Grab bag:
Failures of uniform convergence
PAC-Bayes
Online learning

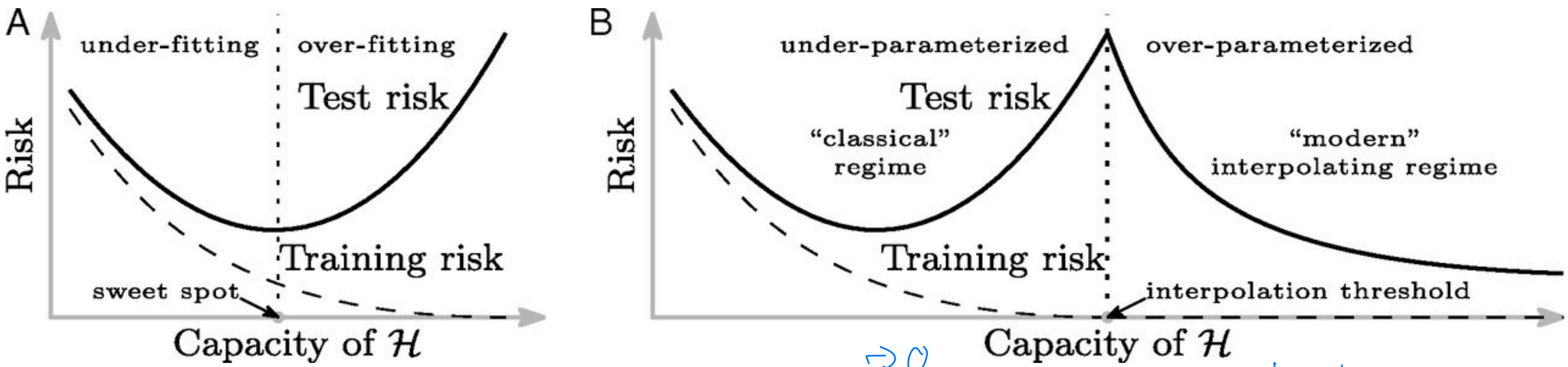
CPSC 532D: Modern Statistical Learning Theory

7 Dec 2022

cs.ubc.ca/~dsuth/532D/22w1/

Admin

- Topics that *won't* be on the final:
  - "Kernels IV", the stuff about operators / etc
  - The last couple lectures:
    - Implicit regularization
    - Neural tangent kernels
    - Universality
    - Rademacher complexity of deep nets
  - Details of any proof
- Stuff that could:
  - Working with basic definitions, etc
  - The homework question about monotonicity of VC/Rademacher is a decent example



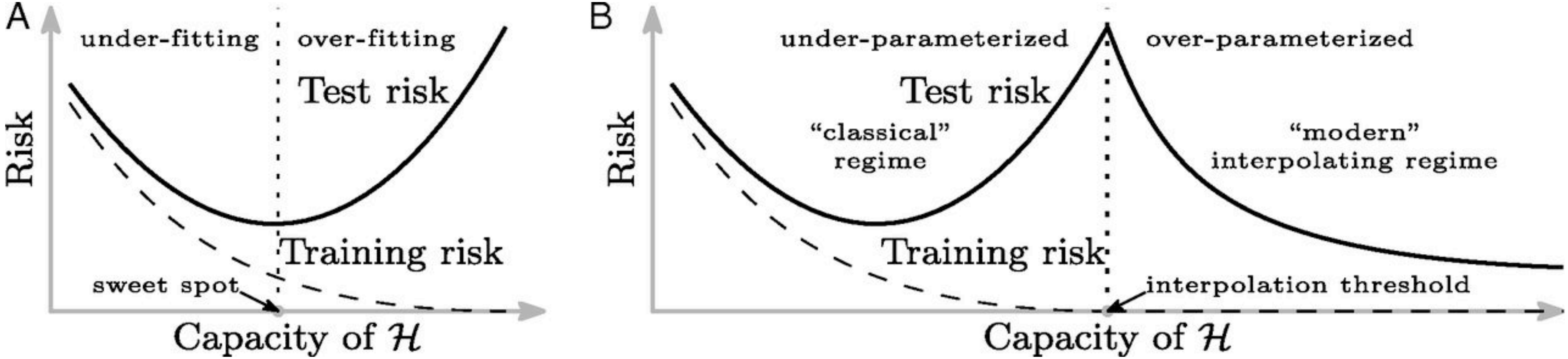
non-realizable classically:

$$\overbrace{\underbrace{L_D(h)}_{\ge \text{ Bayes error}}}^{0} \le \overbrace{\underbrace{L_S(h)}_{\to \approx \text{ Bayes error},\ \to \inf_{h\in\mathcal{H}} L_D(h)}}^{0} + \overbrace{\underbrace{\underbrace{\sup_{h\in\mathcal{H}} L_D(h)-L_S(h)}_{\text{bound}}}_{\to 0}}^{\to 0 \text{ if realizable}}$$

$$\sup_{\substack{h\in\mathcal{H}\\ \text{s.t. } L_S(h)=0}} L_D(h) - \cancel{L_S(h)}$$

interpolating non-realizable:

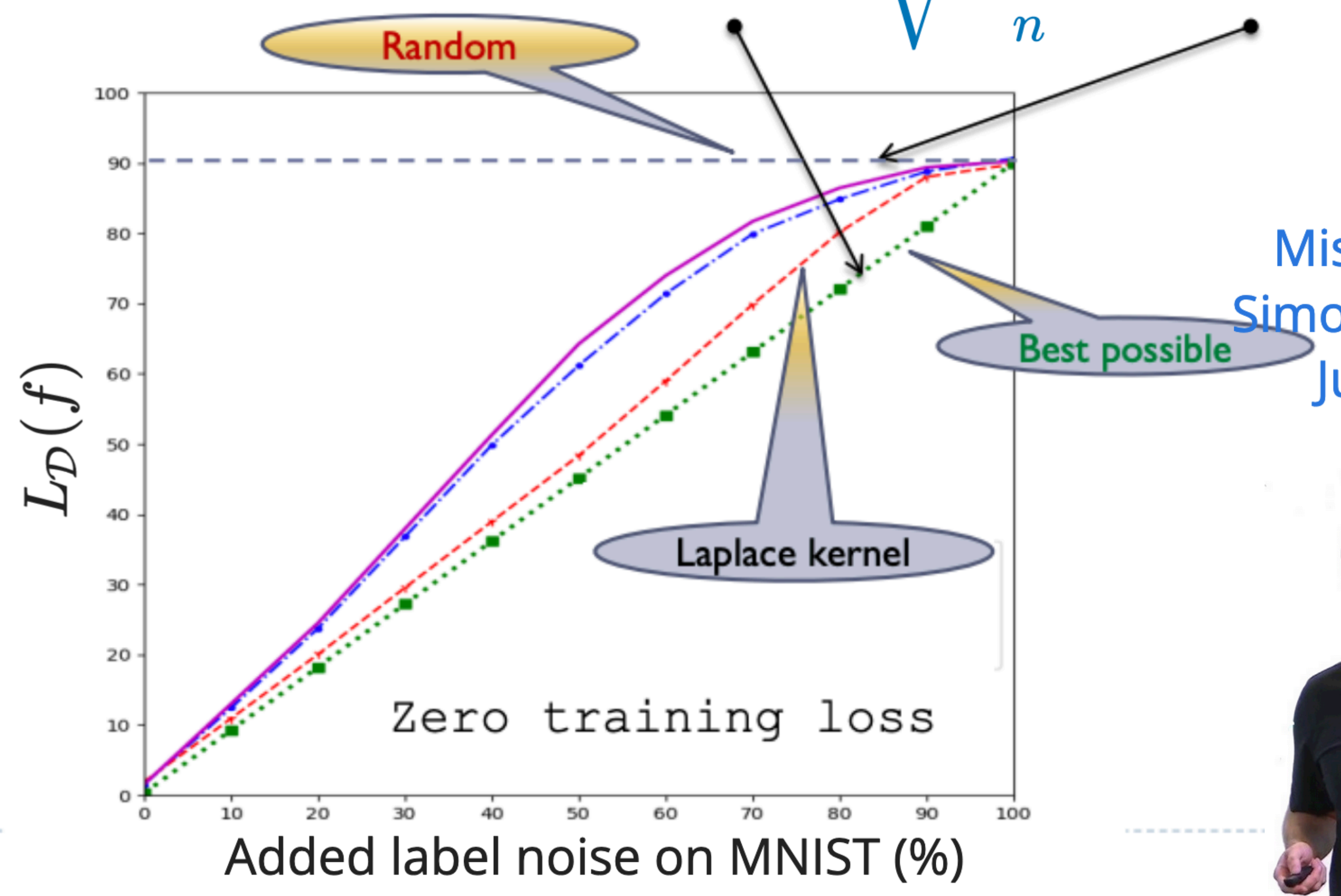
$$\underbrace{L_D(h)}_{\ge \text{ Bayes error}} \le \underbrace{L_S(h)}_{0} + \underbrace{\sup_{h\in\mathcal{H}} L_D(h)-L_S(h)}_{\text{bound} \xrightarrow{\text{want}} \text{Bayes error}}$$



Bounds? $L_{\mathcal{D}}(\hat{f}) \leq \underbrace{L_{\mathbf{S}}(\hat{f})}_{0} + \sqrt{\frac{C_{\mathcal{F},\delta}}{n}}$

What kind of generalization bound could work here?

