CSCI 699: Privacy Preserving Machine Learning - Week 3

Algorithms for Differentially Privacy and Machine Learning

Sai Praneeth Karimireddy, Sep 13 2024

Recap

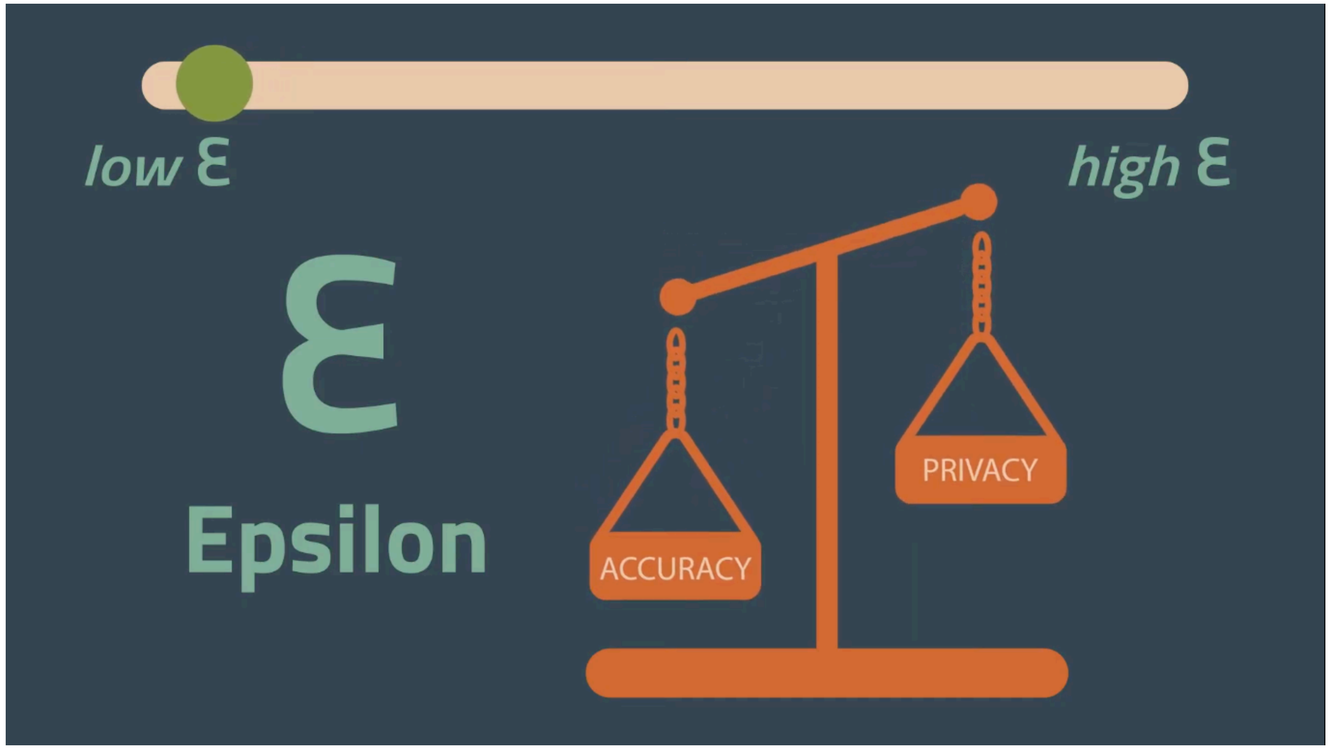
- Differential privacy
  - $\forall y, \forall$ similar D, D' : $\dfrac{\Pr[Y = y \,|\, \mathscr{D} = D]}{\Pr[Y = y \,|\, \mathscr{D} = D']} \leq e^{\varepsilon}$
  - connection to tradeoff curves of attacker
  - ℓ_1-sensitivity and Laplace mechanism
- Gaussian Mechanism & Approximate Differential Privacy
- f-DP and Gaussian-DP

Outline for today

How to make ML private?

- Deep dive into approximate DP
- Making mean estimation private
- ML training

Approximate Differential Privacy



Approximate Differential Privacy

Original definition

(ε, δ)-Differential Privacy:

An algorithm A satisfies (ε, δ)-DP if for any similar datasets $D, D' \in \chi^n$ and $y \in \mathcal{Y}$

$$Pr[A(D) = y] \leq Pr[A(D') = y] \cdot \exp(\varepsilon) + \delta$$

Approximate Differential Privacy

Privacy Loss Variable Reformulation

Lemma 3.17 [Dwork and Roth 2014]

Let us draw a variable $t \sim A(D)$. Then the privacy loss random variable

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

An algorithm A satisfies (ε, δ)-DP iff for any similar/neighboring datasets $D, D' \in \chi^n$ we have

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

Approximate Differential Privacy

Selecting δ: rare terrible events

- A(D) = outputs $\emptyset$ with probability $1 - \delta$,
  outputs D with prob δ
- Is this private?

Approximate Differential Privacy

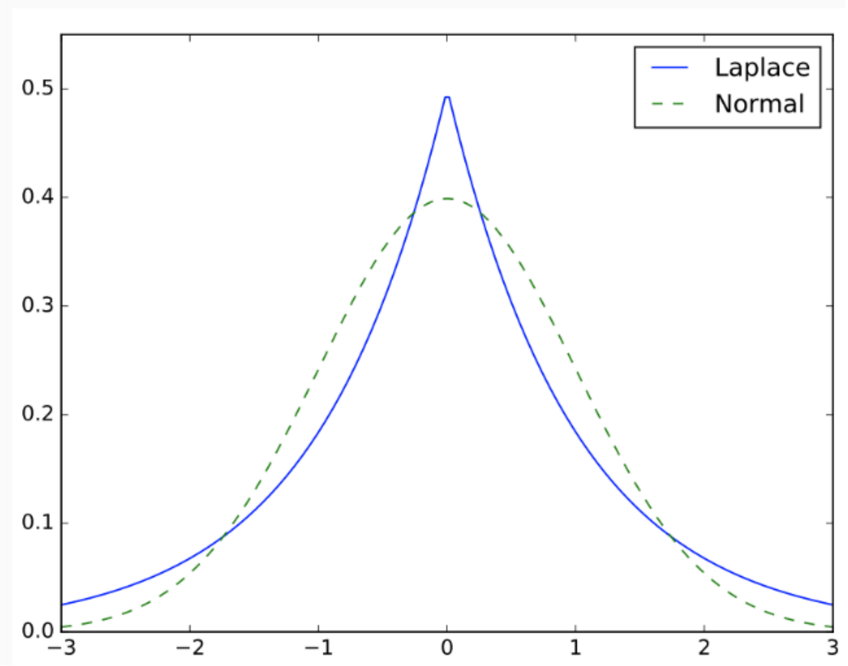
Selecting δ: name and shame

- $D \in \chi^n$ has n data points.
- A(D) for each $x_i \in D$,
    - outputs x_i with probability δ
- Is A private?

Approximate Differentially Private Algorithms

Gaussian mechanism

- Suppose A(D) = 0, A(D') = 1. Release $\hat{y} = y + \text{Gaussian}(0, \sigma^2)$
- $z \sim \text{Gaussian}(\mu, \sigma^2) \Rightarrow p(z) \propto \frac{1}{\sigma} e^{-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2}$
- What value of σ should we pick to satisfy (ε, δ)-DP?



Approximate Differentially Private Algorithms

Gaussian mechanism

Approximate Differentially Private Algorithms

Gaussian mechanism

Theorem

Suppose $f : \chi^n \to \mathbb{R}^d$ is Δ_2-sensitive wrt ℓ_2. Then, the following A satisfies (ε, δ)-DP:

- $A(D) = f(D) + \mathcal{N}(0, \rho^2 I)$
- with $\rho = \dfrac{\Delta_2 \sqrt{2 \log(2/\delta)}}{\varepsilon}$

