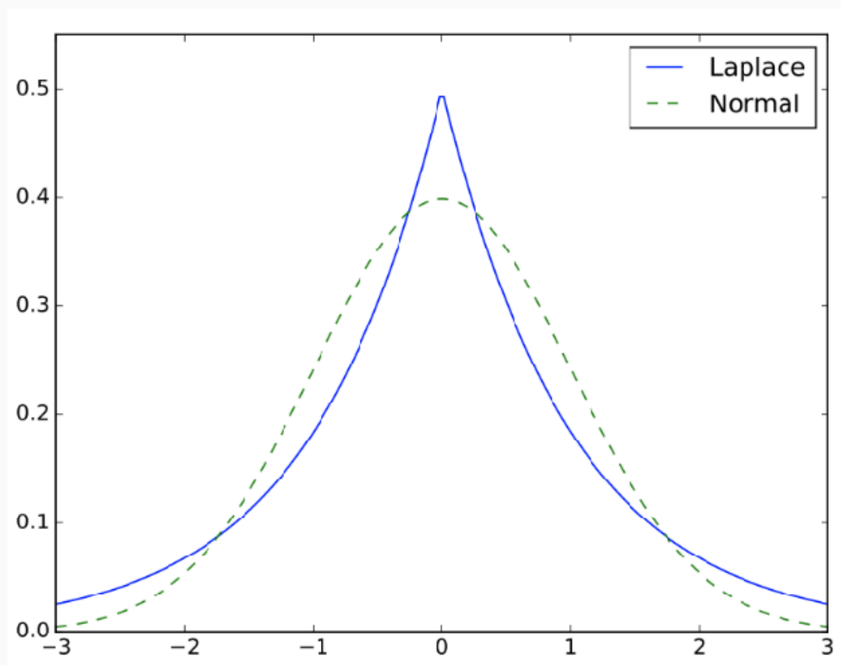
A satisfies (ε, δ)-DP iff for any neighboring datasets $D, D' \in \chi^n$

$$Pr\left[\mathscr{L}_{D,D'} \geq \varepsilon\right] \leq \delta$$

- $\delta \leq 1/n$ or 10^{-5}

Recap

- Gaussian mechanism
- If f is Δ_2-sensitive wrt ℓ_2 norm, Gaussian mechani
  - output $f(D) + \mathcal{N}(0\ ,\ \sigma^2 I_d)$
  - How large should σ^2 be for (ε, δ)-DP ?



Analytic vs. programatic Gaussian Mechanism

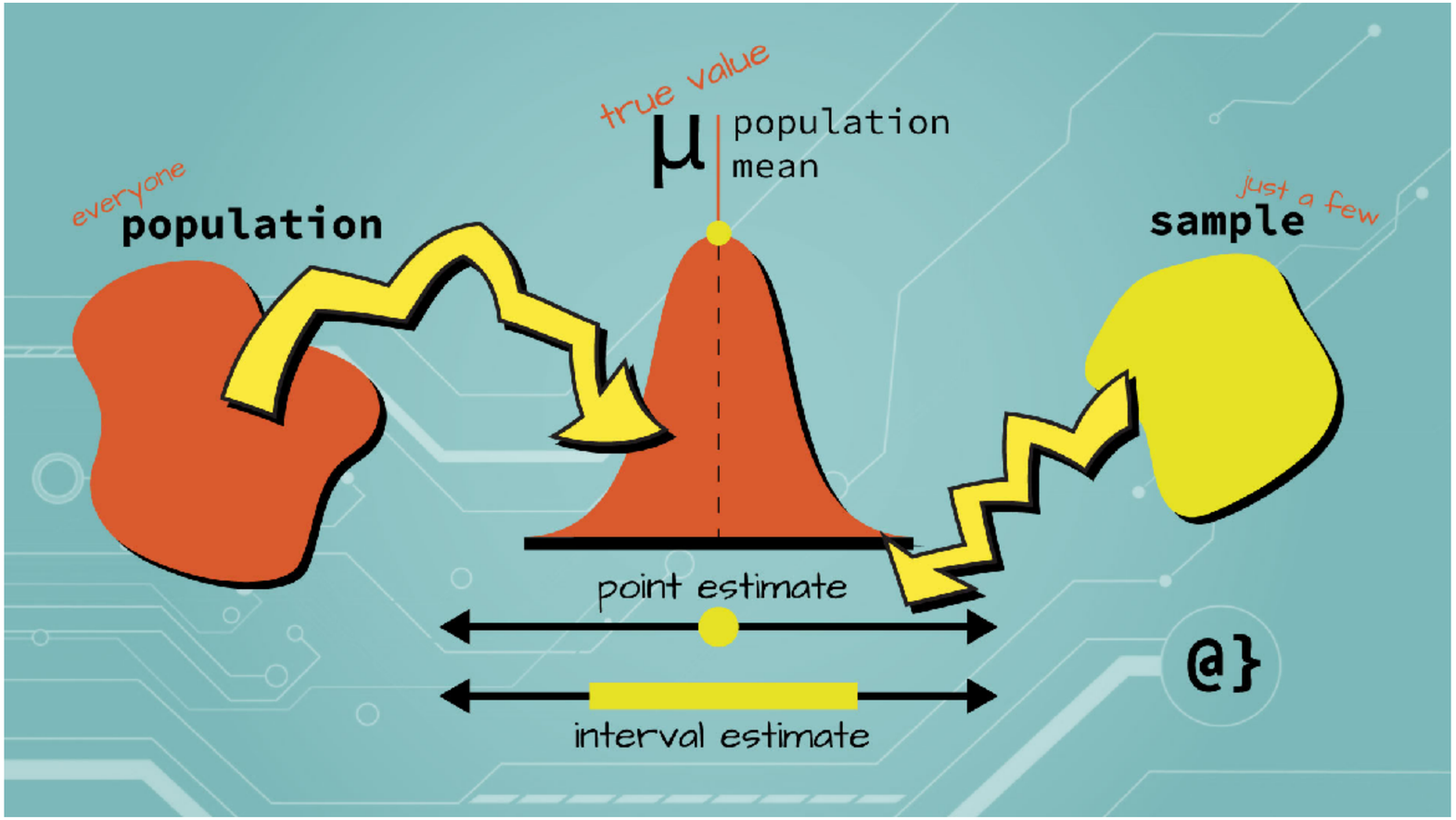
- Original paper [Dwork et al. 2006]:
  - $\sigma \geq \dfrac{\Delta_2 \sqrt{2\ln(1.25/\delta)}}{\varepsilon}$ suffices.
- But suboptimal. What is the optimal value?
- [Balle and Wang ICML '18]:
  - $u^* = \arg\min_u \Phi(-\sqrt{\varepsilon u}) - \Phi(-\sqrt{\varepsilon(u+2)})$
  - Use $\sigma = (\sqrt{2+u^*} + \sqrt{u^*}) \cdot \dfrac{\Delta_2}{\varepsilon}$
- Code: https://github.com/BorjaBalle/analytic-gaussian-mechanism

Outline for today

How to make ML private?

- Making mean estimation private
- ML training
- Private ML training

Privacy vs. utility: mean



Binary Mean Estimation

Utility of exact mean

- We have n i.i.d samples $(x_1, \ldots, x_n)$ where $x_i \in \{0,1\}$.
- Estimate mean as $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$. What is the expected error?

Binary Mean Estimation

Utility of the private mean

- We have n i.i.d samples $(x_1, \ldots, x_n)$ where $x_i \in \{0,1\}$.
- Estimate mean as $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i + \text{Lap}(\Delta/\varepsilon)$. What is Δ?
- Net error is $\frac{1}{n} + \frac{2}{n^2\varepsilon^2}$.
- Error with Gaussian mechanism is similar.

Unbounded Mean Estimation

Utility of exact mean

- We have n i.i.d samples $(x_1, \ldots, x_n)$ with $E\|x_i\|_2^2 \leq \sigma^2$.
- Estimate mean as $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$. What is the expected error?
- What is the sensitivity?

Unbounded Mean Estimation

Bounding sensitivity

- We have n i.i.d samples $(x_1, \ldots, x_n)$ with $E\|x_i\|_2^2 \leq \sigma^2$.
- Let us clip x_i with a threshold of τ. The expected error is $\leq \frac{\sigma^4}{\tau^2} + \frac{\sigma^2}{n}$.

Unbounded Mean Estimation

Utility of private mean

- We have n i.i.d samples $(x_1, \ldots, x_n)$ with $E[x_i^2] \leq \sigma^2$.
- Output $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \text{clip}_\tau(x_i) + \text{Lap}(2\tau/n\varepsilon)$.

Theorem

$\hat{\mu}$ with $\tau = \sigma\sqrt{n\varepsilon}/2$ satisfies ε-DP and has an error

$$E[(\hat{\mu} - \mu)^2] \leq \frac{\sigma^2}{n} + \frac{4\sigma^2}{n\varepsilon}$$

Unbounded Mean Estimation

Utility of private mean

- We have n i.i.d samples $(x_1, \ldots, x_n)$ for $x_i \in \mathbb{R}^d$ with $E\|x_i\|_2^2 \leq \sigma^2$.
- Output $\frac{1}{n}\sum_{i=1}^{n} \text{clip}_\tau(x_i) + \mathcal{N}(0, \rho^2)$ for $\rho = 2\tau \log(2/\delta)/n\varepsilon$.
- Error?