$0.7 < \sqrt{\frac{C_{\mathcal{F},\delta}}{n}} < 0.9$



Misha Belkin
Simons Institute
July 2019

Belkin/Ma/Mandal, ICML 2018

Uniform convergence may be unable to explain generalization in deep learning

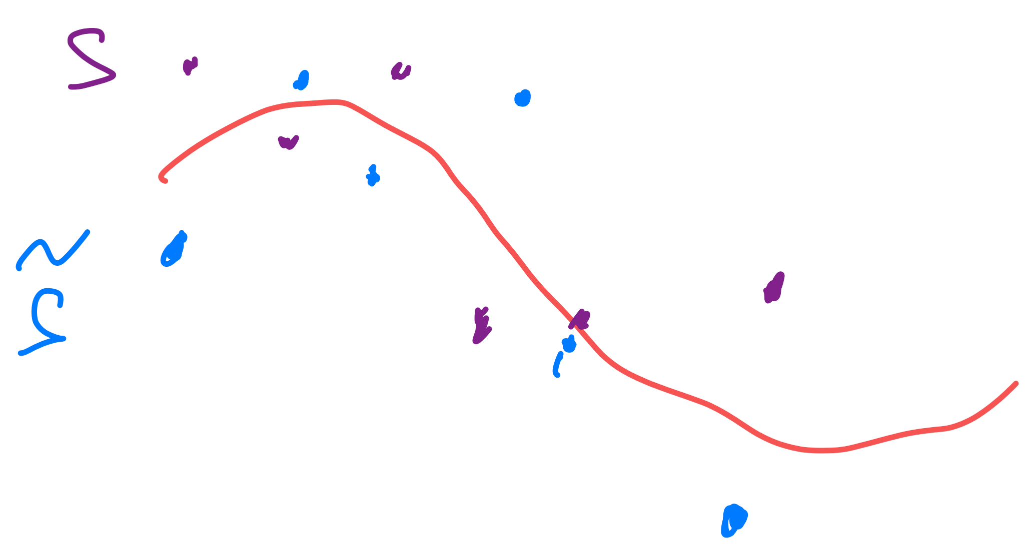
Vaishnavh Nagarajan
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA
vaishnavh@cs.cmu.edu

J. Zico Kolter
Department of Computer Science
Carnegie Mellon University &
Bosch Center for Artificial Intelligence
Pittsburgh, PA
zkolter@cs.cmu.edu

$$\lim_{n\to\infty} \mathbb{E}_{S\sim D^n} \sup_{h\in\mathcal{H}} \Big| \overset{\sigma^2}{L_D(h)} - \overset{4\sigma^2}{L_S(h)} \Big| \ge 3\sigma^2$$

[and $L_S(h)=0$]

$L_D(A(S)) \xrightarrow{n\to\infty} \sigma^2 \leftarrow$ Bayes error



$X \sim \mathcal{N}(0,\Sigma) \in \mathbb{R}^d$ $\quad$ $d = \omega(n)$

$y = Xw^* + \varepsilon$

$\varepsilon \sim \mathcal{N}(0,\sigma^2)$

$A(S) = X^\dagger y$

$L_S(A(S)) = 0$

$L_S(A(\tilde{S}))$

$= \frac{1}{n}\sum_{i=1}^{n} \left(w^*\cdot x_i - \varepsilon_i - (w^*\cdot x_i + \varepsilon_i) \right)^2$

$= \frac{1}{n}\sum_i (2\varepsilon_i)^2$

$= 4 \cdot \underbrace{\frac{1}{n}\sum_i \varepsilon_i^2} \to 4\sigma^2$

If Σ satisfies some conditions,

$L_D(A(S)) \to \sigma^2$

Bartlett, Long, Lugosi, Tsigler (2020) "Benign overfitting in ..."

$L_D(A(\tilde{S})) \to \sigma^2$

choose e.g. $\mathcal{H} = \{x \mapsto w^T x : \|w\| \le B_n\}$:(

smallest possible $\mathcal{H}$: $\mathcal{H}_n = \{A(S) : S \in \mathcal{S}_n\}$, $\Pr_{S\sim D^n}(S \in \mathcal{S}_n) \ge 1-\delta$

must be at least one pair with $S \in \mathcal{S}_n$, $\tilde{S} \in \mathcal{S}_n$

S has $\underset{n\times d}{X}, \underset{n\times 1}{y}$

$\tilde{S}$ with X, $2Xw^* - y = 2Xw^* - (Xw^* + \varepsilon) = Xw^* - \varepsilon$

(pause)

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis
  - Observe data S with likelihood $\mathscr{L}(S \mid h)$

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis
  - Observe data S with likelihood $\mathscr{L}(S \mid h)$
  - End up with posterior distribution $\rho(h \mid S) \propto \mathscr{L}(S \mid h)\,\pi(h)$

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis
  - Observe data S with likelihood $\mathscr{L}(S \mid h)$
  - End up with posterior distribution $\rho(h \mid S) \propto \mathscr{L}(S \mid h)\, \pi(h)$
  - Make predictions/decision based on posterior mean/median, MAP, single draw, …

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis
  - Observe data S with likelihood $\mathscr{L}(S \mid h)$
  - End up with posterior distribution $\rho(h \mid S) \propto \mathscr{L}(S \mid h)\, \pi(h)$
  - Make predictions/decision based on posterior mean/median, MAP, single draw, …
- This is optimal if you believe in your prior + likelihood! 😊

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis
  - Observe data S with likelihood $\mathscr{L}(S \mid h)$
  - End up with posterior distribution $\rho(h \mid S) \propto \mathscr{L}(S \mid h)\,\pi(h)$
  - Make predictions/decision based on posterior mean/median, MAP, single draw, …
- This is optimal if you believe in your prior + likelihood! 😊
  - Frequentists say: “but how good is it actually???”

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis
  - Observe data S with likelihood $\mathscr{L}(S \mid h)$
  - End up with posterior distribution $\rho(h \mid S) \propto \mathscr{L}(S \mid h)\,\pi(h)$
  - Make predictions/decision based on posterior mean/median, MAP, single draw, …
- This is optimal if you believe in your prior + likelihood! 😊
  - Frequentists say: "but how good is it actually???"
  - What if your model class / prior / … are wrong?

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis
  - Observe data S with likelihood $\mathscr{L}(S \mid h)$
  - End up with posterior distribution $\rho(h \mid S) \propto \mathscr{L}(S \mid h)\, \pi(h)$
  - Make predictions/decision based on posterior mean/median, MAP, single draw, …
- This is optimal if you believe in your prior + likelihood! 😊
  - Frequentists say: “but how good is it actually???”
  - What if your model class / prior / … are wrong?
- Tempered likelihood (Zhang 06) / SafeBayes (Grünwald 12):

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis
  - Observe data S with likelihood $\mathscr{L}(S \mid h)$
  - End up with posterior distribution $\rho(h \mid S) \propto \mathscr{L}(S \mid h)\, \pi(h)$
  - Make predictions/decision based on posterior mean/median, MAP, single draw, …
- This is optimal if you believe in your prior + likelihood! 😊
  - Frequentists say: "but how good is it actually???"
  - What if your model class / prior / … are wrong?
- Tempered likelihood (Zhang 06) / SafeBayes (Grünwald 12):
  - If your model is misspecified, can be provably better to use $\mathscr{L}^\lambda$ for $\lambda < 1$

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis
  - Observe data S with likelihood $\mathscr{L}(S \mid h)$
  - End up with posterior distribution $\rho(h \mid S) \propto \mathscr{L}(S \mid h)\, \pi(h)$
  - Make predictions/decision based on posterior mean/median, MAP, single draw, …
- This is optimal if you believe in your prior + likelihood! 😊
  - Frequentists say: "but how good is it actually???"
  - What if your model class / prior / … are wrong?
- Tempered likelihood (Zhang 06) / SafeBayes (Grünwald 12):
  - If your model is misspecified, can be provably better to use $\mathscr{L}^\lambda$ for $\lambda < 1$
  - No longer quite Bayesian inference, but turns a prior into a posterior

A road to PAC-Bayes

- Bayesians say:
  - Start with a prior distribution $\pi(h)$ on choice of hypothesis
  - Observe data S with likelihood $\mathscr{L}(S \mid h)$
  - End up with posterior distribution $\rho(h \mid S) \propto \mathscr{L}(S \mid h)\, \pi(h)$
  - Make predictions/decision based on posterior mean/median, MAP, single draw, …
- This is optimal if you believe in your prior + likelihood! 😊
  - Frequentists say: “but how good is it actually???”
  - What if your model class / prior / … are wrong?
- Tempered likelihood (Zhang 06) / SafeBayes (Grünwald 12):
  - If your model is misspecified, can be provably better to use $\mathscr{L}^\lambda$ for $\lambda < 1$
  - No longer quite Bayesian inference, but turns a prior into a posterior
- PAC-Bayes: analyzes *any* prior-posterior pair (potentially even totally unrelated)

PAC-Bayes: McAllester bound

- We start with some prior π (independent of the data S) on hypotheses

PAC-Bayes: McAllester bound

- We start with some prior π (independent of the data S) on hypotheses
- Our learning algorithm sees S and gives us a posterior ρ

PAC-Bayes: McAllester bound

- We start with some prior π (independent of the data S) on hypotheses
- Our learning algorithm sees S and gives us a posterior ρ
- We'll analyze $L_{\mathscr{D}}(\rho) = \mathbb{E}_{h \sim \rho}[L_{\mathscr{D}}(h)]$ based on $L_S(\rho) = \mathbb{E}_{h \sim \rho}[L_S(h)]$