Approximate Differentially Private Algorithms

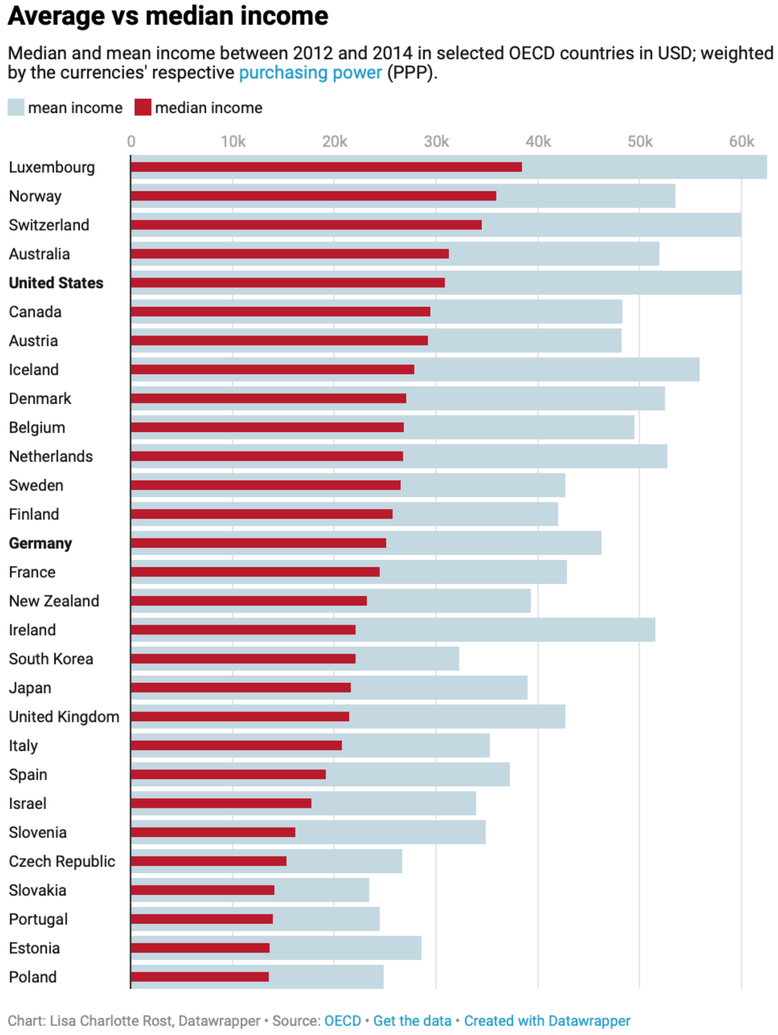
Gaussian mechanism

An Aside: beyond global sensitivity

Mean vs median

Mean:
21 Dollars / 7 people
= 3 Dollars per person

Median:
Person 4 owns 2 Dollars.
= 2 Dollars per person



An Aside: beyond global sensitivity

Mean vs median

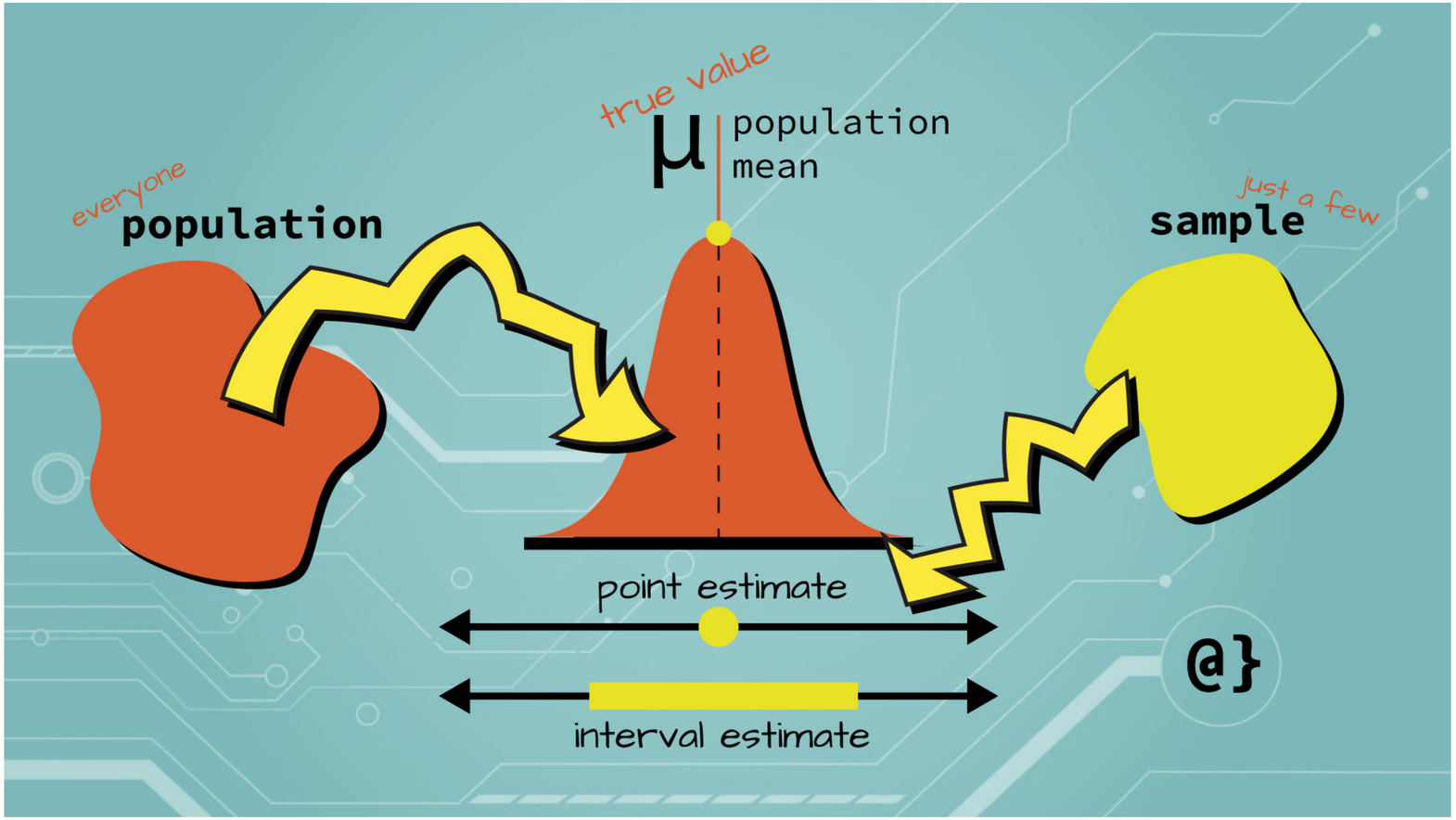
- We have dataset $D = (x_1, \ldots, x_n)$
- How sensitive is counting (e.g. number of households below the poverty line)?
- How sensitive is the mean (avg household income)?

An Aside: beyond global sensitivity

Mean vs median

- We have dataset $D = (x_1, \ldots, x_n)$
- How sensitive is the median?

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

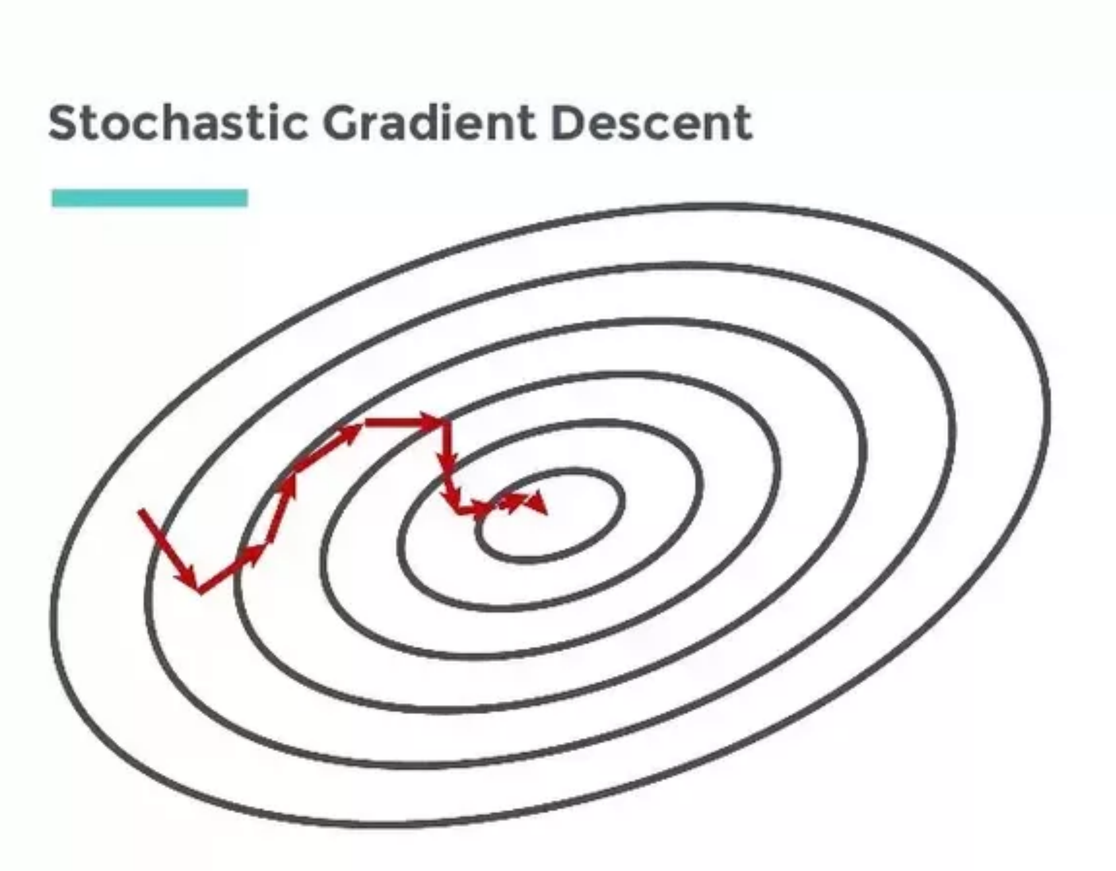
- We have n i.i.d samples $(x_1, \ldots, x_n)$ for $x_i \in \mathbb{R}^d$ with $E\|x_i\|_2^2 \leq \sigma^2$.
- Output $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} \text{clip}_\tau(x_i) + \mathcal{N}(0, \rho^2)$ for $\rho = 2\tau\log(2/\delta)/n\varepsilon$.
- Error?