Unbounded Mean Estimation

Utility of private mean

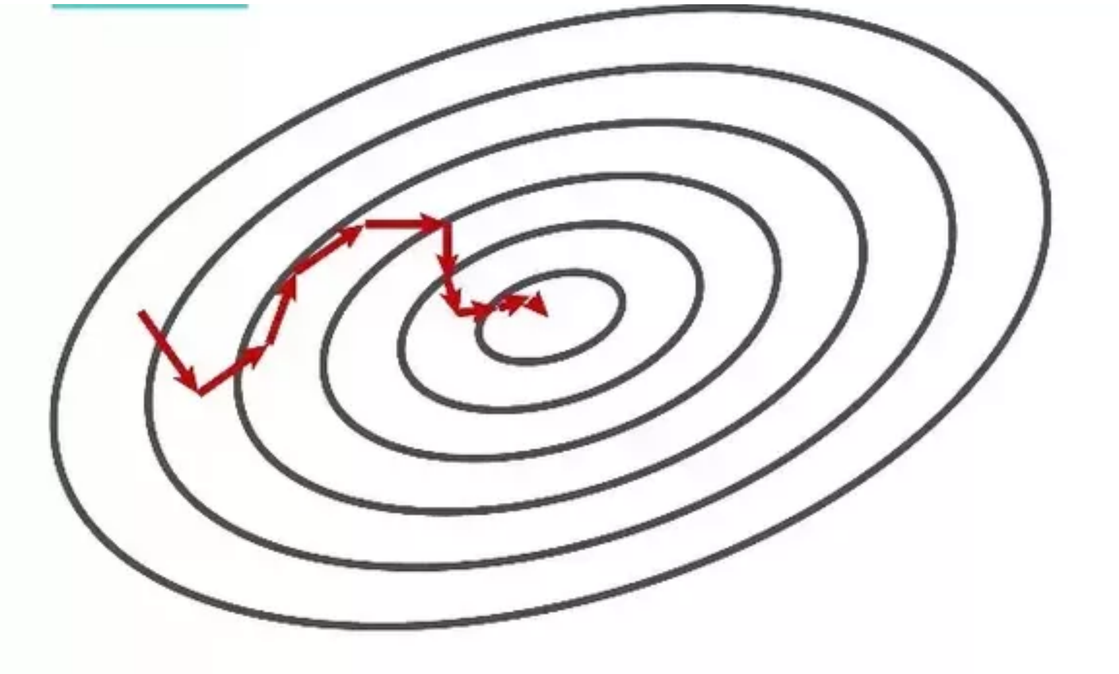
- We have n i.i.d samples $(x_1, \ldots, x_n)$ for $x_i \in \mathbb{R}^d$ with $E\|x_i\|_2^2 \leq \sigma^2$.
- Output $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \text{clip}_\tau(x_i) + \mathcal{N}(0, \rho^2)$ for $\rho = 2\tau\log(2/\delta)/n\varepsilon$.
- Error?

Theorem

$\hat{\mu}$ with $\tau = O(\sigma\sqrt{n\varepsilon}/d^{1/4})$ satisfies (ε, δ)-DP and has an error

$$E[(\hat{\mu} - \mu)^2] \leq O\left(\frac{\sigma^2}{n} + \frac{\sigma^2\sqrt{d}\log(1/\delta)}{n\varepsilon}\right)$$

Training ML models: GD



Machine Learning

How to train a model?

- We are given i.i.d data: $(x_1, y_1), \ldots, (x_n, y_n)$.
- We have a parameterized family of predictors $f(x; \theta) : \mathcal{X} \to \mathcal{Y}$.
    - Linear models $f(x; \theta) = \theta^\top x$
    - Neural Networks $f(x; \theta) = \theta_2^\top \cdot \text{Relu}(\theta_1^\top x)$

Machine Learning

How to train a model?

- We are given i.i.d data: $(x_1, y_1), \ldots, (x_n, y_n)$.
- We have a parameterized family of predictors $f(x; \theta) : \mathcal{X} \to \mathcal{Y}$.
    - Linear models $f(x; \theta) = \theta^\top x$
    - Neural Networks $f(x; \theta) = \theta_2^\top \cdot \text{Relu}(\theta_1^\top x)$
- We want to find parameters which minimizes test-loss $L(\theta) = E_{(x,y)}[\ell(f(x; \theta), y)]$
-

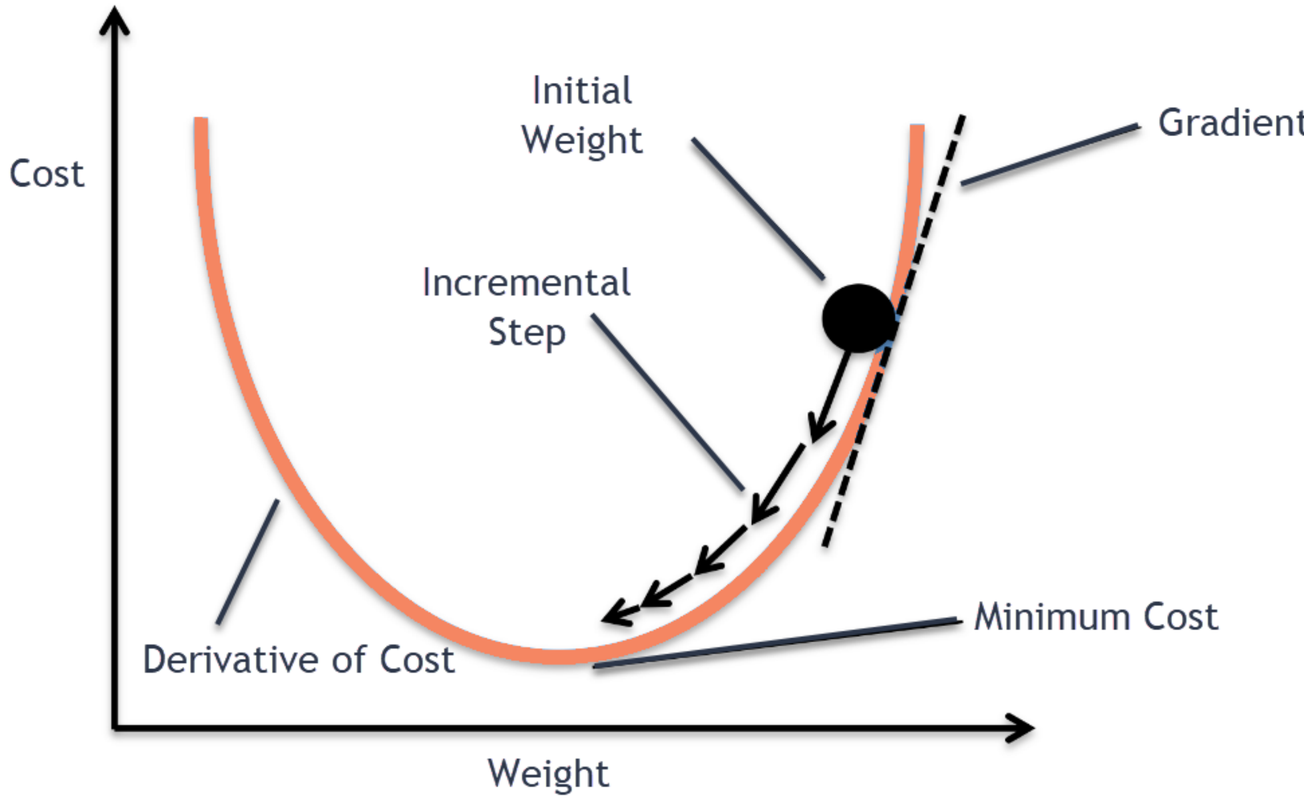
Machine Learning

How to train a model?

- We are given i.i.d data: $(x_1, y_1), \ldots, (x_n, y_n)$.
- We have a parameterized family of predictors $f(x; \theta) : \mathcal{X} \to \mathcal{Y}$.
    - Linear models $f(x; \theta) = \theta^\top x$
    - Neural Networks $f(x; \theta) = \theta_2^\top \cdot \text{Relu}(\theta_1^\top x)$
- We want to find parameters which minimizes test-loss $L(\theta) = E_{(x,y)}[\ell(f(x; \theta), y)]$
- We instead minimize training loss $\hat{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} [\ell(f(x_i; \theta), y_i)]$

Understanding Gradient Descent

- We want to minimize our function $L(\theta)$
- Iterative algorithm. Starting from θ_t in step t,