PAC-Bayes: McAllester bound

- We start with some prior π (independent of the data S) on hypotheses
- Our learning algorithm sees S and gives us a posterior ρ
- We'll analyze $L_{\mathscr{D}}(\rho) = \mathbb{E}_{h\sim\rho}[L_{\mathscr{D}}(h)]$ based on $L_S(\rho) = \mathbb{E}_{h\sim\rho}[L_S(h)]$
- McAllester-style bound (SSBD theorem 31.1):

PAC-Bayes: McAllester bound

- We start with some prior π (independent of the data S) on hypotheses
- Our learning algorithm sees S and gives us a posterior ρ
- We'll analyze $L_{\mathcal{D}}(\rho) = \mathbb{E}_{h\sim\rho}[L_{\mathcal{D}}(h)]$ based on $L_S(\rho) = \mathbb{E}_{h\sim\rho}[L_S(h)]$
- McAllester-style bound (SSBD theorem 31.1):
    - If $\ell(h, z) \in [0,1]$, with probability at least $1 - \delta$ over $S \sim \mathcal{D}^n$,

$$L_{\mathcal{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

where $\mathrm{KL}(\rho \| \pi) = \mathbb{E}_{h\sim\rho} \log \frac{\rho(h)}{\pi(h)}$ (the usual KL divergence)

PAC-Bayes: McAllester bound

- We start with some prior π (independent of the data S) on hypotheses
- Our learning algorithm sees S and gives us a posterior ρ
- We'll analyze $L_{\mathscr{D}}(\rho) = \mathbb{E}_{h\sim\rho}[L_{\mathscr{D}}(h)]$ based on $L_S(\rho) = \mathbb{E}_{h\sim\rho}[L_S(h)]$
- McAllester-style bound (SSBD theorem 31.1):
  - If $\ell(h, z) \in [0,1]$, with probability at least $1 - \delta$ over $S \sim \mathscr{D}^n$,

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \le \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

    where $\mathrm{KL}(\rho \| \pi) = \mathbb{E}_{h\sim\rho} \log \dfrac{\rho(h)}{\pi(h)}$ (the usual KL divergence)

  - Proved in SSBD chapter 31 (not bad at all)

What learning algorithm?

$$L_{\mathcal{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best learning algorithm, according to this bound?

What learning algorithm?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \le \sqrt{\frac{\mathrm{KL}(\rho\|\pi) + \log\frac{n}{\delta}}{2(n-1)}}$$

- What's the best learning algorithm, according to this bound?
  - Turns out to be the **Gibbs posterior**: $\rho(h) \propto \exp(-\lambda\, L_S(h))\ \pi(h)$

What learning algorithm?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best learning algorithm, according to this bound?
  - Turns out to be the **Gibbs posterior**: $\rho(h) \propto \exp(-\lambda\, L_S(h))\ \pi(h)$
  - Same as tempered likelihood / SafeBayes if $\mathscr{L}(S \mid h) = -\log L_S(h) + \text{const}$

What learning algorithm?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best learning algorithm, according to this bound?
  - Turns out to be the **Gibbs posterior**: $\rho(h) \propto \exp(-\lambda\, L_S(h))\ \pi(h)$
  - Same as tempered likelihood / SafeBayes if $\mathscr{L}(S \mid h) = -\log L_S(h) + \mathrm{const}$
  - Typical choice (see 340): e.g. squared loss ↔ Gaussian likelihood

What learning algorithm?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best learning algorithm, according to this bound?
  - Turns out to be the **Gibbs posterior**: $\rho(h) \propto \exp(-\lambda\, L_S(h))\, \pi(h)$
  - Same as tempered likelihood / SafeBayes if $\mathscr{L}(S \mid h) = -\log L_S(h) + \text{const}$
  - Typical choice (see 340): e.g. squared loss ↔ Gaussian likelihood

- But the bound applies to **any** prior-posterior pair (with π independent of S)

What learning algorithm?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best learning algorithm, according to this bound?
  - Turns out to be the **Gibbs posterior**: $\rho(h) \propto \exp(-\lambda\, L_S(h))\ \pi(h)$
  - Same as tempered likelihood / SafeBayes if $\mathscr{L}(S \mid h) = -\log L_S(h) + \mathrm{const}$
  - Typical choice (see 340): e.g. squared loss ↔ Gaussian likelihood

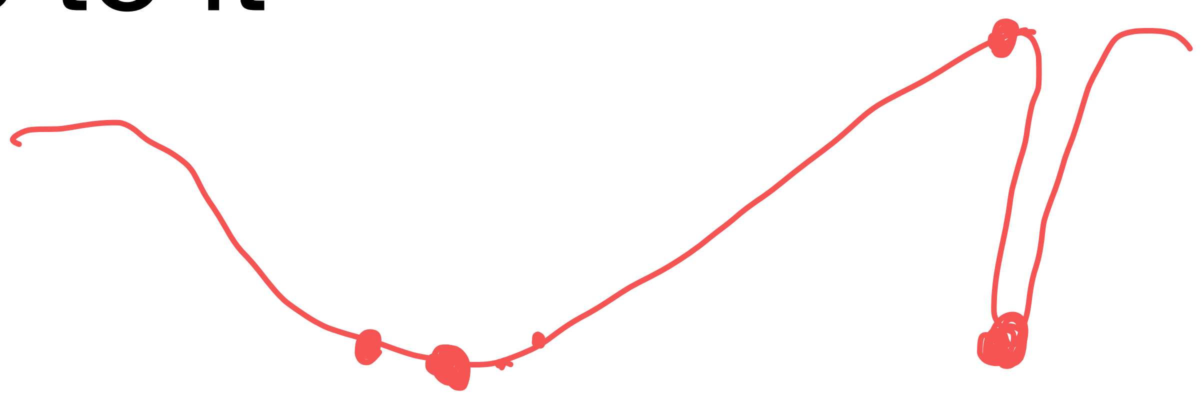
- But the bound applies to **any** prior-posterior pair (with π independent of S)
  - For instance: could learn a $\hat{h}$ with (S)GD and then add noise to it

What learning algorithm?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best learning algorithm, according to this bound?
  - Turns out to be the **Gibbs posterior**: $\rho(h) \propto \exp(-\lambda\, L_S(h))\ \pi(h)$
  - Same as tempered likelihood / SafeBayes if $\mathscr{L}(S \mid h) = -\log L_S(h) + \mathrm{const}$
  - Typical choice (see 340): e.g. squared loss ↔ Gaussian likelihood

- But the bound applies to **any** prior-posterior pair (with π independent of S)
  - For instance: could learn a $\hat{h}$ with (S)GD and then add noise to it
  - If $\hat{h}$ is in a **flat minimum**, then $\hat{h}$ + noise will still be good



What learning algorithm?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best learning algorithm, according to this bound?
  - Turns out to be the **Gibbs posterior**: $\rho(h) \propto \exp(-\lambda\, L_S(h))\ \pi(h)$
  - Same as tempered likelihood / SafeBayes if $\mathscr{L}(S \mid h) = -\log L_S(h) + \mathrm{const}$
  - Typical choice (see 340): e.g. squared loss ↔ Gaussian likelihood

- But the bound applies to **any** prior-posterior pair (with π independent of S)
  - For instance: could learn a $\hat{h}$ with (S)GD and then add noise to it
  - If $\hat{h}$ is in a **flat minimum**, then $\hat{h} +$ noise will still be good
  - But note that if $\rho \to$ point mass and π continuous, $\mathrm{KL}(\rho \| \pi) \to \infty$

What prior?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \le \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best prior?
  - Bound on generalization gap is better if ρ is "closer" to π

What prior?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best prior?
  - Bound on generalization gap is better if ρ is "closer" to π
    - S didn't make us "change our mind" too much – similar to MDL

What prior?

$$L_{\mathcal{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What’s the best prior?
  - Bound on generalization gap is better if ρ is “closer” to π
    - S didn’t make us “change our mind” too much – similar to MDL
  - But we also want a good ρ, i.e. average training loss $L_S(\rho)$ should be small

What prior?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best prior?
  - Bound on generalization gap is better if ρ is "closer" to π
    - S didn't make us "change our mind" too much – similar to MDL
  - But we also want a good ρ, i.e. average training loss $L_S(\rho)$ should be small
- Notice π only shows up in the bound – nothing to do with the learning algorithm

What prior?

$$L_{\mathscr{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best prior?
  - Bound on generalization gap is better if ρ is "closer" to π
    - S didn't make us "change our mind" too much – similar to MDL
  - But we also want a good ρ, i.e. average training loss $L_S(\rho)$ should be small
- Notice π only shows up in the bound – nothing to do with the learning algorithm
  - We could potentially pick a prior that actually **depends on** $\mathscr{D}$

What prior?

$$L_{\mathcal{D}}(\rho) - L_S(\rho) \leq \sqrt{\frac{\mathrm{KL}(\rho \| \pi) + \log \frac{n}{\delta}}{2(n-1)}}$$

- What's the best prior?
  - Bound on generalization gap is better if ρ is "closer" to π
    - S didn't make us "change our mind" too much – similar to MDL
  - But we also want a good ρ, i.e. average training loss $L_S(\rho)$ should be small
- Notice π only shows up in the bound – nothing to do with the learning algorithm
  - We could potentially pick a prior that actually **depends on** $\mathcal{D}$
  - …as long as we can still bound $\mathrm{KL}(\rho \| \pi)$