Theorem

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

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

Optimization for Machine Learning



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 $L(\theta) = E_{(x,y)}[\ell(f(x; \theta), y)]$
-

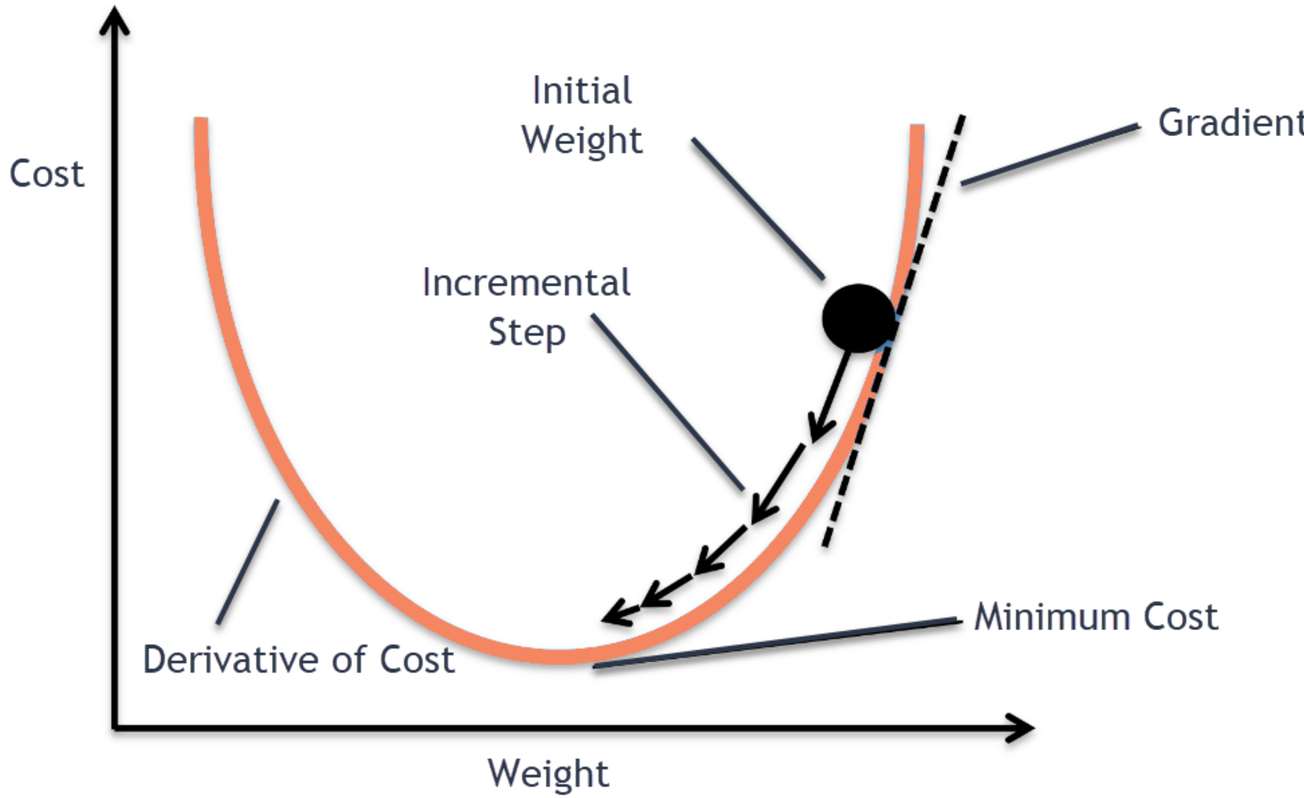
Machine Learning

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,

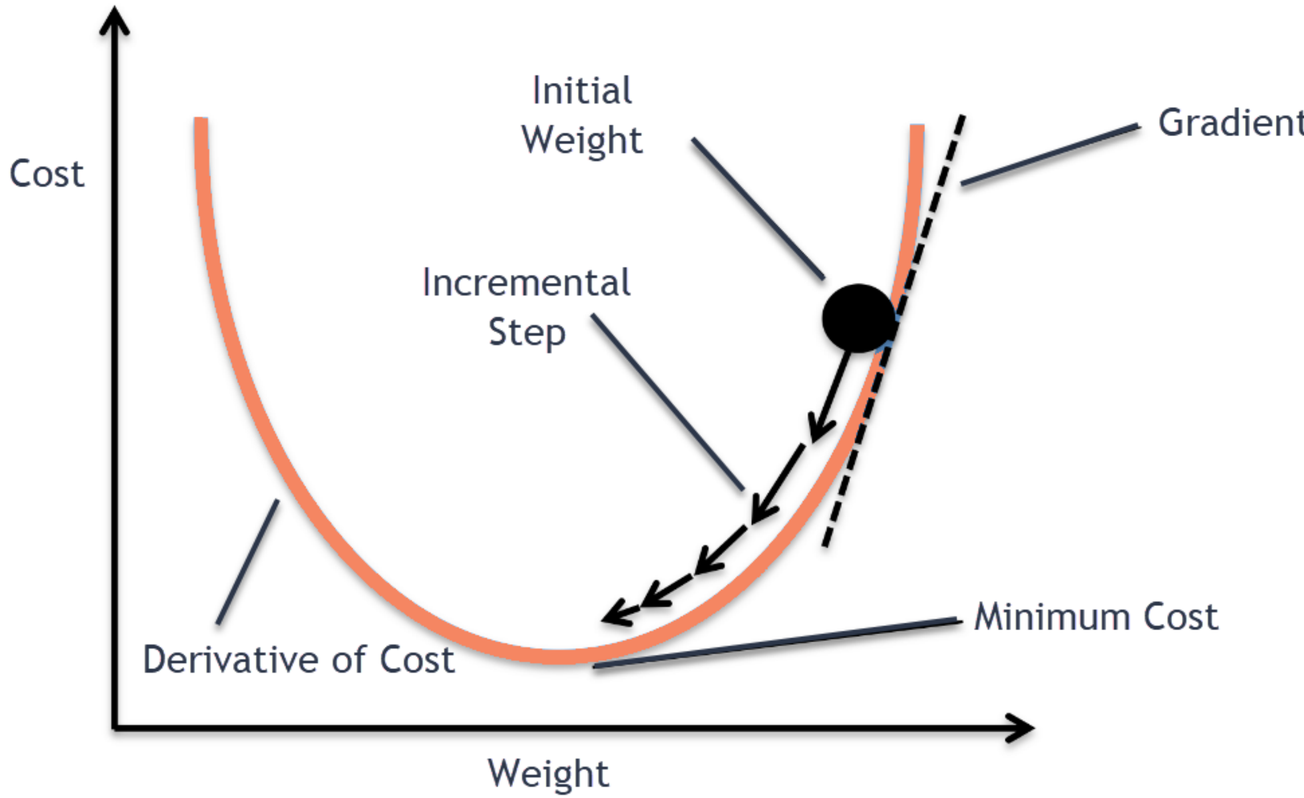


Understanding Gradient Descent

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

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

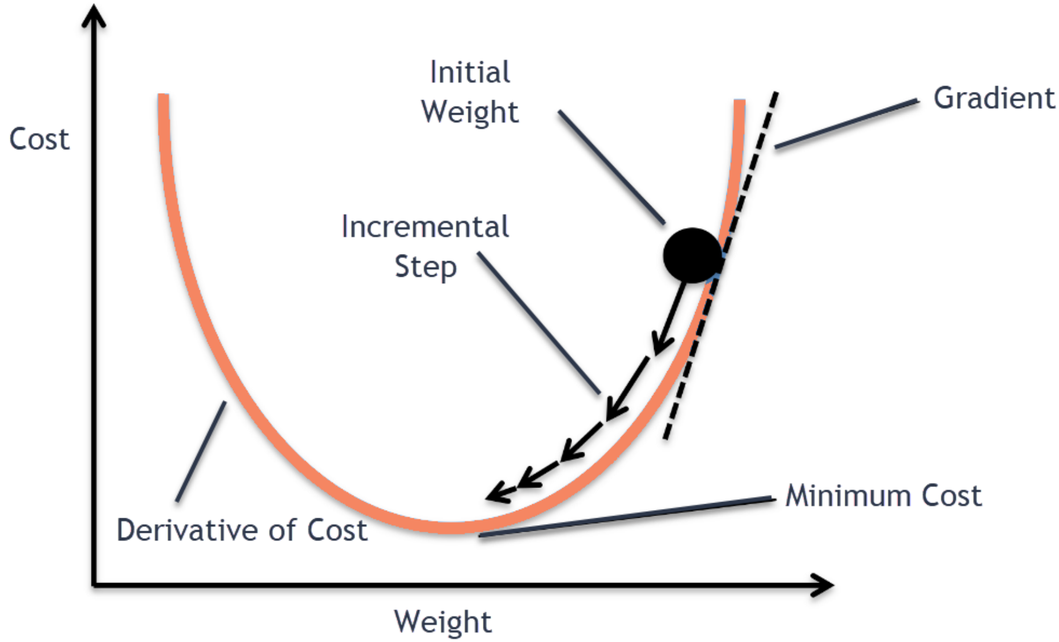
- Move along “steepest” descent direction.

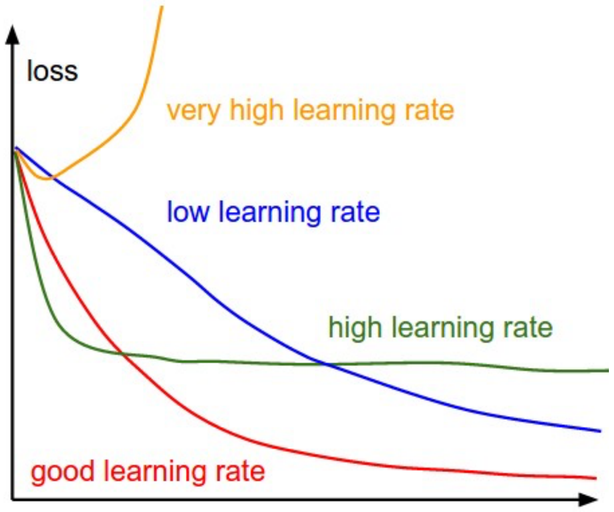


Understanding Gradient Descent

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?