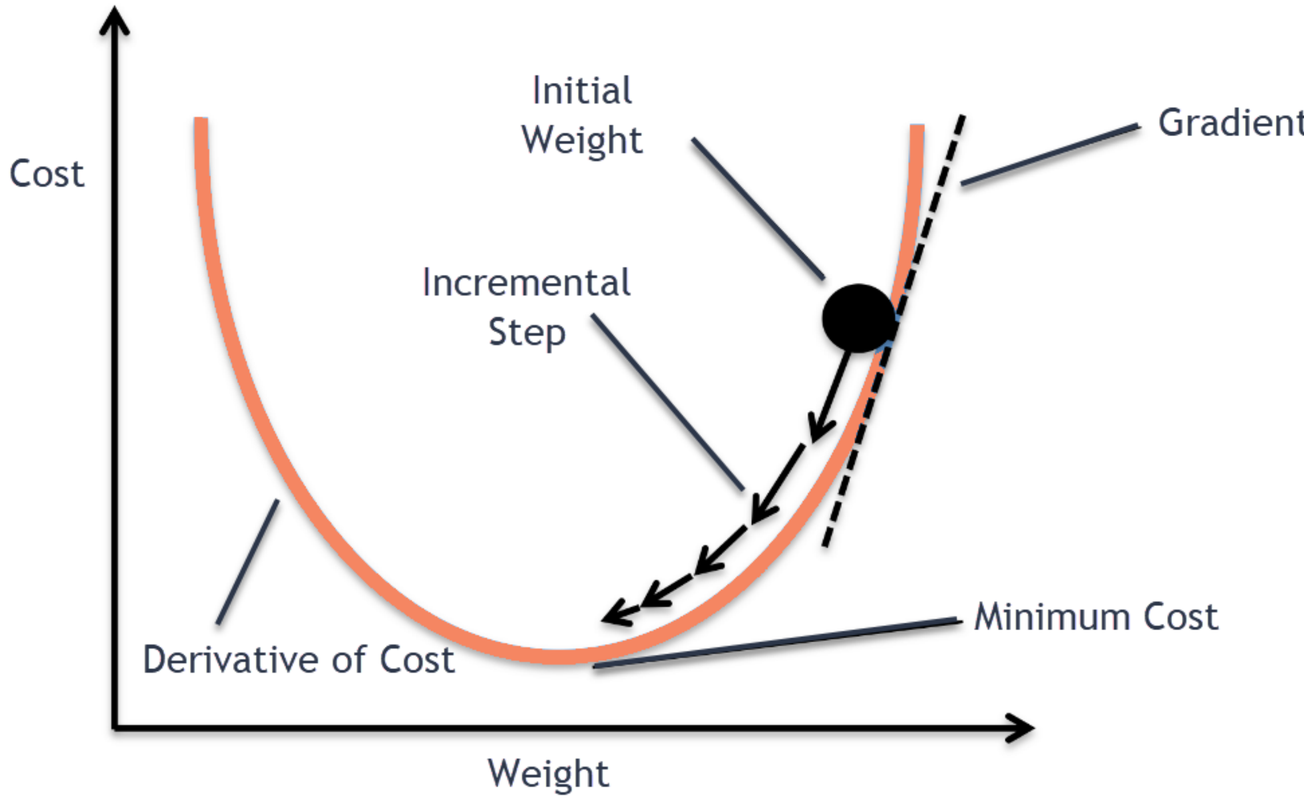


Understanding Gradient Descent

- We want to minimize our function $L(\theta)$
- Iterative algorithm. Starting from θ_t in step t,
- we create a local approximation

$$L(\theta_t + \Delta\theta) \approx L(\theta_t) + \nabla L(\theta_t)^\top \Delta\theta$$

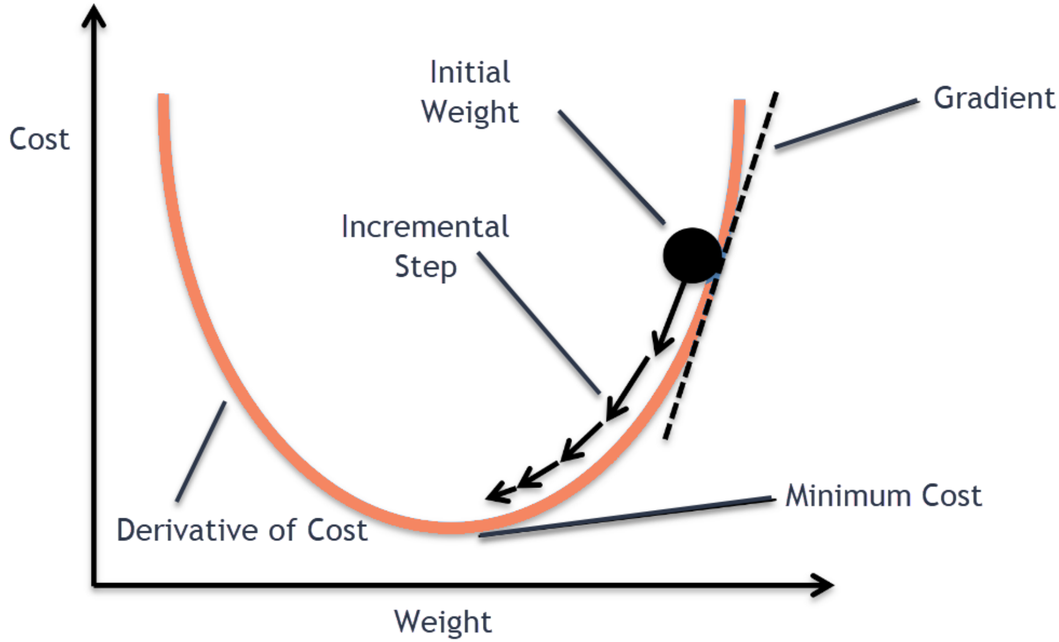
- Move along “steepest” descent direction.

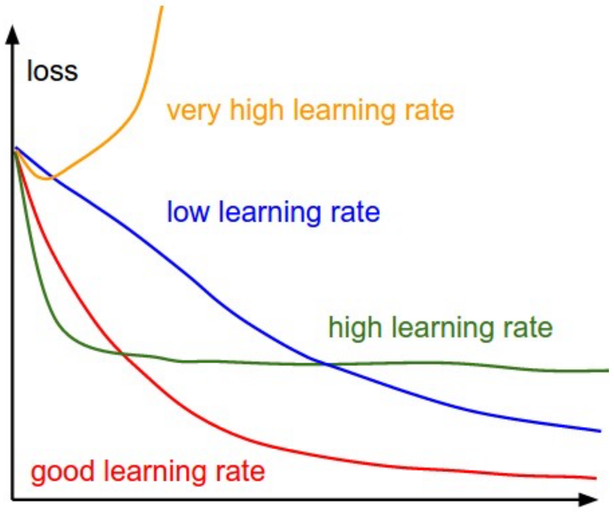


Understanding Gradient Descent

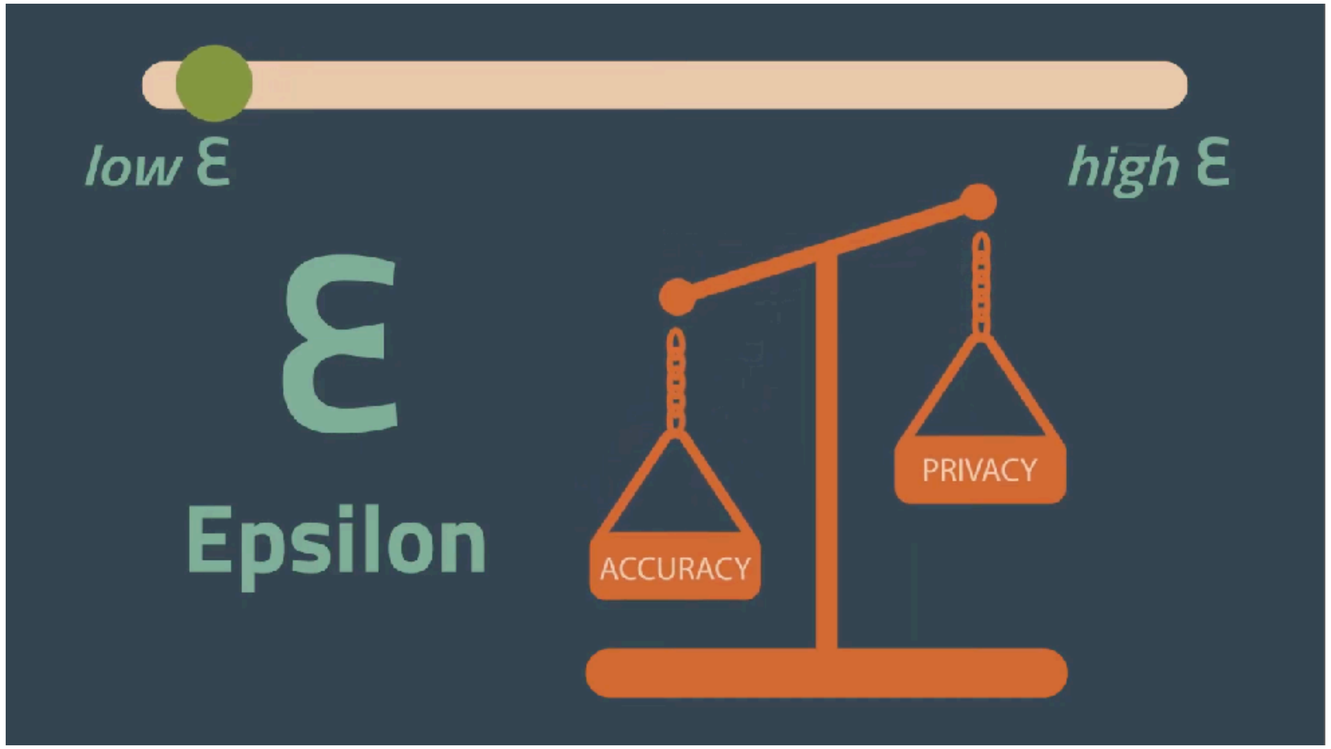
Algorithm

- Initialize θ_0
- For t=1, …, T
  - $\theta_t = \theta_{t-1} - \gamma_t \nabla L(\theta_{t-1})$
- How to decide γ_t?