Other forms of PAC-Bayes bounds

- Linear bound: $L_{\mathscr{D}}(\rho) \leq \frac{1}{\beta} L_S(\rho) + \frac{\mathrm{KL}(\rho \| \pi) + \log \frac{1}{\delta}}{2\beta(1-\beta)n}$ for any $\beta \in (0,1)$

- Catoni bound: for $\alpha > 1$, $\Phi_\gamma^{-1}(x) = (1 - \exp(-\gamma x))/(1 - \exp(-\gamma))$,
$$L_{\mathscr{D}}(\rho) \leq \inf_{\lambda > 1} \Phi_{\lambda/n}^{-1}\left(L_S(\rho) + \frac{\alpha}{\lambda}\left[\mathrm{KL}(\rho \| \pi) - \log \varepsilon + 2 \log \frac{\log(\alpha^2 \lambda)}{\log \alpha}\right]\right)$$
  - Can be much tighter (unfortunately) if $\mathrm{KL}(\rho \| \pi)/n$ is big

- Also variants based on general f-divergences, Wasserstein, …

Non-vacuous Generalization Bounds at the ImageNet Scale: a PAC-Bayesian Compression Approach

Wenda Zhou
Columbia University
New York, NY
wz2335@columbia.edu

Victor Veitch
Columbia University
New York, NY
victorveitch@gmail.com

Morgane Austern
Columbia University
New York, NY
ma3293@columbia.edu

Ryan P. Adams
Princeton University
Princeton, NJ
rpa@princeton.edu

Peter Orbanz
Columbia University
New York, NY
porbanz@stat.columbia.edu

- Pre-pick a coding scheme to represent networks (e.g. compress the weights)
- Train a network with SGD, sparsify it/etc to $\hat{h}$, then add a little noise to weights

NON-VACUOUS GENERALIZATION BOUNDS AT THE IMAGENET SCALE: A PAC-BAYESIAN COMPRESSION APPROACH

Wenda Zhou
Columbia University
New York, NY
wz2335@columbia.edu

Victor Veitch
Columbia University
New York, NY
victorveitch@gmail.com

Morgane Austern
Columbia University
New York, NY
ma3293@columbia.edu

Ryan P. Adams
Princeton University
Princeton, NJ
rpa@princeton.edu

Peter Orbanz
Columbia University
New York, NY
porbanz@stat.columbia.edu

- Pre-pick a coding scheme to represent networks (e.g. compress the weights)
- Train a network with SGD, sparsify it/etc to $\hat{h}$, then add a little noise to weights

Table 1: Summary of bounds obtained from compression

Dataset	Orig. size	Comp. size	Robust. Adj.	Eff. Size	Error Bound Top 1	Error Bound Top 5
MNIST	168.4 KiB	8.1 KiB	1.88 KiB	6.23 KiB	$< 46\,\%$	NA
ImageNet	5.93 MiB	452 KiB	102 KiB	350 KiB	$< 96.5\,\%$	$< 89\,\%$

Derandomizing PAC-Bayes

- In practice, we don't actually use randomized predictors (usually)

Derandomizing PAC-Bayes

- In practice, we don't actually use randomized predictors (usually)
- Possible to "derandomize" to a high-probability bound on $L_{\mathscr{D}}(h) - L_S(h)$:

Derandomizing PAC-Bayes

- In practice, we don't actually use randomized predictors (usually)
- Possible to "derandomize" to a high-probability bound on $L_{\mathscr{D}}(h) - L_S(h)$:
  - Show convergence of $L_{\mathscr{D}}(h)$ to $\mathbb{E}_{h\sim\rho} L_{\mathscr{D}}(h)$, $L_S(h)$ to $\mathbb{E}_{h\sim\rho} L_S(h)$, under ρ

Derandomizing PAC-Bayes

- In practice, we don't actually use randomized predictors (usually)
- Possible to "derandomize" to a high-probability bound on $L_{\mathscr{D}}(h) - L_S(h)$:
  - Show convergence of $L_{\mathscr{D}}(h)$ to $\mathbb{E}_{h\sim\rho} L_{\mathscr{D}}(h)$, $L_S(h)$ to $\mathbb{E}_{h\sim\rho} L_S(h)$, under ρ
  - Or, use a margin-type loss to show 0-1 error doesn't change under ρ

Derandomizing PAC-Bayes

- In practice, we don't actually use randomized predictors (usually)
- Possible to "derandomize" to a high-probability bound on $L_{\mathscr{D}}(h) - L_S(h)$:
  - Show convergence of $L_{\mathscr{D}}(h)$ to $\mathbb{E}_{h\sim\rho} L_{\mathscr{D}}(h)$, $L_S(h)$ to $\mathbb{E}_{h\sim\rho} L_S(h)$, under ρ
  - Or, use a margin-type loss to show 0-1 error doesn't change under ρ
- But…these then become "two-sided" bounds

Derandomizing PAC-Bayes

- In practice, we don't actually use randomized predictors (usually)
- Possible to "derandomize" to a high-probability bound on $L_{\mathscr{D}}(h) - L_S(h)$:
  - Show convergence of $L_{\mathscr{D}}(h)$ to $\mathbb{E}_{h \sim \rho} L_{\mathscr{D}}(h)$, $L_S(h)$ to $\mathbb{E}_{h \sim \rho} L_S(h)$, under ρ
  - Or, use a margin-type loss to show 0-1 error doesn't change under ρ
- But…these then become "two-sided" bounds
  - Subject to the Nagarajan/Kolter failure mode (their Appendix J)

Uniform convergence may be unable to explain generalization in deep learning

Vaishnavh Nagarajan
Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA
vaishnavh@cs.cmu.edu

J. Zico Kolter
Department of Computer Science
Carnegie Mellon University &
Bosch Center for Artificial Intelligence
Pittsburgh, PA
zkolter@cs.cmu.edu

(pause)

Online learning

- Class so far has been in the **(passive) batch setting**:
  - Observe training set $S \sim \mathscr{D}^n$, pick h, test on new examples from $\mathscr{D}$

Online learning

- Class so far has been in the **(passive) batch setting**:
  - Observe training set $S \sim \mathscr{D}^n$, pick h, test on new examples from $\mathscr{D}$
- Today: the **online** setting

Online learning

- Class so far has been in the **(passive) batch setting**:
  - Observe training set $S \sim \mathscr{D}^n$, pick h, test on new examples from $\mathscr{D}$

- Today: the **online** setting
  - See an x_t, make a prediction $\hat{y}_t$, see true label y_t, repeat

Online learning

- Class so far has been in the **(passive) batch setting**:
  - Observe training set $S \sim \mathscr{D}^n$, pick h, test on new examples from $\mathscr{D}$

- Today: the **online** setting
  - See an x_t, make a prediction $\hat{y}_t$, see true label y_t, repeat
  - We learn how to predict as we go

Online learning

- Class so far has been in the **(passive) batch setting**:
  - Observe training set $S \sim \mathscr{D}^n$, pick h, test on new examples from $\mathscr{D}$

- Today: the **online** setting
  - See an x_t, make a prediction $\hat{y}_t$, see true label y_t, repeat
  - We learn how to predict as we go
  - Focusing on binary classification to start

Online learning

- Class so far has been in the **(passive) batch setting**:
  - Observe training set $S \sim \mathscr{D}^n$, pick h, test on new examples from $\mathscr{D}$

- Today: the **online** setting
  - See an x_t, make a prediction $\hat{y}_t$, see true label y_t, repeat
  - We learn how to predict as we go
  - Focusing on binary classification to start
  - Usual analysis does *not* assume a fixed distribution $\mathscr{D}$

Online learning

- Class so far has been in the **(passive) batch setting**:
  - Observe training set $S \sim \mathscr{D}^n$, pick h, test on new examples from $\mathscr{D}$
- Today: the **online** setting
  - See an x_t, make a prediction $\hat{y}_t$, see true label y_t, repeat
  - We learn how to predict as we go
  - Focusing on binary classification to start
  - Usual analysis does *not* assume a fixed distribution $\mathscr{D}$
    - Labels can even be chosen **adversarially**

Online learning

- Class so far has been in the **(passive) batch setting**:
  - Observe training set $S \sim \mathscr{D}^n$, pick h, test on new examples from $\mathscr{D}$

- Today: the **online** setting
  - See an x_t, make a prediction $\hat{y}_t$, see true label y_t, repeat
  - We learn how to predict as we go
  - Focusing on binary classification to start
  - Usual analysis does *not* assume a fixed distribution $\mathscr{D}$
    - Labels can even be chosen **adversarially**