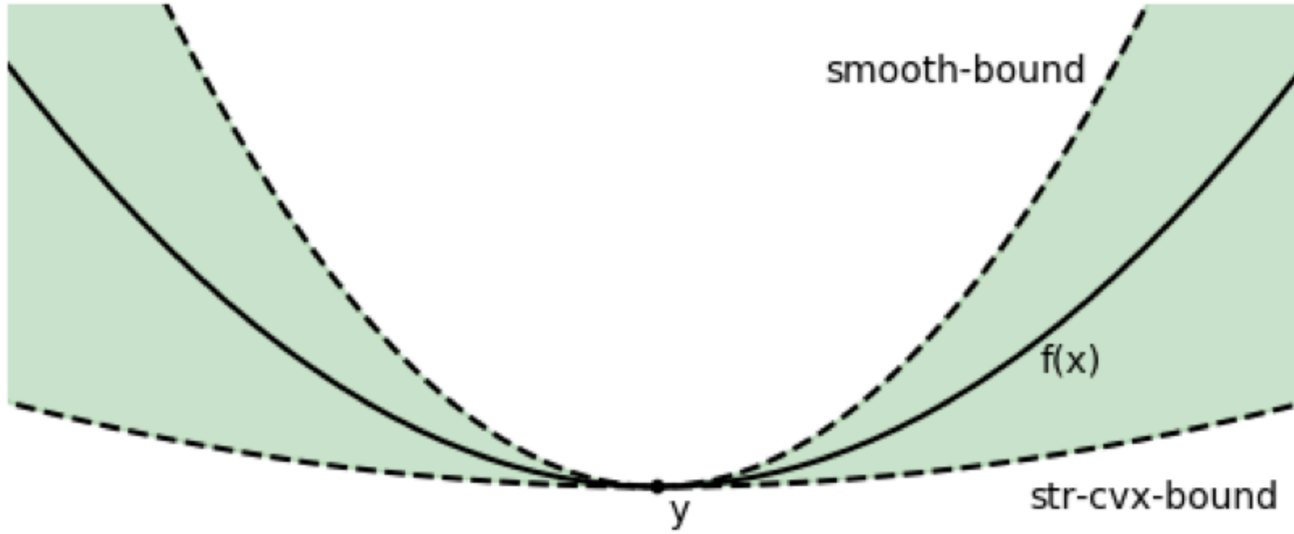




Understanding Gradient Descent

Analysis

- $\theta_t = \theta_{t-1} - \gamma_t \nabla L(\theta_{t-1})$
- How good is our approximation?
  $L(\theta_t + \Delta\theta) \approx L(\theta_t) + \nabla L(\theta_t)^\top \Delta\theta$



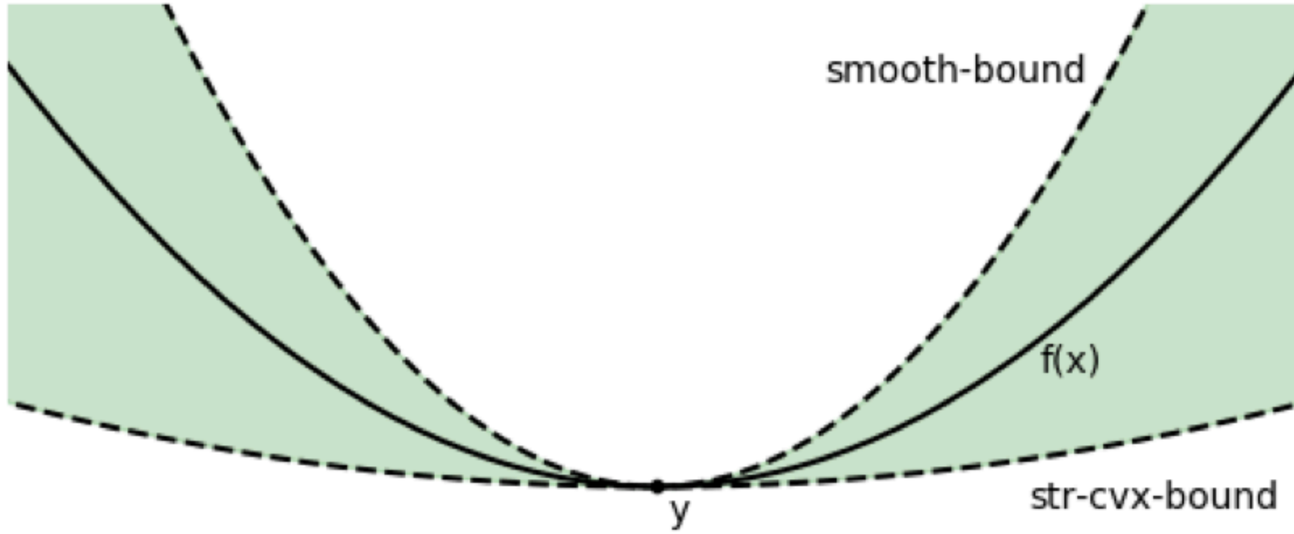
- $\frac{\mu}{2}\|\Delta\theta\|_2^2 \geq L(\theta_t + \Delta\theta) - \left(L(\theta_t) + \nabla L(\theta_t)^\top \Delta\theta\right) \leq \frac{\beta}{2}\|\Delta\theta\|_2^2$

β-Smoothness

Understanding Gradient Descent

Analysis

- $\theta_t = \theta_{t-1} - \gamma_t \nabla L(\theta_{t-1})$
- How good is our approximation?
  $L(\theta_t + \Delta\theta) \approx L(\theta_t) + \nabla L(\theta_t)^\top \Delta\theta$



- $\frac{\mu}{2}\|\Delta\theta\|_2^2 \geq L(\theta_t + \Delta\theta) - \left(L(\theta_t) + \nabla L(\theta_t)^\top \Delta\theta\right) \leq \frac{\beta}{2}\|\Delta\theta\|_2^2$

μ-Strong convex

Understanding Gradient Descent

Convergence analysis

Theorem

If L is β-smooth and μ-strongly convex, gradient descent with $\gamma_t = 1/\beta$ converges as

$$L(\theta_t) - \min_{\theta} L(\theta) \leq \left(1 - \frac{\mu}{\beta}\right)^t \|\theta_0 - \theta^\star\|_2^2$$

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 $L(\theta) = E_{(x,y)}[\ell(f(x;\theta), y)]$

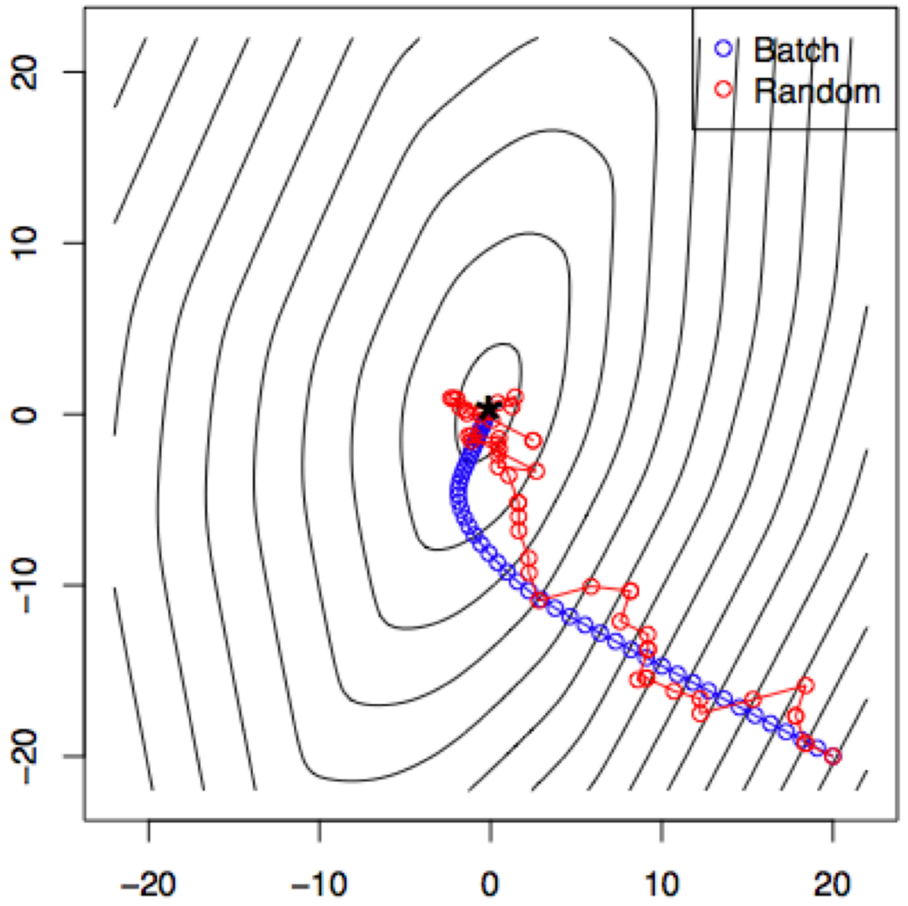
Stochastic Gradient Descent

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)
    - $\theta_t = \theta_{t-1} - \gamma_t \nabla_\theta \ell(f(x_t;\theta_{t-1}), y_t) = \theta_{t-1} - \gamma_t \nabla \ell_t(\theta_{t-1})$