Making Gradient Descent Private: Composition



Private full-batch gradient descent

Algorithm

- Starting from θ_0, at each time step we update
  - $\theta_t = \theta_{t-1} - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \ell(f(x_i; \theta), y_i)$
- To make it private
  - $\theta_t = \theta_{t-1} - \gamma \frac{1}{n} \sum_{i=1}^{n} \text{Clip}_\tau \left(\nabla_\theta \ell(f(x_i; \theta), y_i) \right) + \text{noise}$
  - Assume scalar for now. So noise = $Lap(??)$

Private full-batch gradient descent

One-step privacy

- Suppose we just run step of
  $\theta_t = \theta_{t-1} - \gamma \frac{1}{n} \sum_{i=1}^{n} \text{Clip}_\tau \left(\nabla_\theta \ell(f(x_i; \theta), y_i) \right) + Lap(??)$
- Sensitivity? How much noise?

- How to reason about what happens across time steps?

Post-processing and composition

Post-processing

- You can never undo the output of a DP-algorithm

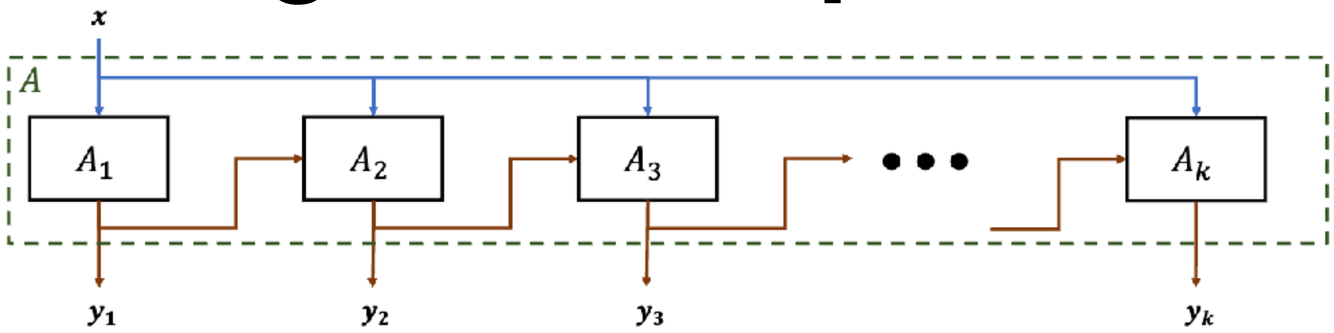
Theorem

$A : \mathcal{X}^n \to \mathbb{R}^d$ is a (ε, δ)-DP algorithm and f is a mapping independent of $\mathcal{X}$, then $f \circ A$ is (ε, δ)-DP

- Upshot: we can plug in our private gradients into any optimizer (e.g. AdamW).

Post-processing and composition

Composition



- What if the new function also depends on our data?

Theorem

$A : \mathcal{X}^n \to \mathbb{R}^d$ is a $(\varepsilon_1, 0)$-DP algorithm and
$B : \mathcal{X}^n \to \mathbb{R}^d$ is a $(\varepsilon_2, 0)$-DP algorithm, then
$(A, B) : \mathcal{X}^n \to \mathbb{R}^d \times \mathbb{R}^d$ is $(\varepsilon_1 + \varepsilon_2, 0)$-DP

Private full-batch gradient descent

Multi-step privacy

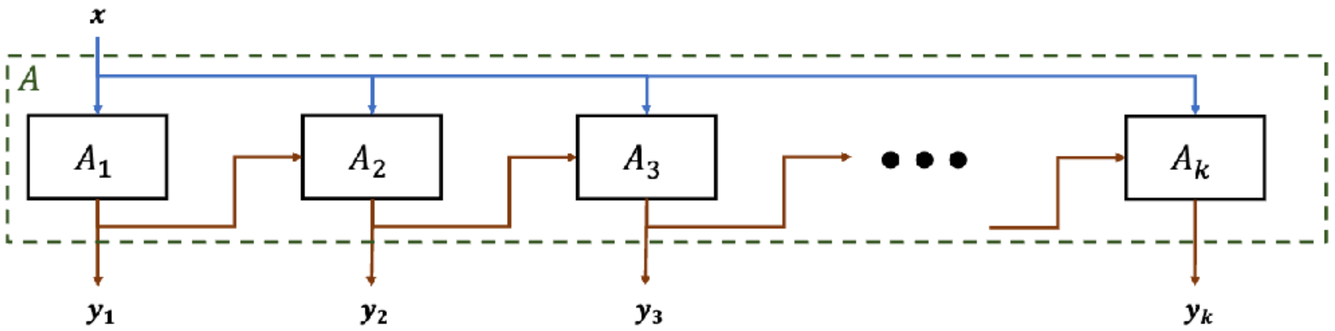
- One step is $(\varepsilon,0)$-DP
  $\theta_t = \theta_{t-1} - \gamma \frac{1}{n} \sum_{i=1}^{n} \text{Clip}_\tau \left(\nabla_\theta \ell(f(x_i;\theta), y_i) \right) + Lap(2\tau/n\varepsilon)$

- k-steps of full-batch gradient descent is $(k\varepsilon,0)$-DP.

- We can do better!

Private full-batch gradient descent

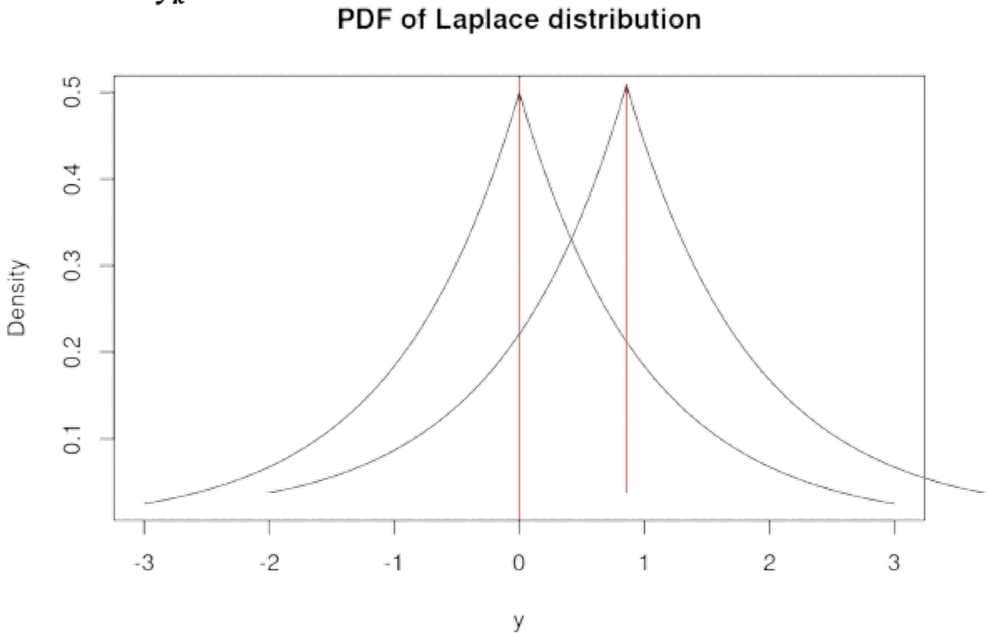
Advanced composition



- Let us compute the privacy random variable:

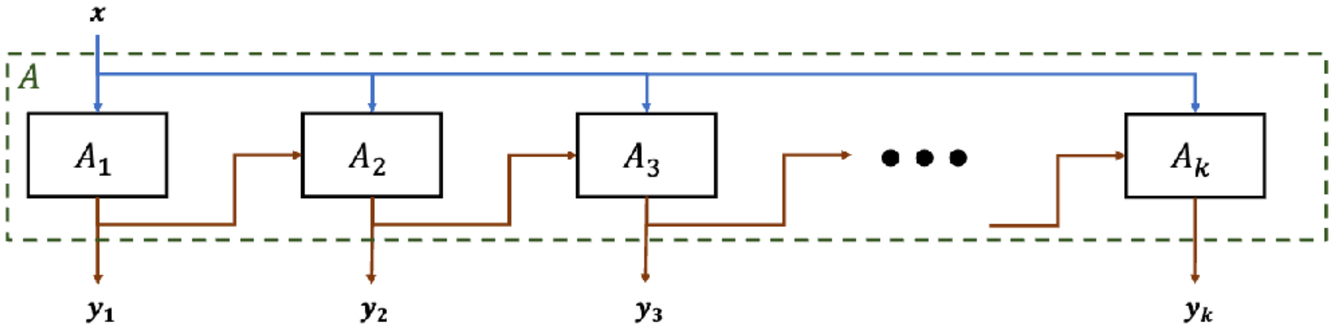
$$R = \log\left(\frac{Pr[A(D) = t]}{Pr[A(D') = t]}\right) \qquad \text{for } t \sim A(D)$$

- $R \in [-\varepsilon, \varepsilon]$ and has mean 0.



Private full-batch gradient descent

Advanced composition



- Privacy random variable of composition:

$$R = \sum_{i=1}^{k} \log\left(\frac{Pr[A_i(D) = t_i]}{Pr[A_i(D') = t_i]}\right) = \sum_{i=1}^{k} R_i$$

- $R_i \in [-\varepsilon, \varepsilon]$, 0-mean, conditionally independent.