Hello Danica,

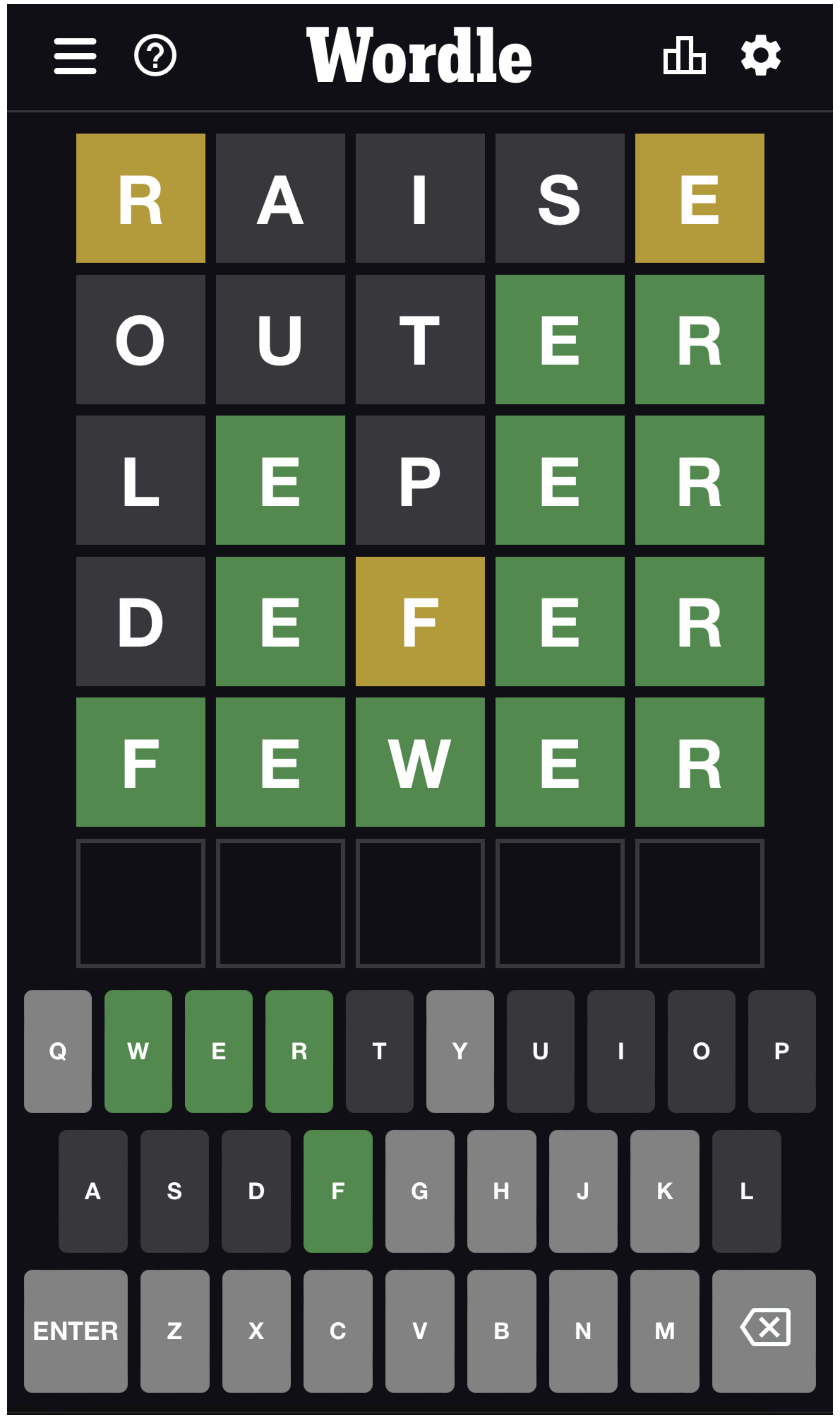
I am incredibly sorry about this! It looks like the earlier CMT emails went to my spam folder. I can do this review within the next 12 hours (i.e. by midnight

Realizable online setting

- **Realizable** setting: labels y_t have to be consistent with some $h^* \in \mathscr{H}$

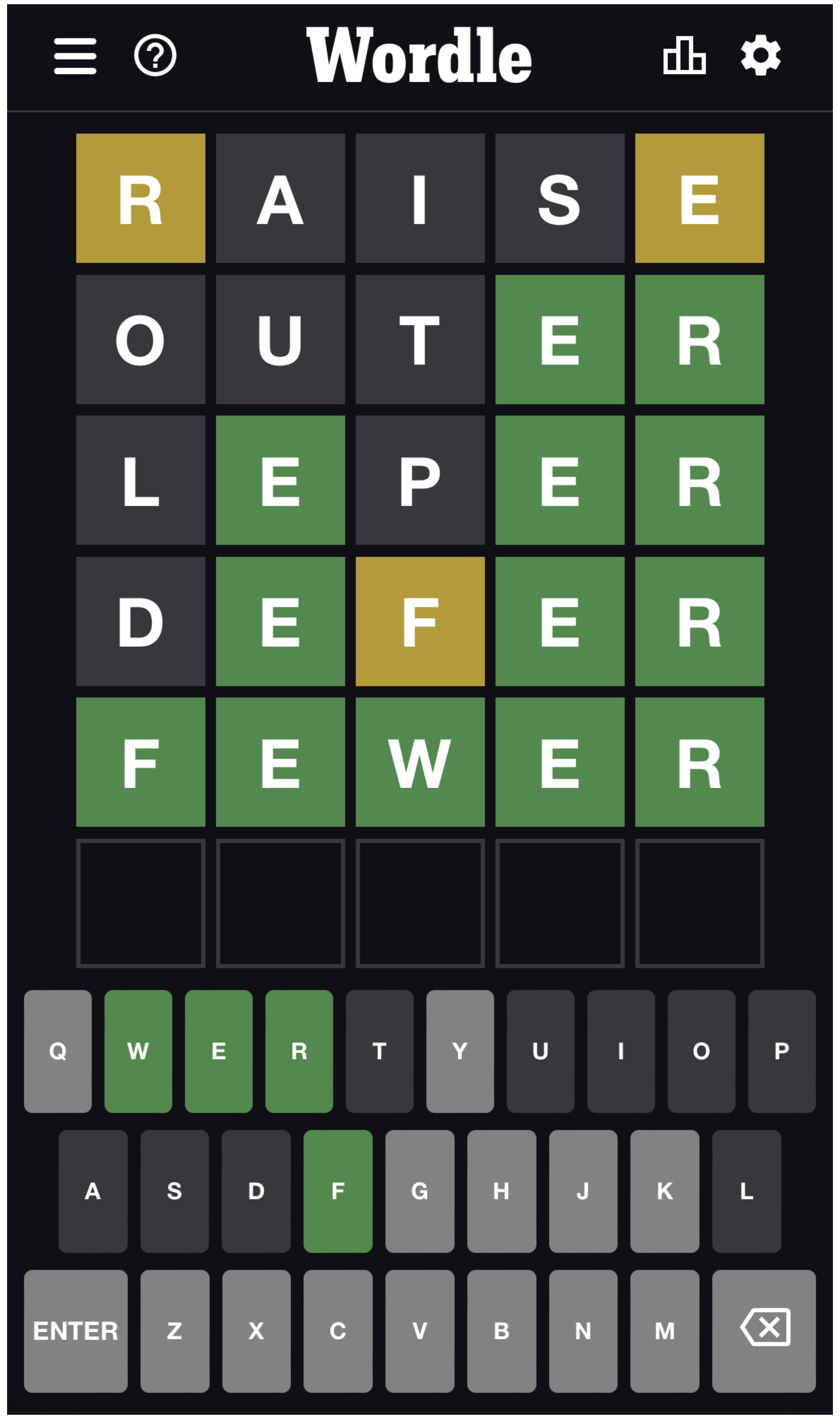
Realizable online setting

- **Realizable** setting: labels y_t have to be consistent with some $h^* \in \mathscr{H}$



Realizable online setting

- **Realizable** setting: labels y_t have to be consistent with some $h^* \in \mathscr{H}$



Mistake bounds

- Take a sequence $S = \big((x_1, h^*(x_1)), \ldots, (x_T, h^*(x_T))\big)$

Mistake bounds

- Take a sequence $S = \big((x_1, h^*(x_1)), \ldots, (x_T, h^*(x_T))\big)$
- $M_A(S)$ is the number of **mistakes** the algorithm A makes on S

Mistake bounds

- Take a sequence $S = \big((x_1, h^*(x_1)), \ldots, (x_T, h^*(x_T))\big)$
- $M_A(S)$ is the number of **mistakes** the algorithm A makes on S
- $M_A(\mathcal{H})$ is the **worst-case** number of mistakes for **any** S with labels in $\mathcal{H}$

Mistake bounds

- Take a sequence $S = \big((x_1, h^*(x_1)), \ldots, (x_T, h^*(x_T))\big)$
- $M_A(S)$ is the number of **mistakes** the algorithm A makes on S
- $M_A(\mathscr{H})$ is the **worst-case** number of mistakes for **any** S with labels in $\mathscr{H}$
- $\mathscr{H}$ is **online learnable** if there's an A with $M_A(\mathscr{H}) < \infty$

Mistake bounds

- Take a sequence $S = \big((x_1, h^*(x_1)), \ldots, (x_T, h^*(x_T))\big)$
- $M_A(S)$ is the number of **mistakes** the algorithm A makes on S
- $M_A(\mathscr{H})$ is the **worst-case** number of mistakes for **any** S with labels in $\mathscr{H}$
- $\mathscr{H}$ is **online learnable** if there's an A with $M_A(\mathscr{H}) < \infty$

- If $\mathscr{H}$ is finite, consider the algorithm `Consistent` (basically ERM):

Mistake bounds

- Take a sequence $S = \big((x_1, h^*(x_1)), \ldots, (x_T, h^*(x_T))\big)$
- $M_A(S)$ is the number of **mistakes** the algorithm A makes on S
- $M_A(\mathscr{H})$ is the **worst-case** number of mistakes for **any** S with labels in $\mathscr{H}$
- $\mathscr{H}$ is **online learnable** if there's an A with $M_A(\mathscr{H}) < \infty$

- If $\mathscr{H}$ is finite, consider the algorithm `Consistent` (basically ERM):
  - Start with the **version space** $V_1 = \mathscr{H}$