Understanding Gradient Descent

Convergence analysis



Understanding Gradient Descent

Convergence analysis

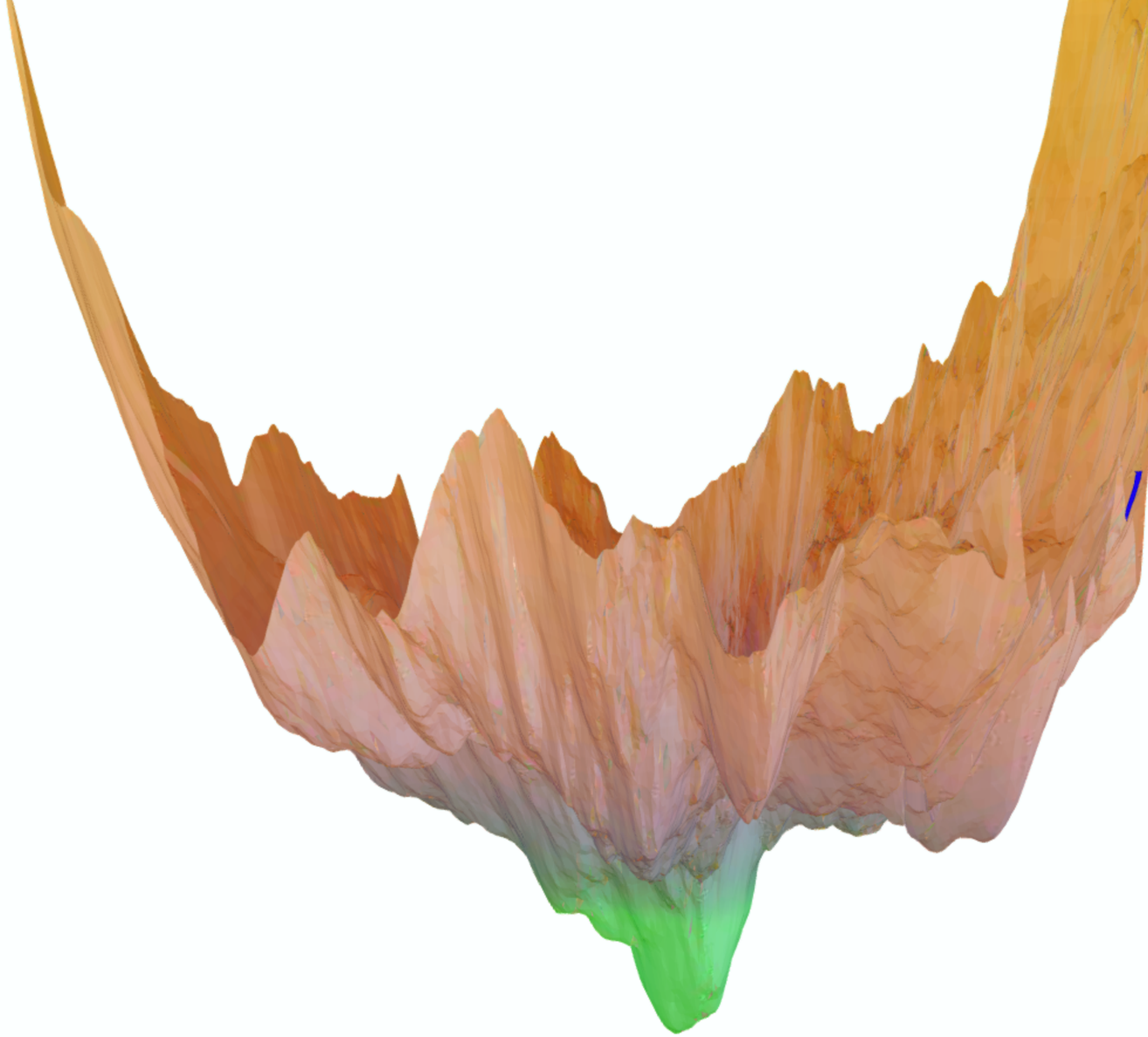
- One final assumption: how bad is this approximation?
- $$\max_{\theta} E\|\nabla \ell_t(\theta) - \nabla L(\theta)\|_2^2 \leq \sigma^2$$
- Proofs cheat sheet: https://gowerrobert.github.io/pdf/M2_statistique_optimisation/grad_conv.pdf

Theorem

If L is β-smooth and μ-strongly convex, SGD with step-size γ converges as

$$E\|\theta^t - \theta^\star\|_2^2 \leq (1 - \gamma\mu)E\|\theta^{t-1} - \theta^\star\|_2^2 + \gamma^2\sigma^2$$