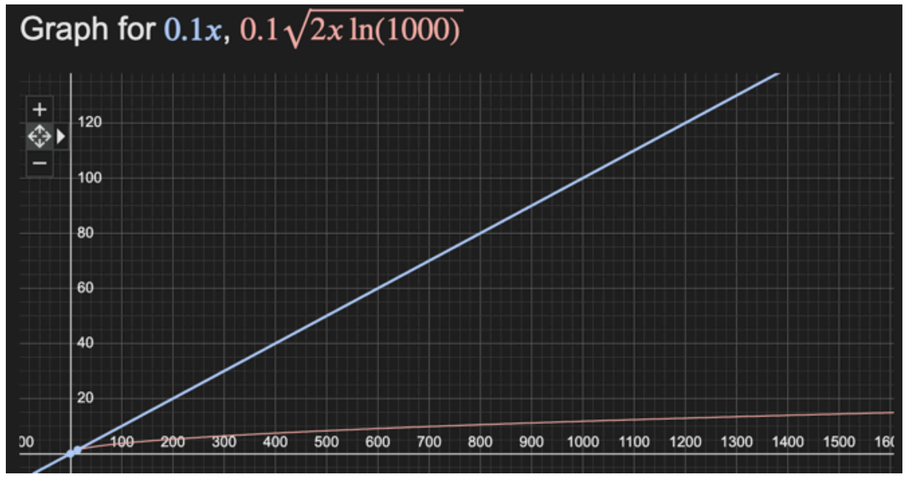
Private full-batch gradient descent

Aside: Azuma's inequality

Azuma's inequality

Given $X_1, \ldots, X_n$ where $E[X_i \mid \text{past}] = 0,\ |X_i| \leq \varepsilon_i$.
Then,

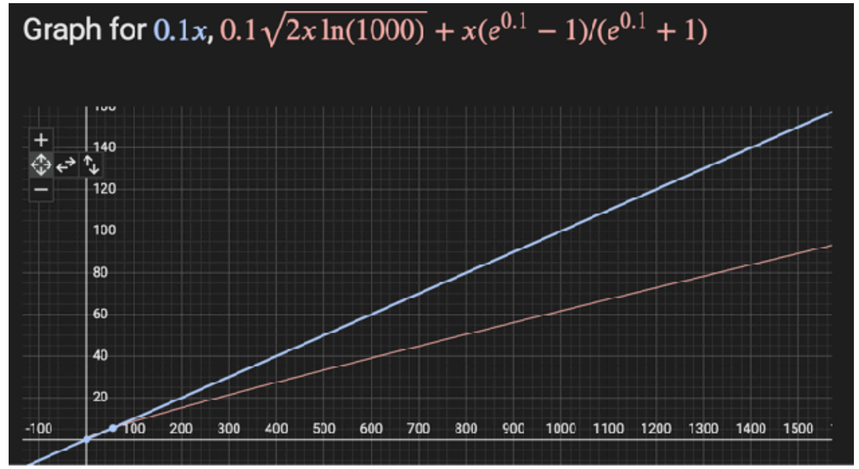
$$Pr[\sum_{i=1}^{k} X_i \geq \Delta] \leq \exp(-\Delta^2/2 \sum_{i=1}^{k} \varepsilon_i^2)$$



- $R_i \in [-\varepsilon, \varepsilon]$, 0-mean, conditionally independent.
- $Pr[\sum_{i=1}^{k} R_i \geq \varepsilon\sqrt{2k \log(1/\delta)}] \leq \delta$ i.e. we have $(\varepsilon\sqrt{2k \ln(1/\delta)}, \delta)$-DP!

Private full-batch gradient descent

Advanced composition



Theorem. Advanced Composition

A combination of $A_1 \circ A_2 \circ A_k$, each of which is (ε, δ)-DP is $(\tilde{\varepsilon}, \tilde{\delta})$-DP where

$$\tilde{\varepsilon} = \varepsilon\sqrt{2k\ln(1/\delta')} + k\frac{e^{\varepsilon}-1}{e^{\varepsilon}+1} \quad \text{and} \quad \tilde{\delta} = k\delta + \delta'$$

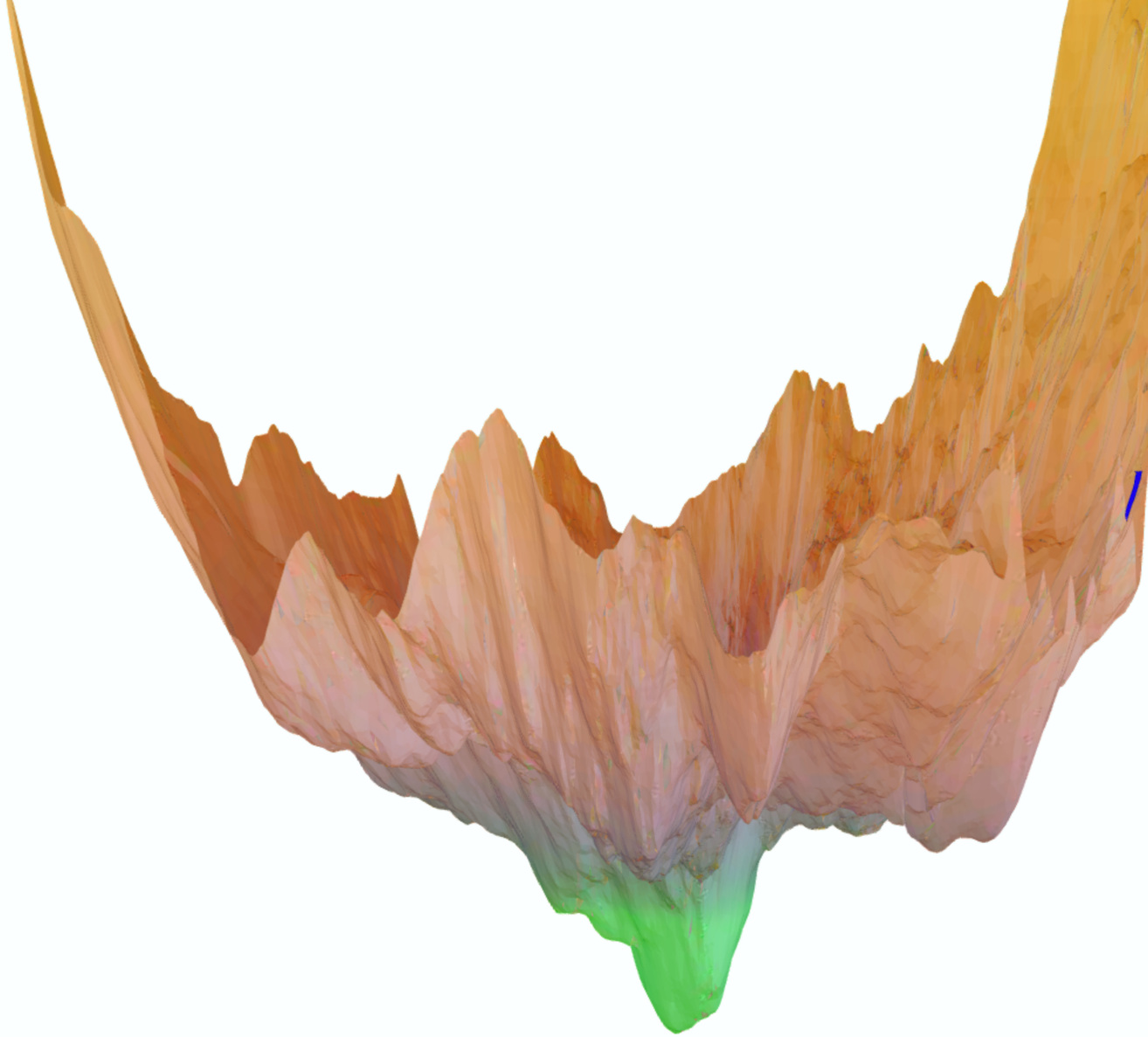
For any choice of δ'.

Private full-batch gradient descent

Multi-step privacy

- One step is $(\varepsilon, 0)$-DP
  $\theta_t = \theta_{t-1} - \gamma \frac{1}{n} \sum_{i=1}^{n} \text{Clip}_\tau \left(\nabla_\theta \ell(f(x_i; \theta), y_i) \right) + Lap(2\tau / n\varepsilon)$
- k-steps of full-batch gradient descent is $(\varepsilon\sqrt{2k \ln(1/\delta)}, \delta)$-DP.
- How about with Gaussian-noise and vectors?

Optimization for Deep Learning



Stochastic Gradient Descent

Convergence analysis

- How do we compute $\nabla L(\theta_t)$?
- We are only given data samples: $(x_1, y_1), \ldots$ i.e. we cannot compute $L(\theta) = E_{(x,y)}[\ell(f(x;\theta), y)]$