Mistake bounds

- Take a sequence $S = \big((x_1, h^*(x_1)), \ldots, (x_T, h^*(x_T))\big)$
- $M_A(S)$ is the number of **mistakes** the algorithm A makes on S
- $M_A(\mathscr{H})$ is the **worst-case** number of mistakes for **any** S with labels in $\mathscr{H}$
- $\mathscr{H}$ is **online learnable** if there's an A with $M_A(\mathscr{H}) < \infty$

- If $\mathscr{H}$ is finite, consider the algorithm `Consistent` (basically ERM):
  - Start with the **version space** $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t = h(x_t)$ for any arbitrary $h \in V_t$

Mistake bounds

- Take a sequence $S = \big((x_1, h^*(x_1)), \ldots, (x_T, h^*(x_T))\big)$
- $M_A(S)$ is the number of **mistakes** the algorithm A makes on S
- $M_A(\mathscr{H})$ is the **worst-case** number of mistakes for **any** S with labels in $\mathscr{H}$
- $\mathscr{H}$ is **online learnable** if there's an A with $M_A(\mathscr{H}) < \infty$

- If $\mathscr{H}$ is finite, consider the algorithm `Consistent` (basically ERM):
  - Start with the **version space** $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t = h(x_t)$ for any arbitrary $h \in V_t$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

Mistake bounds

- Take a sequence $S = \big((x_1, h^*(x_1)), \ldots, (x_T, h^*(x_T))\big)$
- $M_A(S)$ is the number of **mistakes** the algorithm A makes on S
- $M_A(\mathcal{H})$ is the **worst-case** number of mistakes for **any** S with labels in $\mathcal{H}$
- $\mathcal{H}$ is **online learnable** if there's an A with $M_A(\mathcal{H}) < \infty$

- If $\mathcal{H}$ is finite, consider the algorithm `Consistent` (basically ERM):
  - Start with the **version space** $V_1 = \mathcal{H}$
  - Given x_t, predict $\hat{y}_t = h(x_t)$ for any arbitrary $h \in V_t$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
- Have mistake bound $M_{\texttt{Consistent}}(\mathcal{H}) \leq |\mathcal{H}| - 1$

A smarter algorithm for finite, realizable $\mathscr{H}$

- If `Consistent` made a mistake, we might only remove **one** h from V_t
- Better algorithm can always either (a) be right or (b) make lots of progress

A smarter algorithm for finite, realizable $\mathscr{H}$

- If `Consistent` made a mistake, we might only remove **one** h from V_t
- Better algorithm can always either (a) be right or (b) make lots of progress
- `Halving`:
  - Start with the version space $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t \in \operatorname{argmax}_{r \in \{0,1\}} \left| \left\{ h \in V_t : h(x_t) = r \right\} \right|$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

A smarter algorithm for finite, realizable $\mathscr{H}$

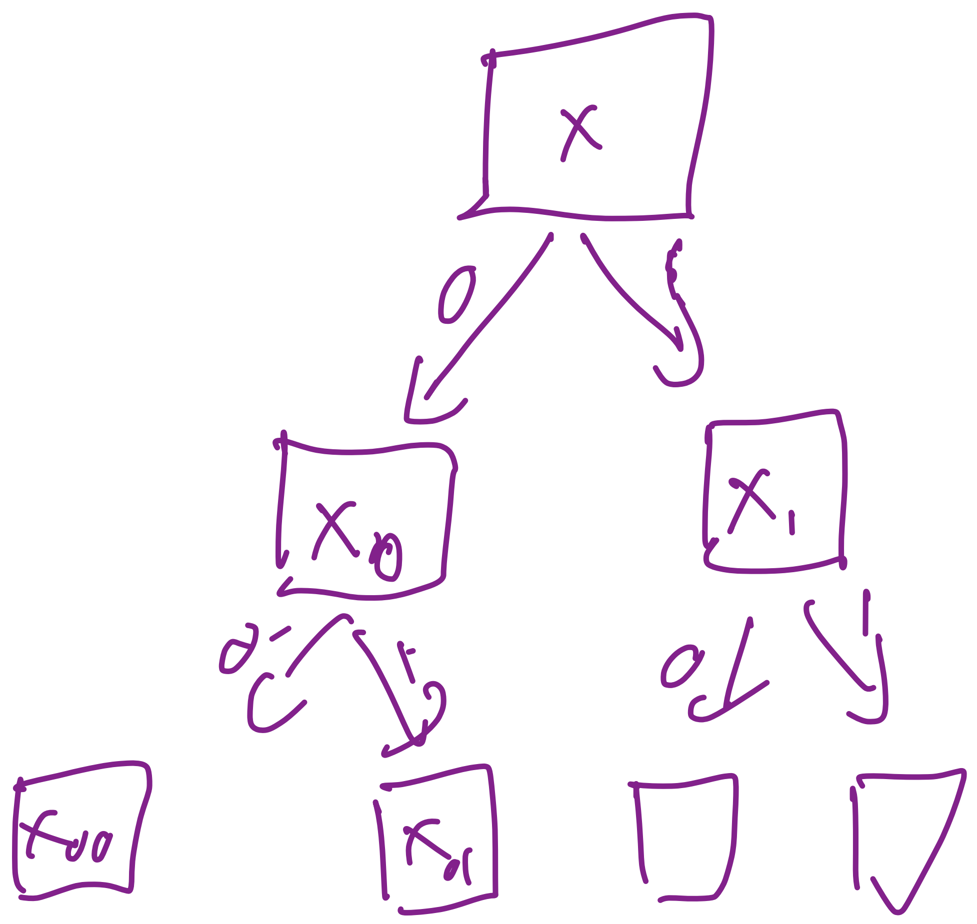
- If `Consistent` made a mistake, we might only remove **one** h from V_t
- Better algorithm can always either (a) be right or (b) make lots of progress
- `Halving`:
  - Start with the version space $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t \in \operatorname{argmax}_{r \in \{0,1\}} \left| \left\{ h \in V_t : h(x_t) = r \right\} \right|$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
- If we were wrong, we removed **at least half** of V_t

A smarter algorithm for finite, realizable $\mathscr{H}$

- If `Consistent` made a mistake, we might only remove **one** h from V_t
- Better algorithm can always either (a) be right or (b) make lots of progress
- `Halving`:
  - Start with the version space $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t \in \operatorname{argmax}_{r \in \{0,1\}} \left| \left\{ h \in V_t : h(x_t) = r \right\} \right|$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
- If we were wrong, we removed **at least half** of V_t
- $M_{\texttt{Halving}}(\mathscr{H}) \leq \log_2 |\mathscr{H}|$ – *way* better bound

Online learnability

- Think about the **game tree** for the learner and the **adversary**
  - Put points $x_t \in \mathcal{X}$ into a full binary tree
  - Start at the root, move left if learner predicts 0, right if it predicts 1



Online learnability

- Think about the **game tree** for the learner and the **adversary**
  - Put points $x_t \in \mathscr{X}$ into a full binary tree
  - Start at the root, move left if learner predicts 0, right if it predicts 1
- $\mathscr{H}$ **shatters a tree** if everywhere in the tree is reached by some $h \in \mathscr{H}$

Online learnability

- Think about the **game tree** for the learner and the **adversary**
  - Put points $x_t \in \mathscr{X}$ into a full binary tree
  - Start at the root, move left if learner predicts 0, right if it predicts 1
- $\mathscr{H}$ **shatters a tree** if everywhere in the tree is reached by some $h \in \mathscr{H}$
- The **Littlestone dimension** $\mathrm{Ldim}(\mathscr{H})$ is the max depth of any tree $\mathscr{H}$ shatters

Online learnability

- Think about the **game tree** for the learner and the **adversary**
  - Put points $x_t \in \mathcal{X}$ into a full binary tree
  - Start at the root, move left if learner predicts 0, right if it predicts 1
- $\mathcal{H}$ **shatters a tree** if everywhere in the tree is reached by some $h \in \mathcal{H}$
- The **Littlestone dimension** $\mathrm{Ldim}(\mathcal{H})$ is the max depth of any tree $\mathcal{H}$ shatters
- Any algorithm A must have $M_A(\mathcal{H}) \geq \mathrm{Ldim}(\mathcal{H})$

Online learnability

- Think about the **game tree** for the learner and the **adversary**
  - Put points $x_t \in \mathscr{X}$ into a full binary tree
  - Start at the root, move left if learner predicts 0, right if it predicts 1
- $\mathscr{H}$ **shatters a tree** if everywhere in the tree is reached by some $h \in \mathscr{H}$
- The **Littlestone dimension** $\mathrm{Ldim}(\mathscr{H})$ is the max depth of any tree $\mathscr{H}$ shatters
- Any algorithm A must have $M_A(\mathscr{H}) \geq \mathrm{Ldim}(\mathscr{H})$
- If $\mathscr{H}$ can shatter a set, it can shatter any tree from that set

Online learnability

- Think about the **game tree** for the learner and the **adversary**
  - Put points $x_t \in \mathcal{X}$ into a full binary tree
  - Start at the root, move left if learner predicts 0, right if it predicts 1
- $\mathcal{H}$ **shatters a tree** if everywhere in the tree is reached by some $h \in \mathcal{H}$
- The **Littlestone dimension** $\mathrm{Ldim}(\mathcal{H})$ is the max depth of any tree $\mathcal{H}$ shatters
- Any algorithm A must have $M_A(\mathcal{H}) \geq \mathrm{Ldim}(\mathcal{H})$
- If $\mathcal{H}$ can shatter a set, it can shatter any tree from that set
  - $\mathrm{VCdim}(\mathcal{H}) \leq \mathrm{Ldim}(\mathcal{H})$