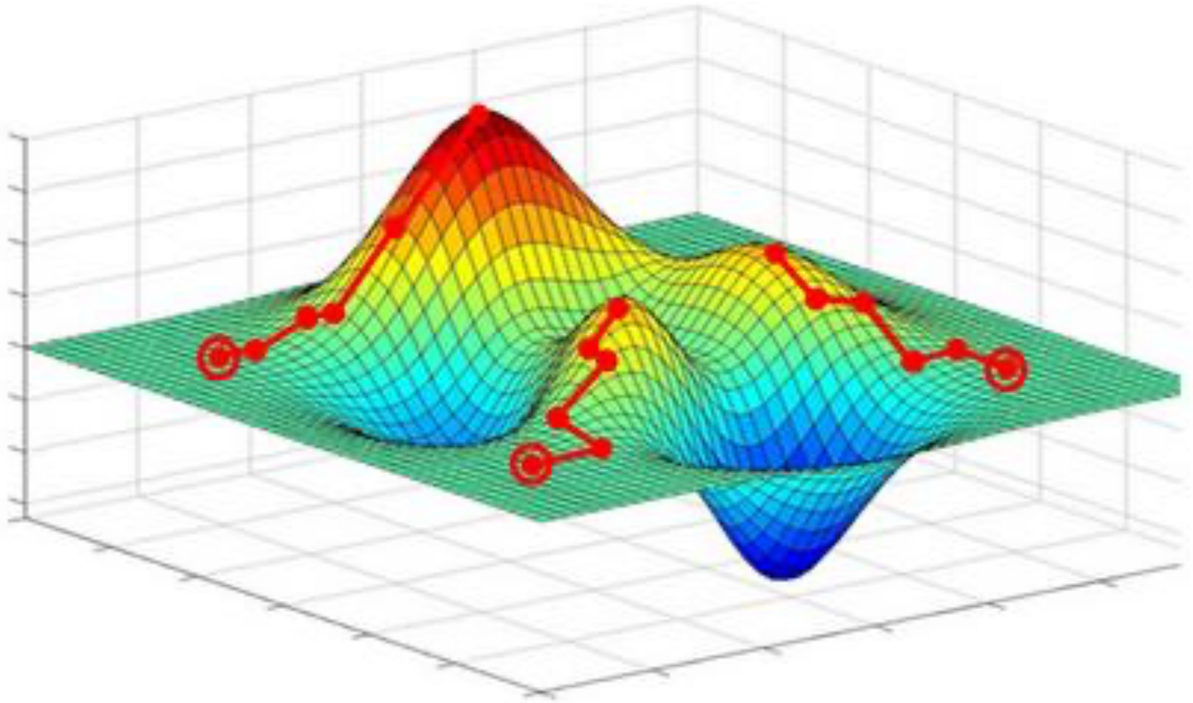
Optimization for Deep Learning



Optimization in Deep Learning

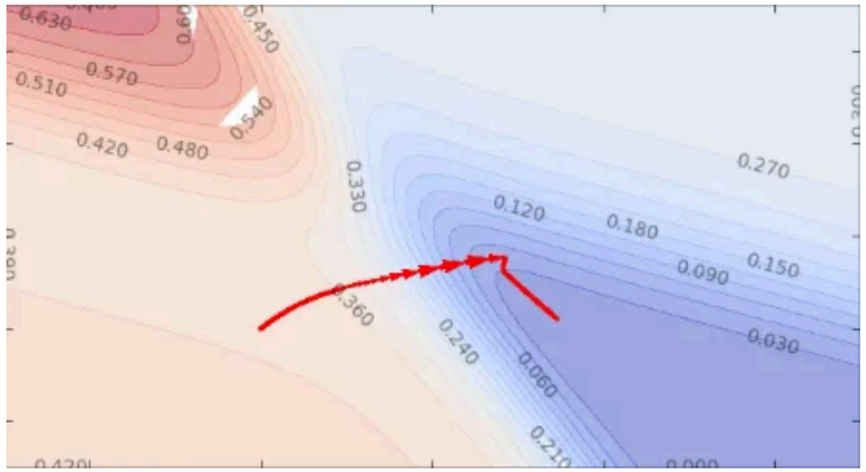
Initialization

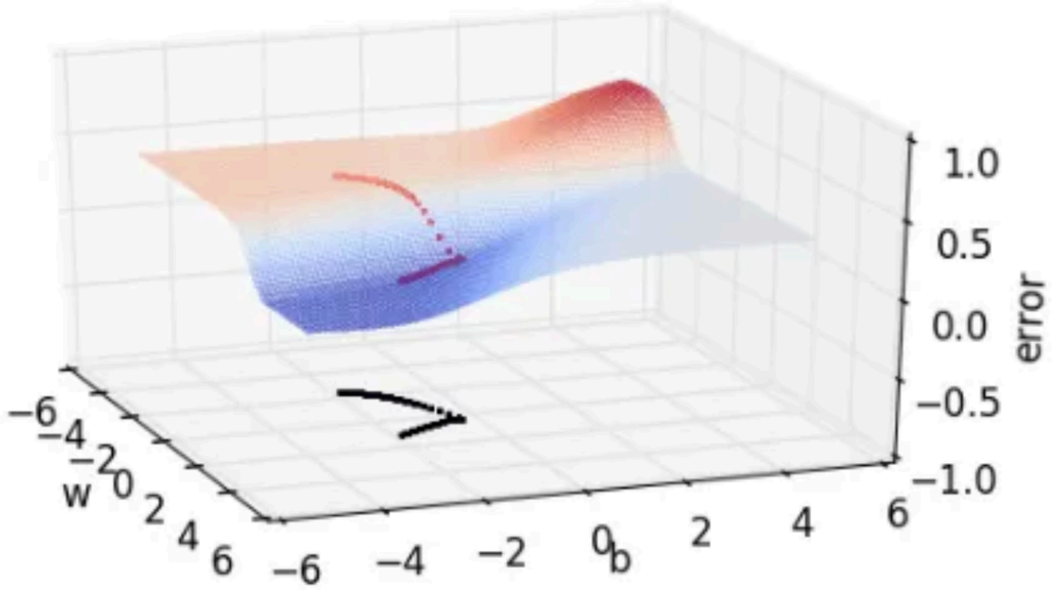
- Initialization matters!
- Always start with a pretrained model if you can.



Optimization in Deep Learning

Momentum





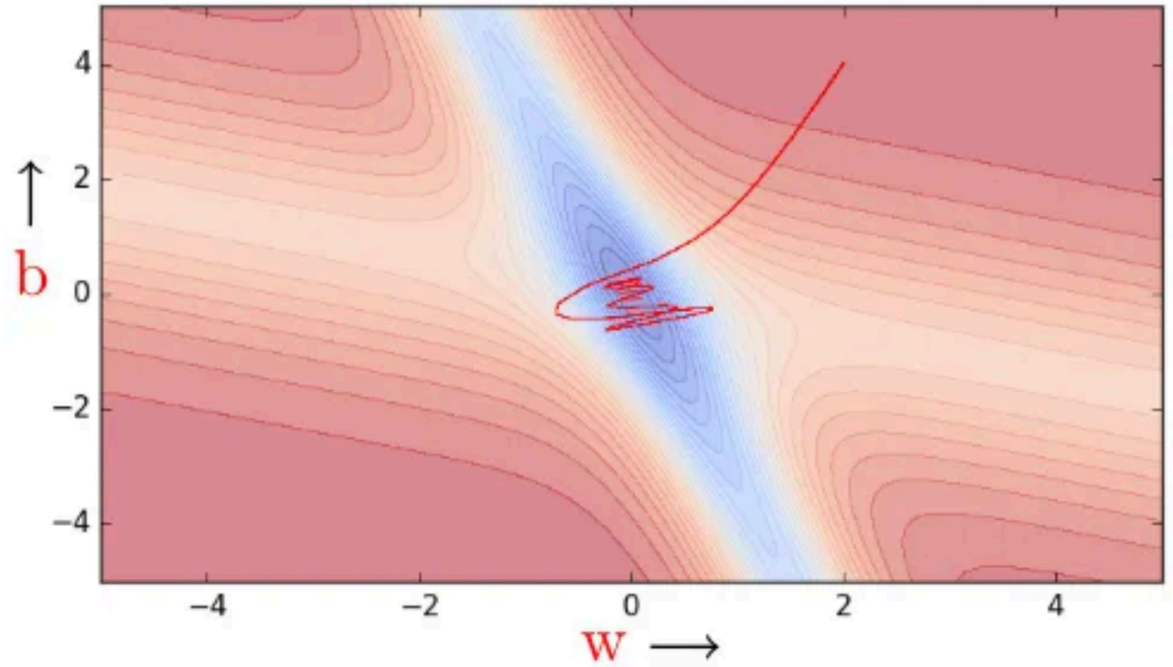
Contour Map with GD applied on it. Arrows represent faster fall in error value. Flat surfaces represented by constant color regions.

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

Optimization in Deep Learning

Momentum

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



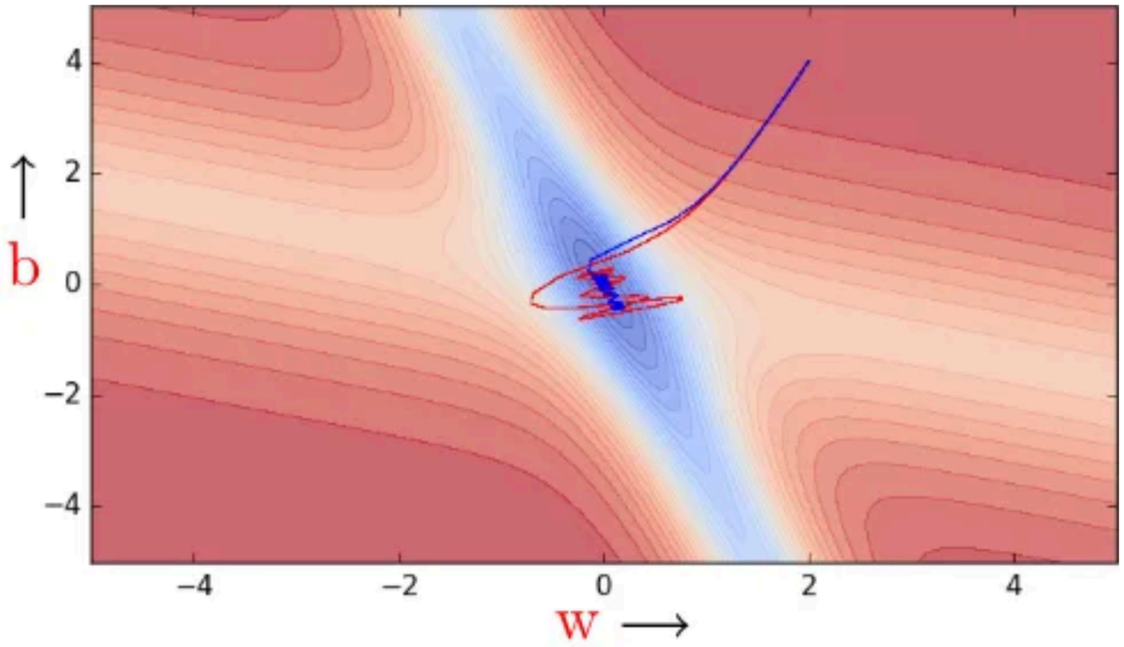
It oscillates in the valley of minima. Take lot U-turns, still converges faster than Vanilla GD.

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

Optimization in Deep Learning

Nesterov Momentum

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



Comparison b/w NAG(blue) v/s Momentum Based Method(Red)

Optimization in Deep Learning

Smoothness

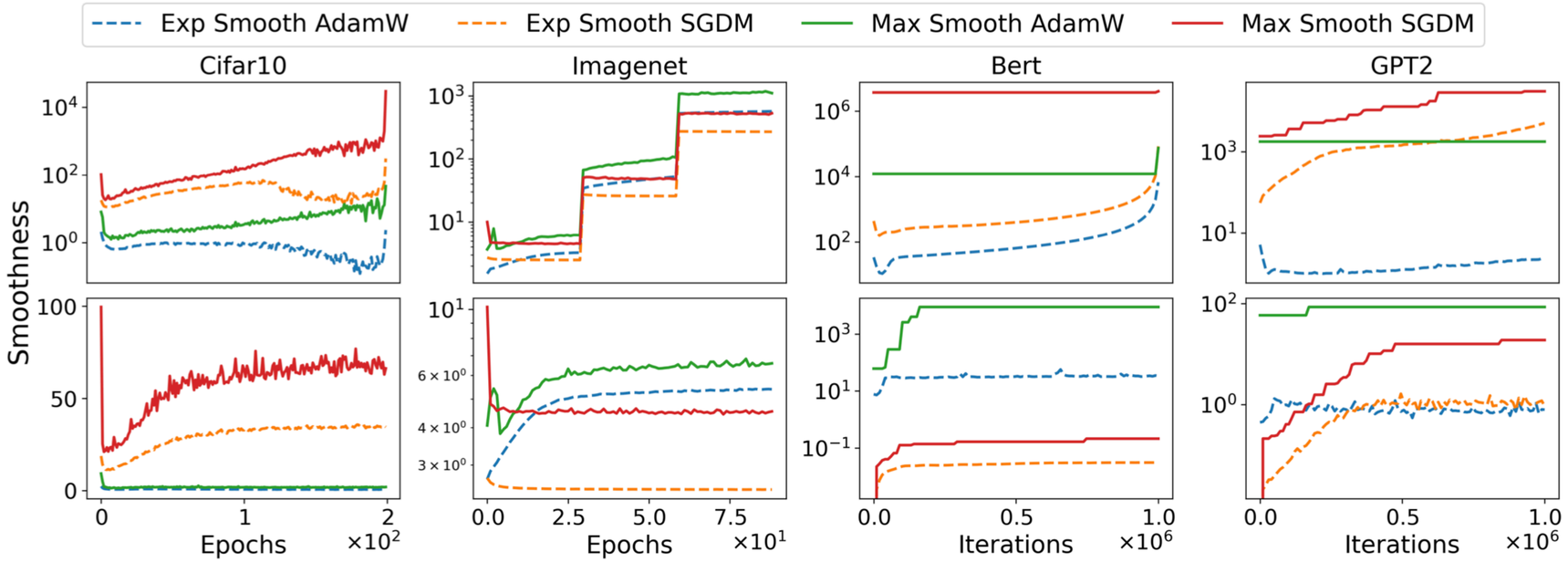


Figure 4: Smoothness measures w.r.t. $\mathbf{y}_t = \mathbf{x}_{t-1}$ of deep learning benchmarks using the optimal configurations. (Top) are the experiments with optimal learning rate scheduler, and (bottom) are the experiments with constant learning rate. Details of experiment setup can be found in Appendix B.