Stochastic Gradient Descent

Convergence analysis

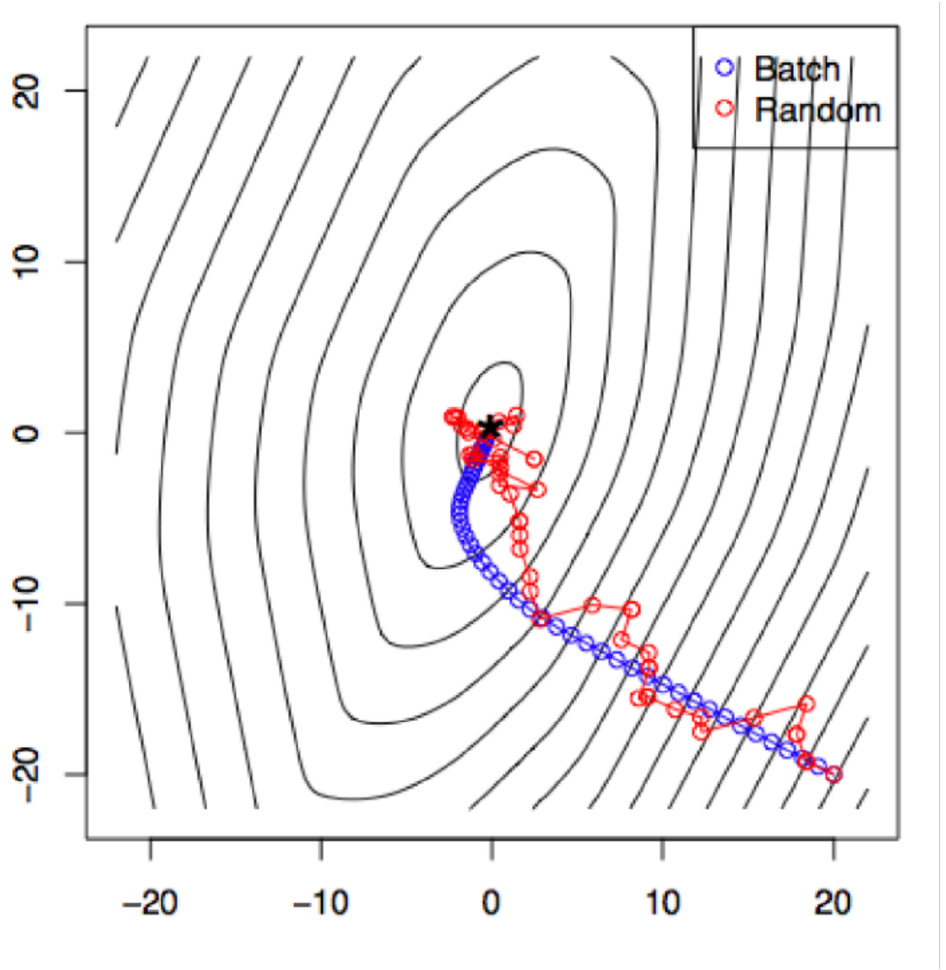
- How do we compute $\nabla L(\theta_t)$?
- We are only given data samples: $(x_1, y_1), \ldots$ i.e. we cannot compute $L(\theta) = E_{(x,y)}[\ell(f(x;\theta), y)]$
- SGD says no problem. Just use sample gradient. Initialize θ_0
- For t=1, …, T
    - Sample a data point (x_t, y_t)
    - $\theta_t = \theta_{t-1} - \gamma_t \nabla_\theta \ell(f(x_t;\theta_{t-1}), y_t) = \theta_{t-1} - \gamma_t \nabla \ell_t(\theta_{t-1})$

Gradient Descent Variants

- we are given n samples $(x_1, y_1), \ldots, (x_n, y_n)$
- We have a few options:
  - Exact gradient: $\nabla_\theta E_{x,y}[\ell(f(x;\theta), y)]$
  - Stochastic gradient: for a random sample (x_i, y_i), $\nabla_\theta \ell(f(x_i;\theta), y_i)$
  - Full-batch gradient: $\frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \ell(f(x_i;\theta), y_i)$
  - Mini-batch gradient: for a sample $\mathcal{B}$, $\frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} \nabla_\theta \ell(f(x_i;\theta), y_i)$

Understanding Gradient Descent

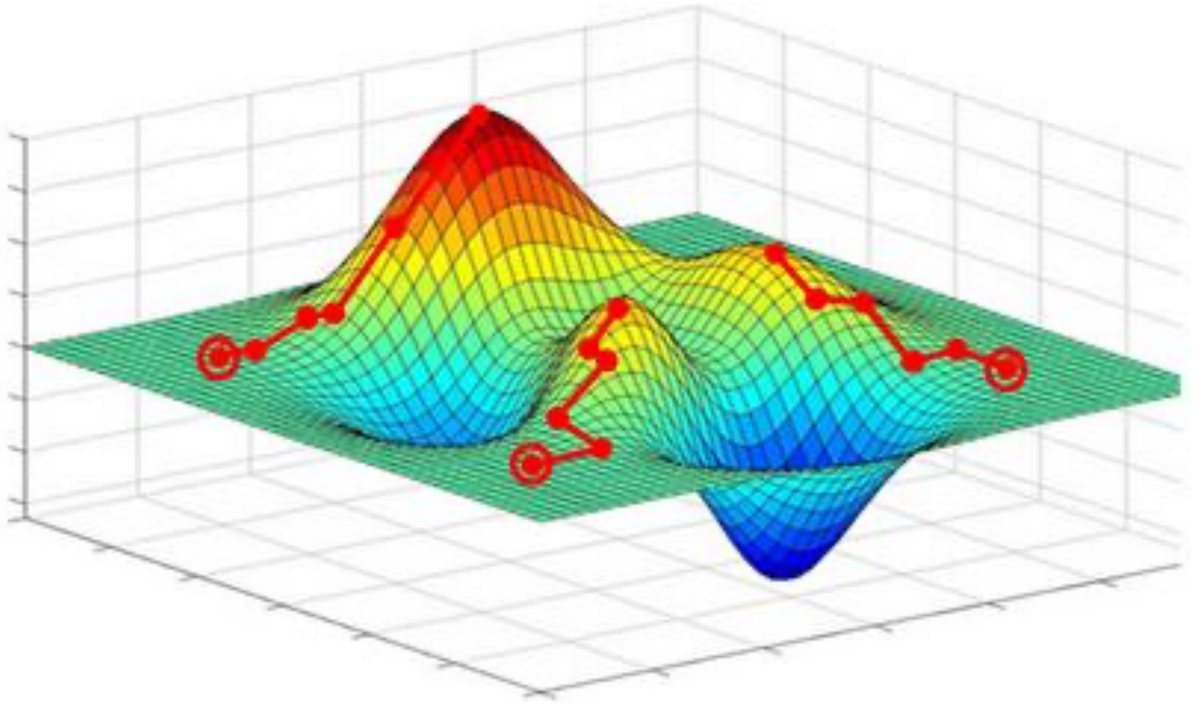
Convergence analysis



Optimization in Deep Learning

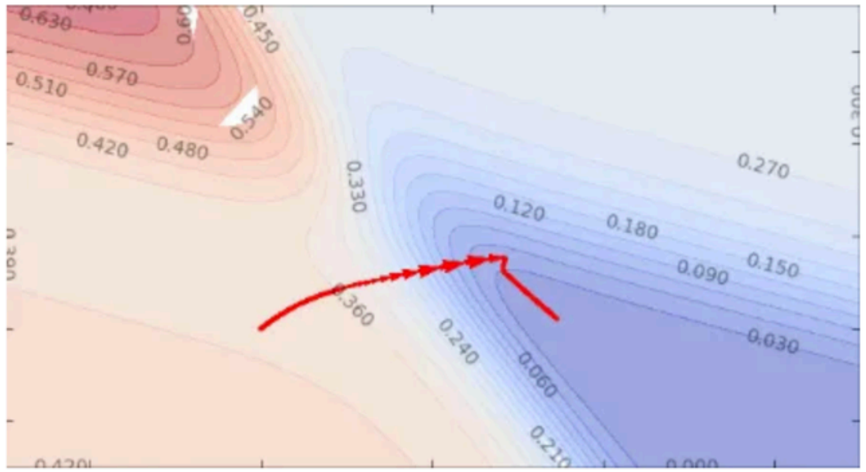
Initialization

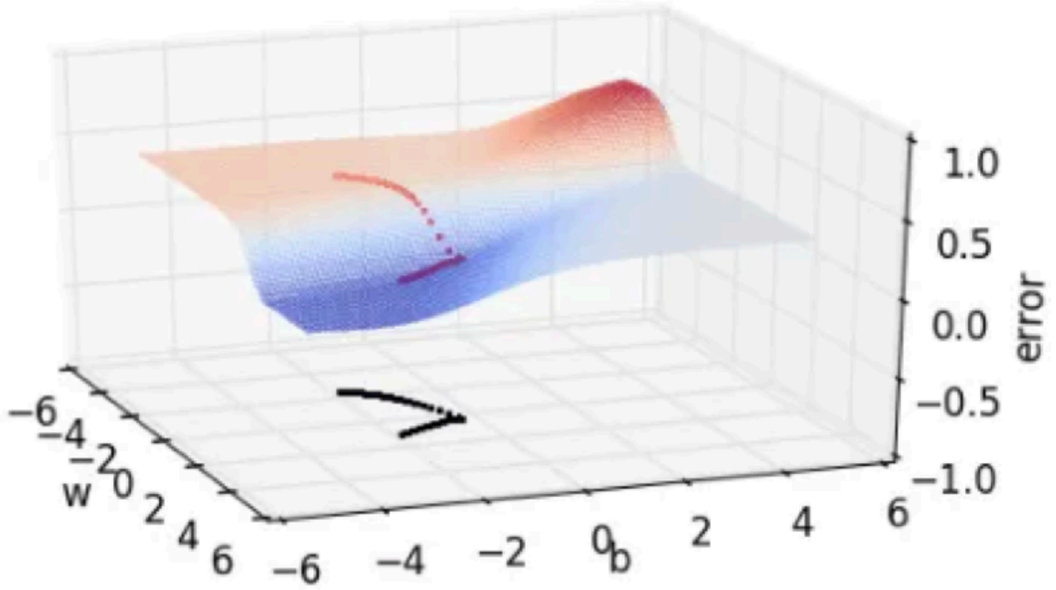
- Initialization matters!
- Always start with a pretrained model if you can.