Littlestone dimension examples

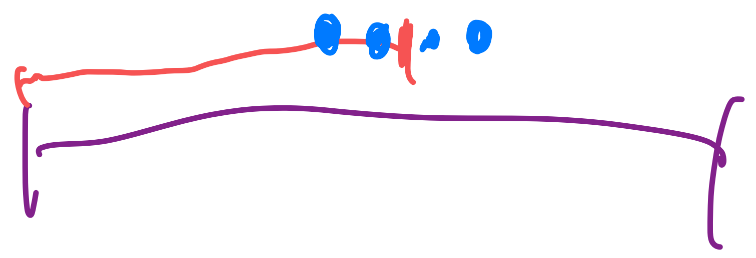
- If $\mathscr{H}$ is finite, can't shatter a full tree deeper than $\log_2 |\mathscr{H}|$

Littlestone dimension examples

- If $\mathscr{H}$ is finite, can't shatter a full tree deeper than $\log_2|\mathscr{H}|$
- If $\mathscr{X} = [d]$, $\mathscr{H} = \{x \mapsto \mathbb{I}(x = i) : i \in [d]\}$, have $\mathrm{Ldim}(\mathscr{H}) = 1$

Littlestone dimension examples

- If $\mathscr{H}$ is finite, can't shatter a full tree deeper than $\log_2|\mathscr{H}|$
- If $\mathscr{X} = [d]$, $\mathscr{H} = \{x \mapsto \mathbb{1}(x = i) : i \in [d]\}$, have $\mathrm{Ldim}(\mathscr{H}) = 1$
- If $\mathscr{X} = [0,1]$ and $\mathscr{H} = \{x \mapsto \mathbb{1}(x \leq a) : a \in [0,1]\}$, have $\mathrm{Ldim}(\mathscr{H}) = \infty$ (!)



Standard Optimal Algorithm

- Like `Halving`, but tries to reduce Littlestone dimension instead of cardinality:
  - Start with the version space $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t \in \mathrm{argmax}_{r\in\{0,1\}} \mathrm{Ldim}\left(\left\{h \in V_t : h(x_t) = r\right\}\right)$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$

Standard Optimal Algorithm

- Like `Halving`, but tries to reduce Littlestone dimension instead of cardinality:
  - Start with the version space $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t \in \operatorname{argmax}_{r \in \{0,1\}} \operatorname{Ldim}\left(\left\{h \in V_t : h(x_t) = r\right\}\right)$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
- Whenever we make a mistake, $\operatorname{Ldim}(V_{t+1}) \leq \operatorname{Ldim}(V_t) - 1$:

Standard Optimal Algorithm

- Like `Halving`, but tries to reduce Littlestone dimension instead of cardinality:
  - Start with the version space $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t \in \operatorname{argmax}_{r \in \{0,1\}} \mathrm{Ldim}\left(\left\{h \in V_t : h(x_t) = r\right\}\right)$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
- Whenever we make a mistake, $\mathrm{Ldim}(V_{t+1}) \leq \mathrm{Ldim}(V_t) - 1$:
  - If not, $\mathrm{Ldim}\left(\left\{h \in V_t : h(x_t) = 0\right\}\right) = \mathrm{Ldim}(V_t) = \mathrm{Ldim}\left(\left\{h \in V_t : h(x_t) = 1\right\}\right)$

Standard Optimal Algorithm

- Like `Halving`, but tries to reduce Littlestone dimension instead of cardinality:
  - Start with the version space $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t \in \operatorname{argmax}_{r \in \{0,1\}} \mathrm{Ldim}\left(\left\{h \in V_t : h(x_t) = r\right\}\right)$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
- Whenever we make a mistake, $\mathrm{Ldim}(V_{t+1}) \leq \mathrm{Ldim}(V_t) - 1$:
  - If not, $\mathrm{Ldim}\left(\left\{h \in V_t : h(x_t) = 0\right\}\right) = \mathrm{Ldim}(V_t) = \mathrm{Ldim}\left(\left\{h \in V_t : h(x_t) = 1\right\}\right)$
  - Then combine shattered trees into one shattered tree of depth $\mathrm{Ldim}(V_t) + 1$

Standard Optimal Algorithm

- Like `Halving`, but tries to reduce Littlestone dimension instead of cardinality:
  - Start with the version space $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t \in \operatorname{argmax}_{r \in \{0,1\}} \operatorname{Ldim}\left(\left\{h \in V_t : h(x_t) = r\right\}\right)$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
- Whenever we make a mistake, $\operatorname{Ldim}(V_{t+1}) \leq \operatorname{Ldim}(V_t) - 1$:
  - If not, $\operatorname{Ldim}\left(\left\{h \in V_t : h(x_t) = 0\right\}\right) = \operatorname{Ldim}(V_t) = \operatorname{Ldim}\left(\left\{h \in V_t : h(x_t) = 1\right\}\right)$
  - Then combine shattered trees into one shattered tree of depth $\operatorname{Ldim}(V_t) + 1$
  - But then $\operatorname{Ldim}(V_t) = \operatorname{Ldim}(V_t) + 1$…contradiction

Standard Optimal Algorithm

- Like `Halving`, but tries to reduce Littlestone dimension instead of cardinality:
  - Start with the version space $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t \in \operatorname{argmax}_{r \in \{0,1\}} \mathrm{Ldim}\left(\left\{h \in V_t : h(x_t) = r\right\}\right)$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
- Whenever we make a mistake, $\mathrm{Ldim}(V_{t+1}) \leq \mathrm{Ldim}(V_t) - 1$:
  - If not, $\mathrm{Ldim}\left(\left\{h \in V_t : h(x_t) = 0\right\}\right) = \mathrm{Ldim}(V_t) = \mathrm{Ldim}\left(\left\{h \in V_t : h(x_t) = 1\right\}\right)$
  - Then combine shattered trees into one shattered tree of depth $\mathrm{Ldim}(V_t) + 1$
  - But then $\mathrm{Ldim}(V_t) = \mathrm{Ldim}(V_t) + 1$…contradiction
- Thus $M_{\mathtt{SOA}}(\mathscr{H}) = \mathrm{Ldim}(\mathscr{H})$, the best possible mistake bound

Standard Optimal Algorithm

- Like `Halving`, but tries to reduce Littlestone dimension instead of cardinality:
  - Start with the version space $V_1 = \mathscr{H}$
  - Given x_t, predict $\hat{y}_t \in \operatorname{argmax}_{r \in \{0,1\}} \operatorname{Ldim}\left(\left\{h \in V_t : h(x_t) = r\right\}\right)$
  - Seeing y_t, update $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$
- Whenever we make a mistake, $\operatorname{Ldim}(V_{t+1}) \leq \operatorname{Ldim}(V_t) - 1$:
  - If not, $\operatorname{Ldim}\left(\left\{h \in V_t : h(x_t) = 0\right\}\right) = \operatorname{Ldim}(V_t) = \operatorname{Ldim}\left(\left\{h \in V_t : h(x_t) = 1\right\}\right)$
  - Then combine shattered trees into one shattered tree of depth $\operatorname{Ldim}(V_t) + 1$
  - But then $\operatorname{Ldim}(V_t) = \operatorname{Ldim}(V_t) + 1$…contradiction
- Thus $M_{\texttt{SOA}}(\mathscr{H}) = \operatorname{Ldim}(\mathscr{H})$, the best possible mistake bound
- But SOA is not necessarily easy to compute!

(pause)

Unrealizable online learning

- In the batch setting:
  - Realizable PAC assumes labels come from $h^* \in \mathscr{H}$
  - Agnostic PAC just has us **compete with** best $h^* \in \mathscr{H}$
- In the online setting:
  - Realizable assumes labels come from $h^* \in \mathscr{H}$

Unrealizable online learning

- In the batch setting:
  - Realizable PAC assumes labels come from $h^* \in \mathscr{H}$
  - Agnostic PAC just has us **compete with** best $h^* \in \mathscr{H}$
- In the online setting:
  - Realizable assumes labels come from $h^* \in \mathscr{H}$
  - Unrealizable has us compete with best $h^* \in \mathscr{H}$

Unrealizable online learning

- In the batch setting:
  - Realizable PAC assumes labels come from $h^* \in \mathscr{H}$
  - Agnostic PAC just has us **compete with** best $h^* \in \mathscr{H}$
- In the online setting:
  - Realizable assumes labels come from $h^* \in \mathscr{H}$
  - Unrealizable has us compete with best $h^* \in \mathscr{H}$

$$\text{Regret}_A(h, T) = \sup_{(x_1,y_1),\ldots,(x_T,y_T)} \left[\sum_{t=1}^{T} |\hat{y}_t - y_t| - \sum_{t=1}^{T} |h(x_t) - y_t| \right]$$

Unrealizable online learning

- In the batch setting:
  - Realizable PAC assumes labels come from $h^* \in \mathscr{H}$
  - Agnostic PAC just has us **compete with** best $h^* \in \mathscr{H}$
- In the online setting:
  - Realizable assumes labels come from $h^* \in \mathscr{H}$
  - Unrealizable has us compete with best $h^* \in \mathscr{H}$

$$\mathrm{Regret}_A(h, T) = \sup_{(x_1,y_1),\ldots,(x_T,y_T)} \left[\sum_{t=1}^{T} |\hat{y}_t - y_t| - \sum_{t=1}^{T} |h(x_t) - y_t| \right]$$

$$\mathrm{Regret}_A(\mathscr{H}, T) = \sup_{h \in \mathscr{H}} \mathrm{Regret}_A(h, T)$$