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

Optimization in Deep Learning

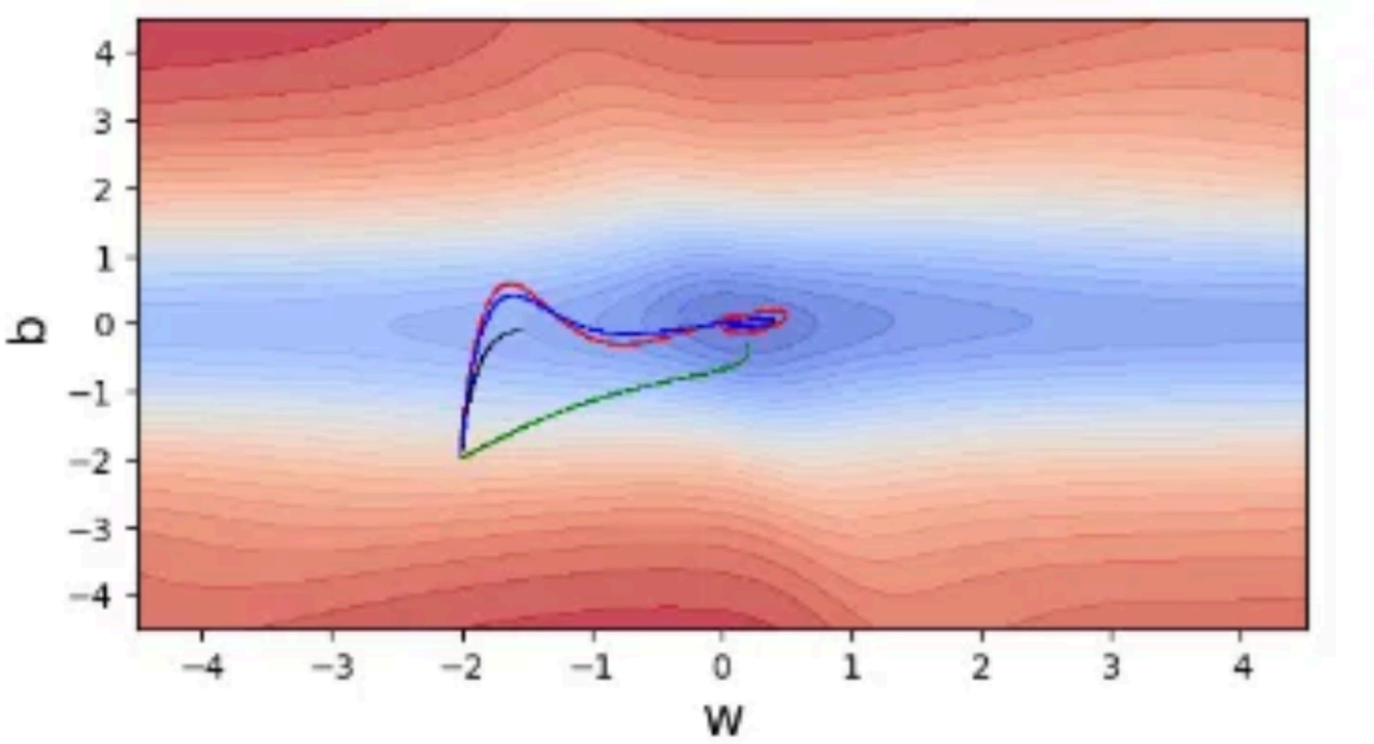
Adaptivity

- 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

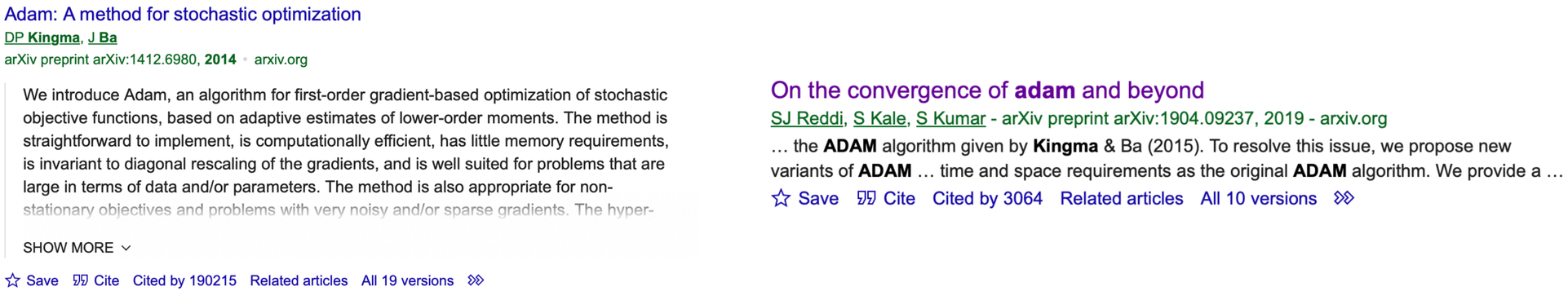


- Progress of AdaGrad in green is much more consistent across steps.
- But it is slower than momentum methods (blue / red)