Optimization in Deep Learning

Momentum





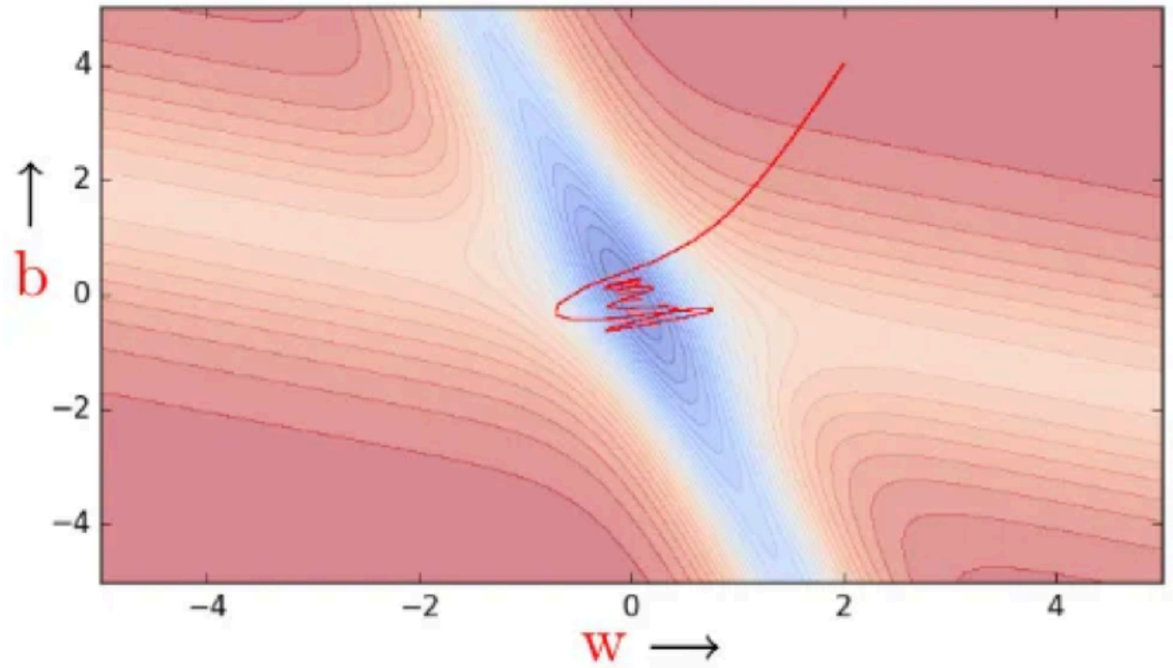
Contour Map with GD applied on it. Arrows represent faster fall in error value. Flat surfaces represented by constant color regions.

Gradient descent slows down a lot when it encounters large flat sections

Optimization in Deep Learning

Momentum

- Add momentum to speed it up
- $m_t = m_{t-1} + \beta \nabla L(\theta_{t-1})$
- $\theta_t = \theta_{t-1} - \gamma m_t$



It oscillates in the valley of minima. Take lot U-turns, still converges faster than Vanilla GD.

Physical intuition - biking down a hill is faster than walking down

Optimization in Deep Learning

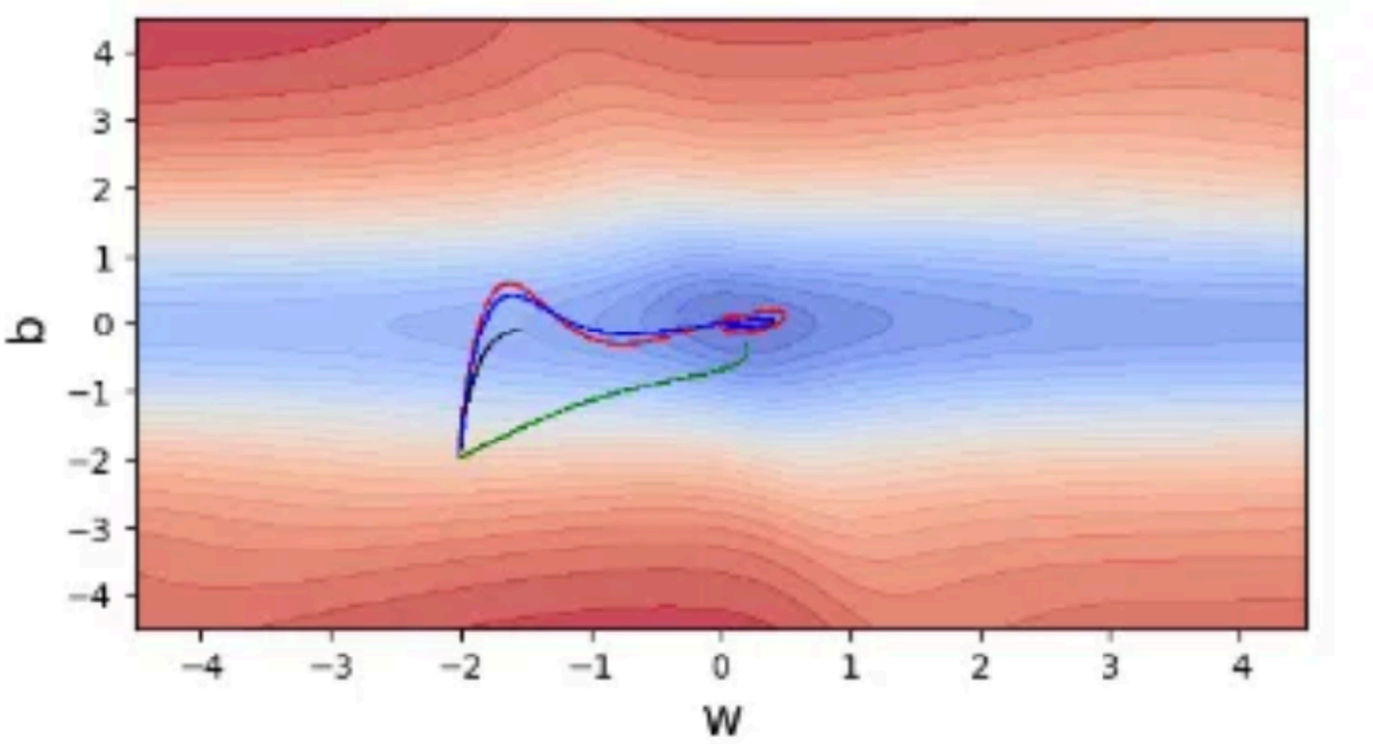
Adaptivity

- We need to keep changing step size since smoothness keeps changing all the time
- Make the step-size adaptive - AdaGrad
- $v_t = v_{t-1} + \beta_2(\nabla L(\theta_{t-1}))^2$ - running estimate of second moment
- $\theta_t = \theta_{t-1} - \gamma \nabla L(\theta_{t-1}) / \sqrt{v_t + \epsilon}$ - normalize the updates.

Tran et al. 2024 “Empirical Tests of Optimization Assumptions in Deep Learning”

Optimization in Deep Learning

AdaGrad

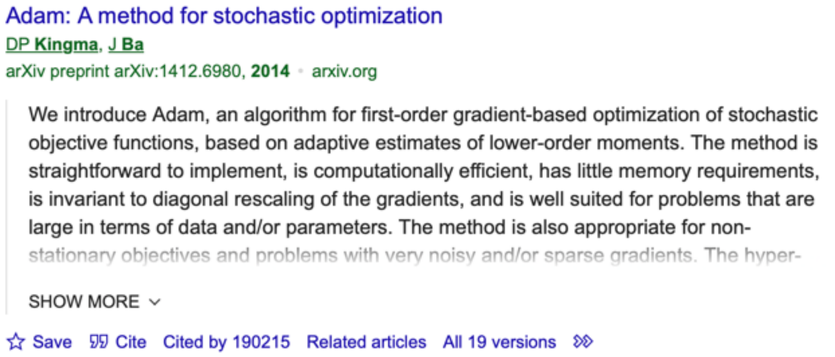


- Progress of AdaGrad in green is much more consistent across steps.
- But it is slower than momentum methods (blue / red)