Optimization in Deep Learning

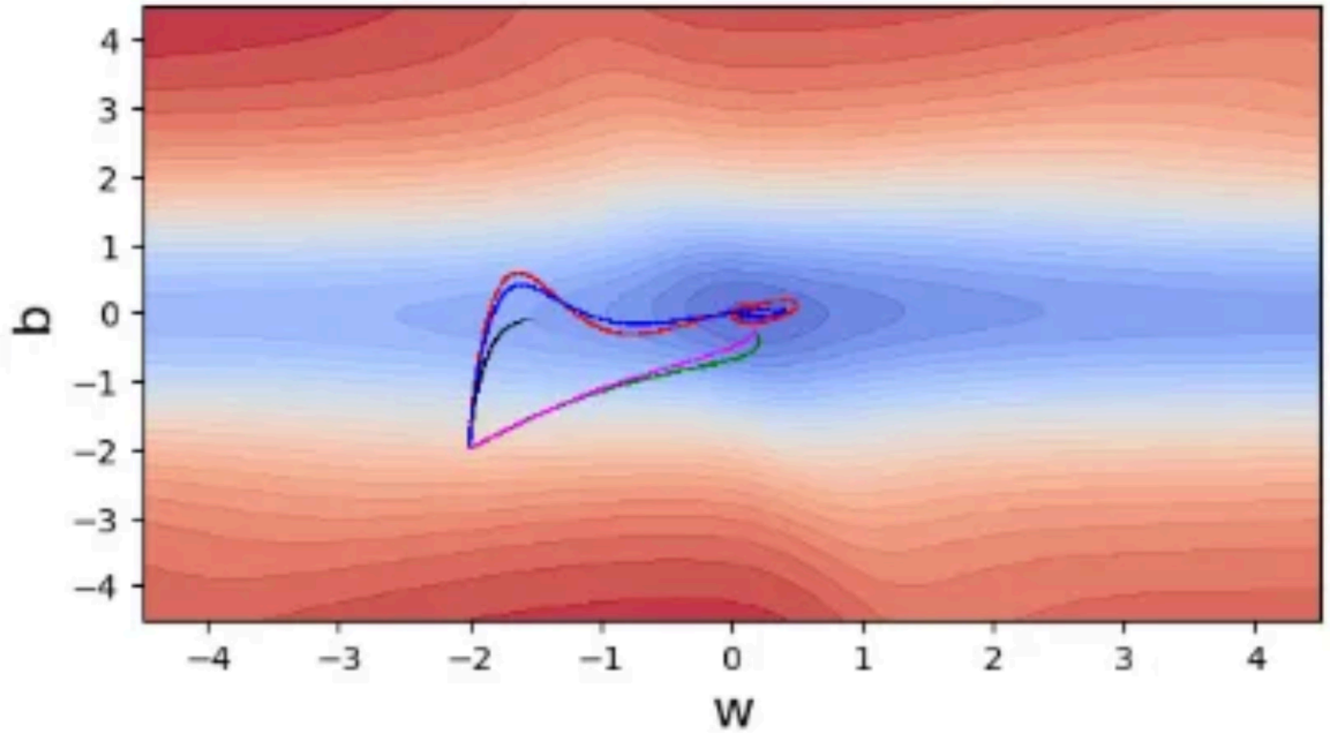
Adam

- 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

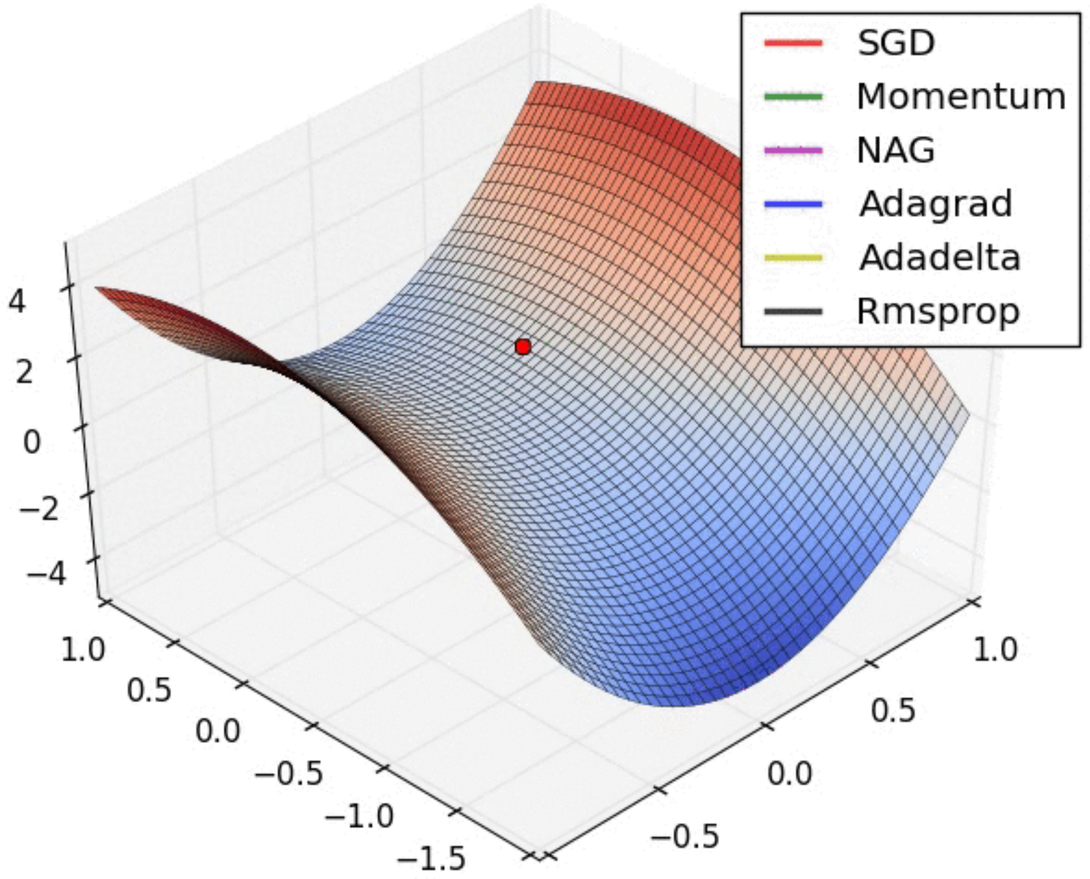


- AdamW (a minor variant) is likely the best default optimizer
- In LLMs, additionally learning rate warm up is used for 5 epochs.

Understanding Deep Learning

Escaping saddle points

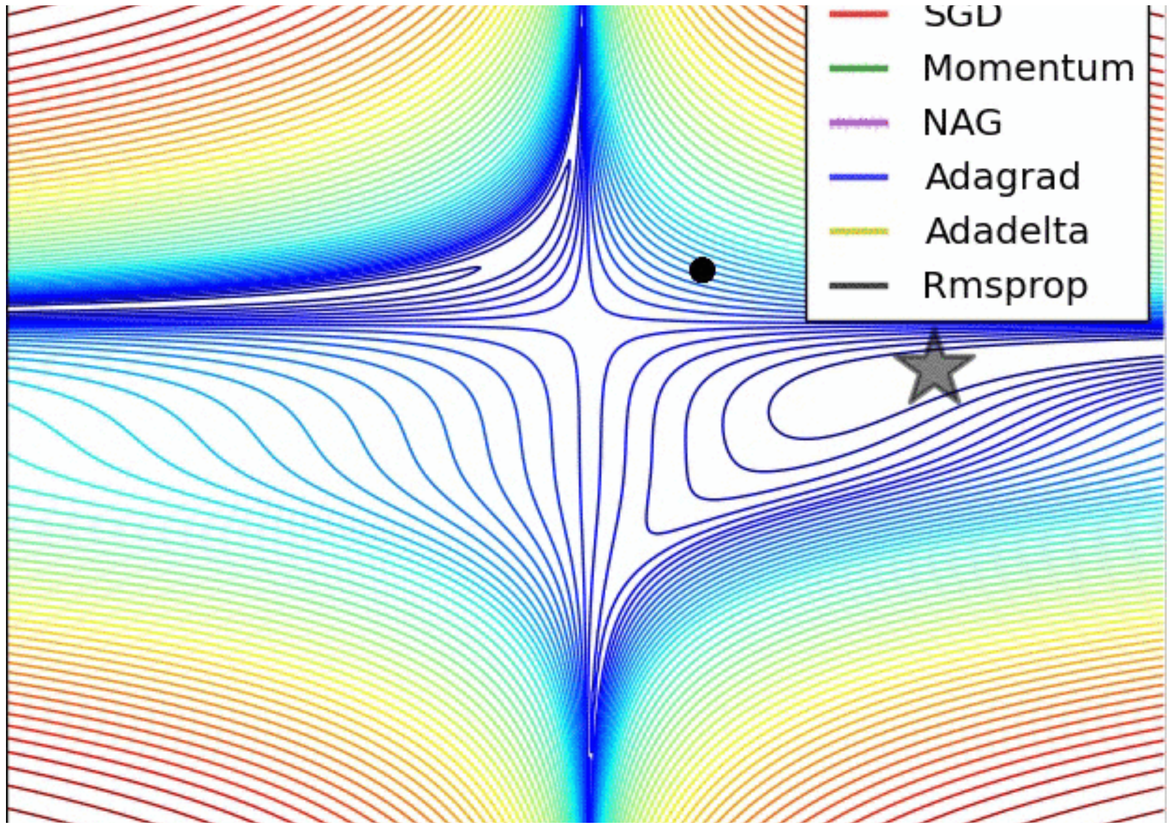
- Non convexity means there are a lot of saddle points
- Adaptivity helps



Understanding Deep Learning

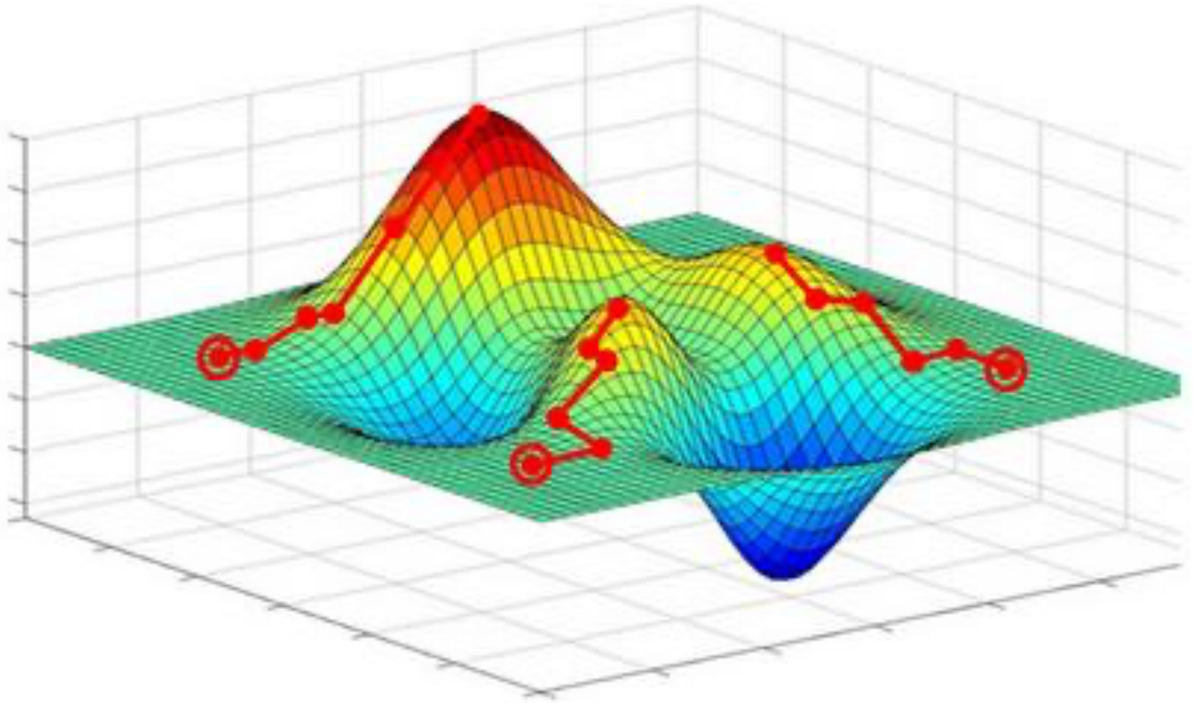
Escaping saddle points

- Non convexity means there are a lot of saddle points
- Adaptivity helps
- Momentum also helps



More tips and tricks

- Normalization: Batchnorm, layer norms etc. improves smoothness
- Skips connections improves smoothness
- Wider networks improve smoothness
- Most important: starting from pertained model $\approx$ convex landscape



Other tips and tricks

- See https://cs231n.github.io/neural-networks-3/
- Check your implementation with backprop and pen and paper
- Initially turn off all bells - no regularization, augmentation, etc.
- Train on 1-2 data points and overfit to convergence

What about privacy?

Can we make SGD private?

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