Optimization in Deep Learning

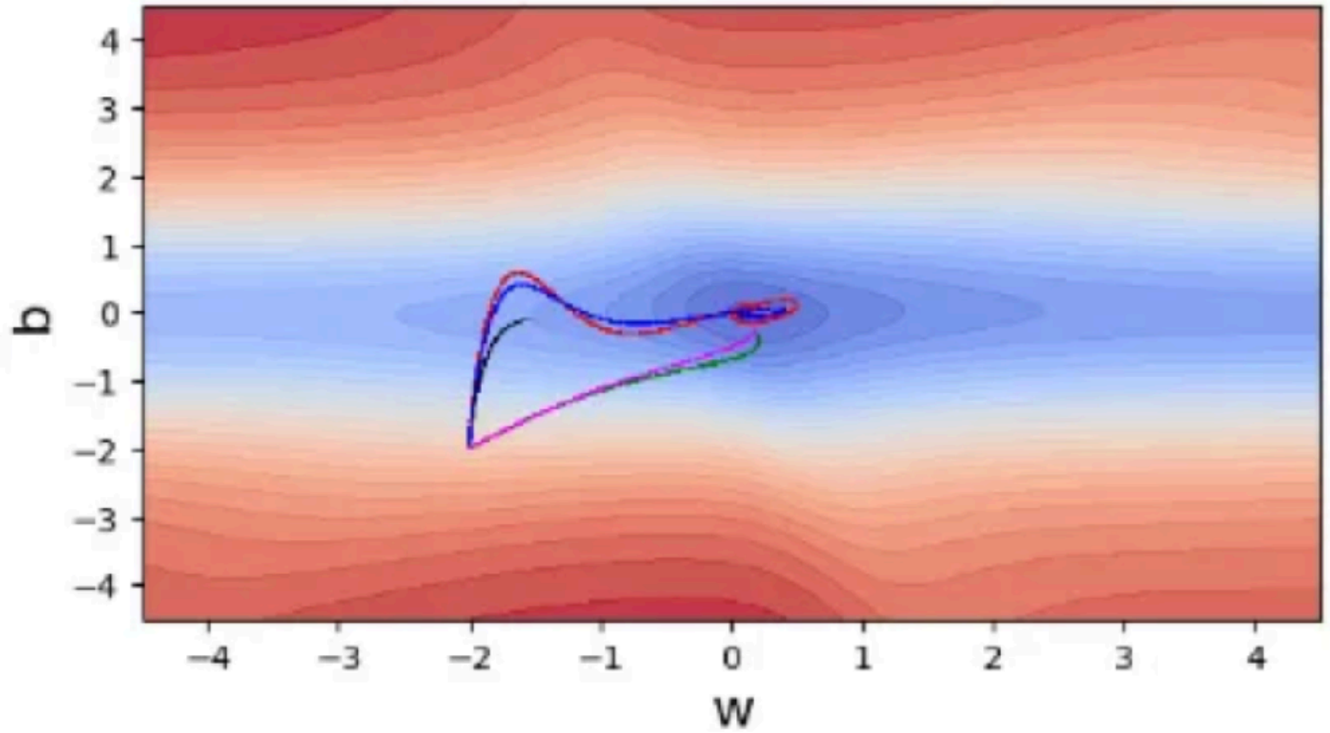
Adam

- Combine everything - adaptivity + momentum = Adam
- $m_t = m_{t-1} + \beta \nabla L(\theta_{t-1} - \gamma m_{t-1})$ - first moment estimate
- $v_t = v_{t-1} + \beta_2 (\nabla L(\theta_{t-1}))^2$ - second moment estimate
- $\theta_t = \theta_{t-1} - \gamma m_t / \sqrt{v_t + \epsilon}$



Optimization in Deep Learning

Adam



- AdamW (a minor variant) is likely the best default optimizer
- In LLMs, additionally learning rate warm up is used for 5 epochs.

What about privacy?

Can we make SGD private?

- For t=1, …, T
  - Sample a data point (x_t, y_t)
  - $\theta_t = \theta_{t-1} - \gamma_t \nabla_\theta \ell(f(x_t; \theta_{t-1}), y_t) = \theta_{t-1} - \gamma_t \nabla \ell_t(\theta_{t-1})$

Private stochastic gradient descent

Algorithm

- Starting from θ_0, at each time step
  - sample (x_i, y_i) randomly from $(x_1, y_1), \ldots, (x_n, y_n)$
  - $\theta_t = \theta_{t-1} - \gamma \nabla_\theta \ell(f(x_i; \theta), y_i)$
- To make it private
  - $\theta_t = \theta_{t-1} - \gamma \text{Clip}_\tau \left(\nabla_\theta \ell(f(x_i; \theta), y_i) \right) + \text{noise}$
  - Assume scalar for now. So noise = $Lap(??)$

Private stochastic gradient descent

One-step privacy

- Suppose we just run step:
  - $\theta_t = \theta_{t-1} - \gamma \text{Clip}_\tau \left(\nabla_\theta \ell(f(x_t; \theta), y_t) \right) + Lap(2\tau/\varepsilon)$
- No improvement due to n
- Important note: use poisson sampling! Not uniform.
- This makes analyzing what happens to each data-point independent.

Privacy amplification via subsampling

- Given a dataset $D \in \mathcal{X}^n$, and $m \in [n]$
- We define S to be a random m-subsample of D
- Is releasing S private?
- Now suppose A is ε-DP on D. What is the privacy of A composed with subsampling?

Privacy amplification via subsampling

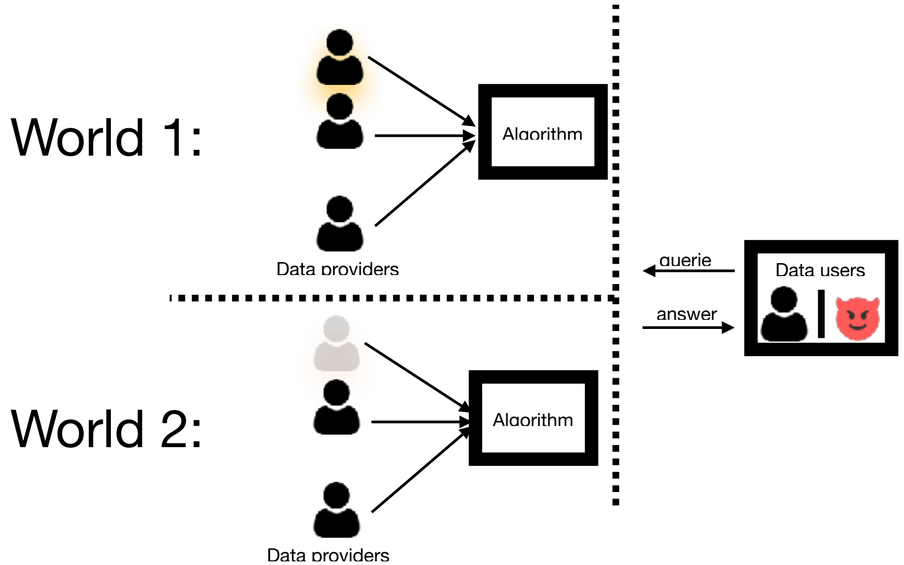
Theorem. Subsampling Amplification

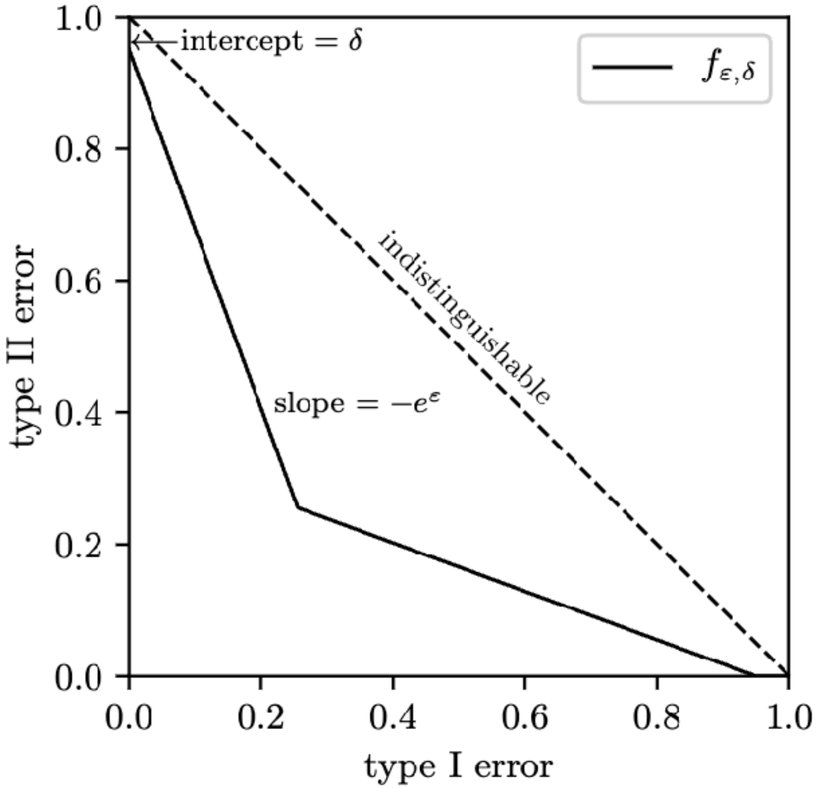
Composing an (ε, δ)-DP A with a sampling rate of q results in an $(\tilde{\varepsilon}, \tilde{\delta})$-DP algorithm where

$$\tilde{\varepsilon} = \log(1 - q + qe^{\varepsilon}) = O(q\varepsilon) \quad \text{and} \quad \tilde{\delta} = q\delta$$

Recall

Membership Inference definition of privacy





- Claim: $\beta + (1 - q + qe^{\varepsilon})\alpha \geq 1 - \delta$
- and, $(1 - q + qe^{\varepsilon})\beta + \alpha \geq 1 - \delta$, where $\alpha =$ type I error, $\beta =$ type II error

Private stochastic gradient descent

One-step privacy

- Suppose we just run step:
  - $\theta_t = \theta_{t-1} - \gamma \text{Clip}_\tau \left(\nabla_\theta \ell(f(x_t; \theta), y_t) \right) + Lap(2\tau/\varepsilon)$
- We have $q = 1/n$. So, we have $\tilde{\varepsilon} = \log(1 - 1/n + e^\varepsilon/n) = O(\varepsilon/n)$
- Adding in advanced composition, k rounds of SGD satisfies $(O(\varepsilon/n\sqrt{k \ln(1/\delta)}), \delta)$-DP

Private stochastic gradient descent

One-step privacy

- Suppose we just run step:
  - $\theta_t = \theta_{t-1} - \gamma \text{Clip}_\tau \left(\nabla_\theta \ell(f(x_t; \theta), y_t) \right) + Lap(2\tau/\varepsilon)$
- We have $q = 1/n$. So, we have $\tilde{\varepsilon} = \log(1 - 1/n + e^\varepsilon/n) = O(\varepsilon/n)$
- Adding in advanced composition, k rounds of SGD satisfies $(O(\varepsilon/n\sqrt{k \ln(1/\delta)}), \delta)$-DP
- Hyper-parameter - tune sampling rate q in practice