Unrealizable online learning

- In the batch setting:
  - Realizable PAC assumes labels come from $h^* \in \mathcal{H}$
  - Agnostic PAC just has us **compete with** best $h^* \in \mathcal{H}$
- In the online setting:
  - Realizable assumes labels come from $h^* \in \mathcal{H}$
  - Unrealizable has us compete with best $h^* \in \mathcal{H}$

$$\mathrm{Regret}_A(h, T) = \sup_{(x_1,y_1),\ldots,(x_T,y_T)} \left[\sum_{t=1}^{T} |\hat{y}_t - y_t| - \sum_{t=1}^{T} |h(x_t) - y_t| \right]$$

$$\mathrm{Regret}_A(\mathcal{H}, T) = \sup_{h \in \mathcal{H}} \mathrm{Regret}_A(h, T)$$

- Ideally, we want **sublinear regret**: $\frac{1}{T} \mathrm{Regret}_A(\mathcal{H}, T) \xrightarrow{T \to \infty} 0$

Regret: impossible to avoid

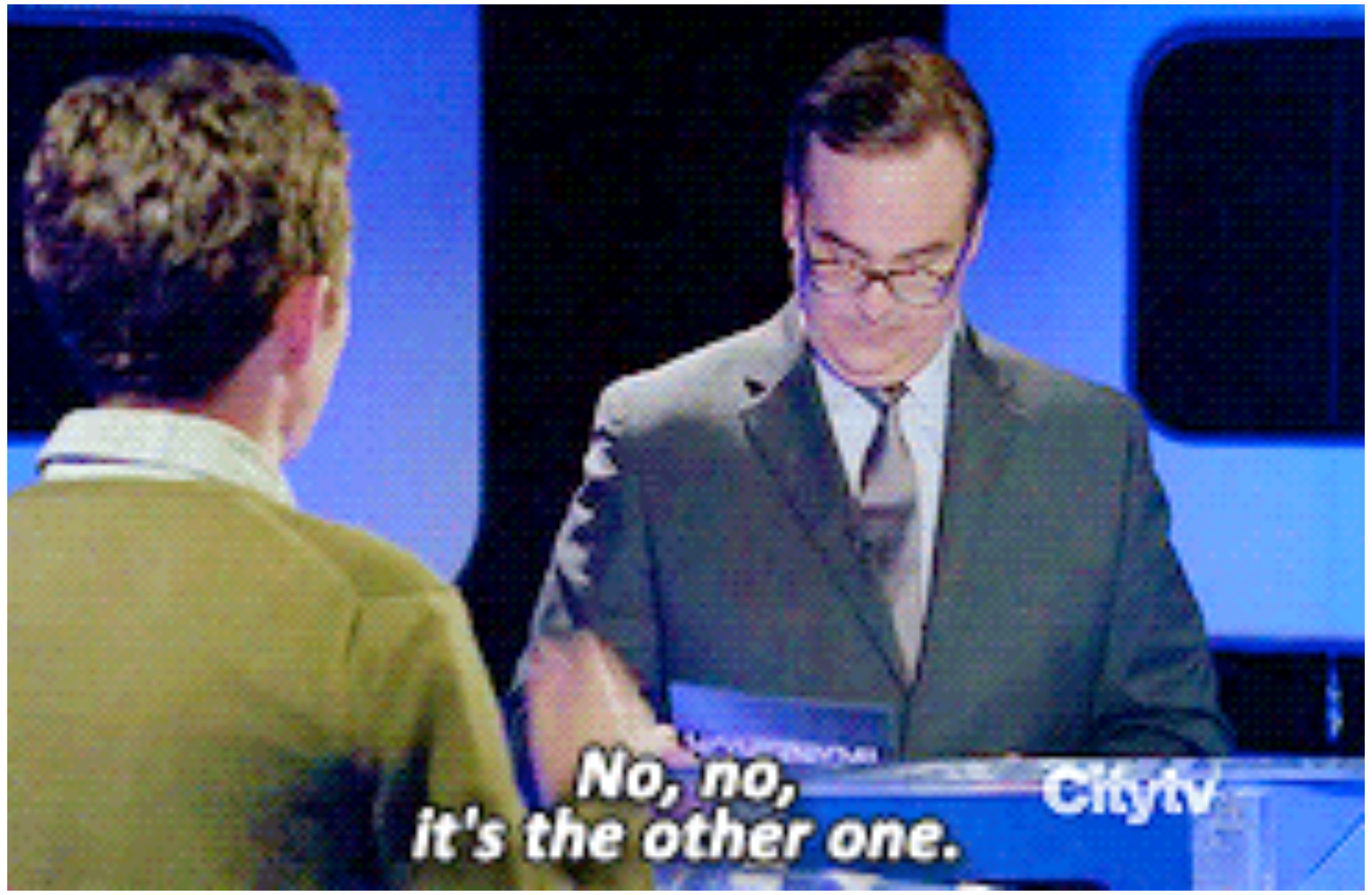
- Regret: “how much better it would have been to just play $h(x_t)$ every time”
- Consider $\mathscr{H} = \{x \mapsto 0, x \mapsto 1\}$

Regret: impossible to avoid

- Regret: “how much better it would have been to just play $h(x_t)$ every time”
- Consider $\mathscr{H} = \{x \mapsto 0, x \mapsto 1\}$
  - Adversary could always just say “no, you’re wrong” and get T mistakes

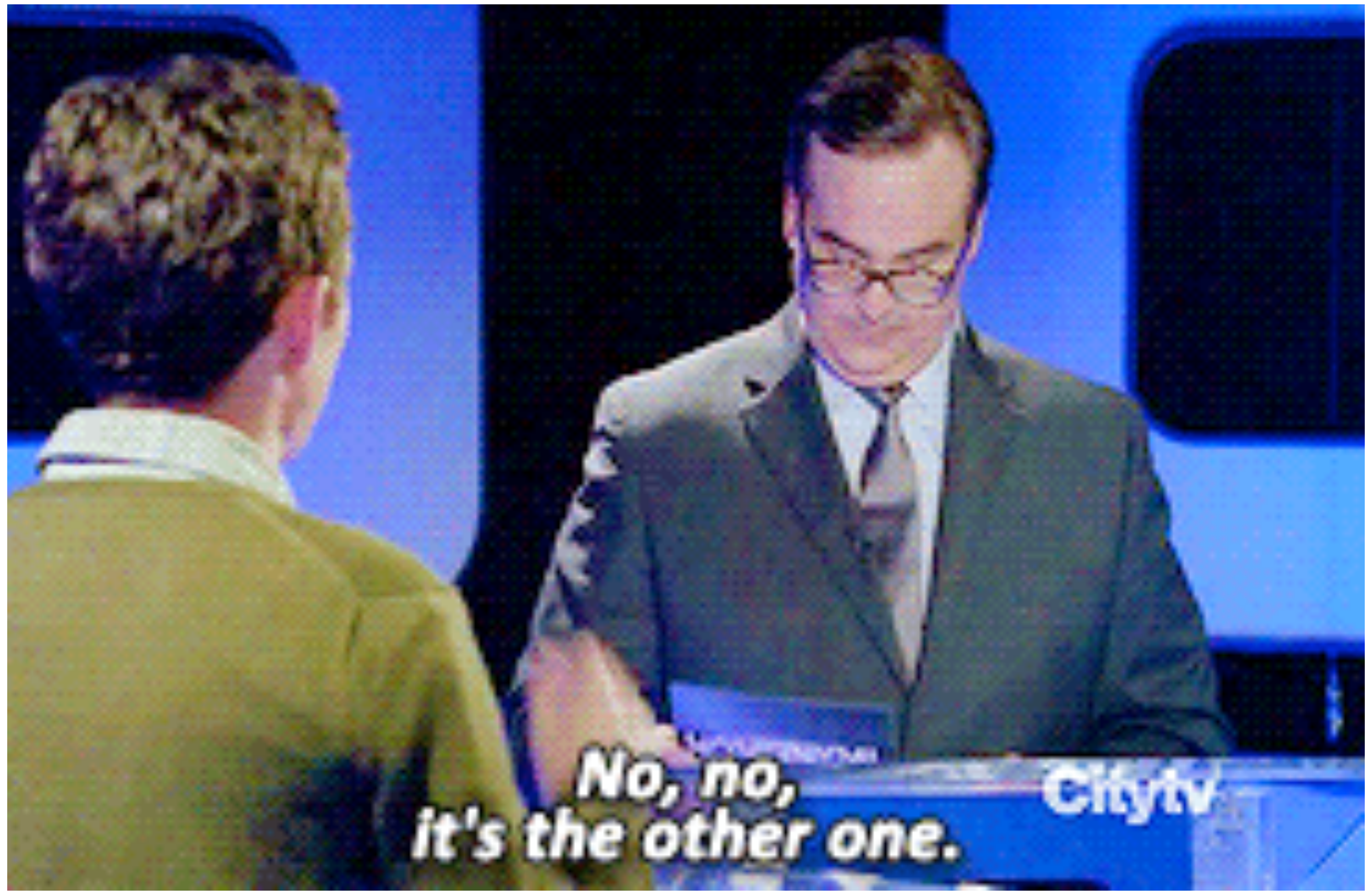
Regret: impossible to avoid

- Regret: "how much better it would have been to just play $h(x_t)$ every time"
- Consider $\mathscr{H} = \{x \mapsto 0, x \mapsto 1\}$
  - Adversary could always just say "no, you're wrong" and get T mistakes



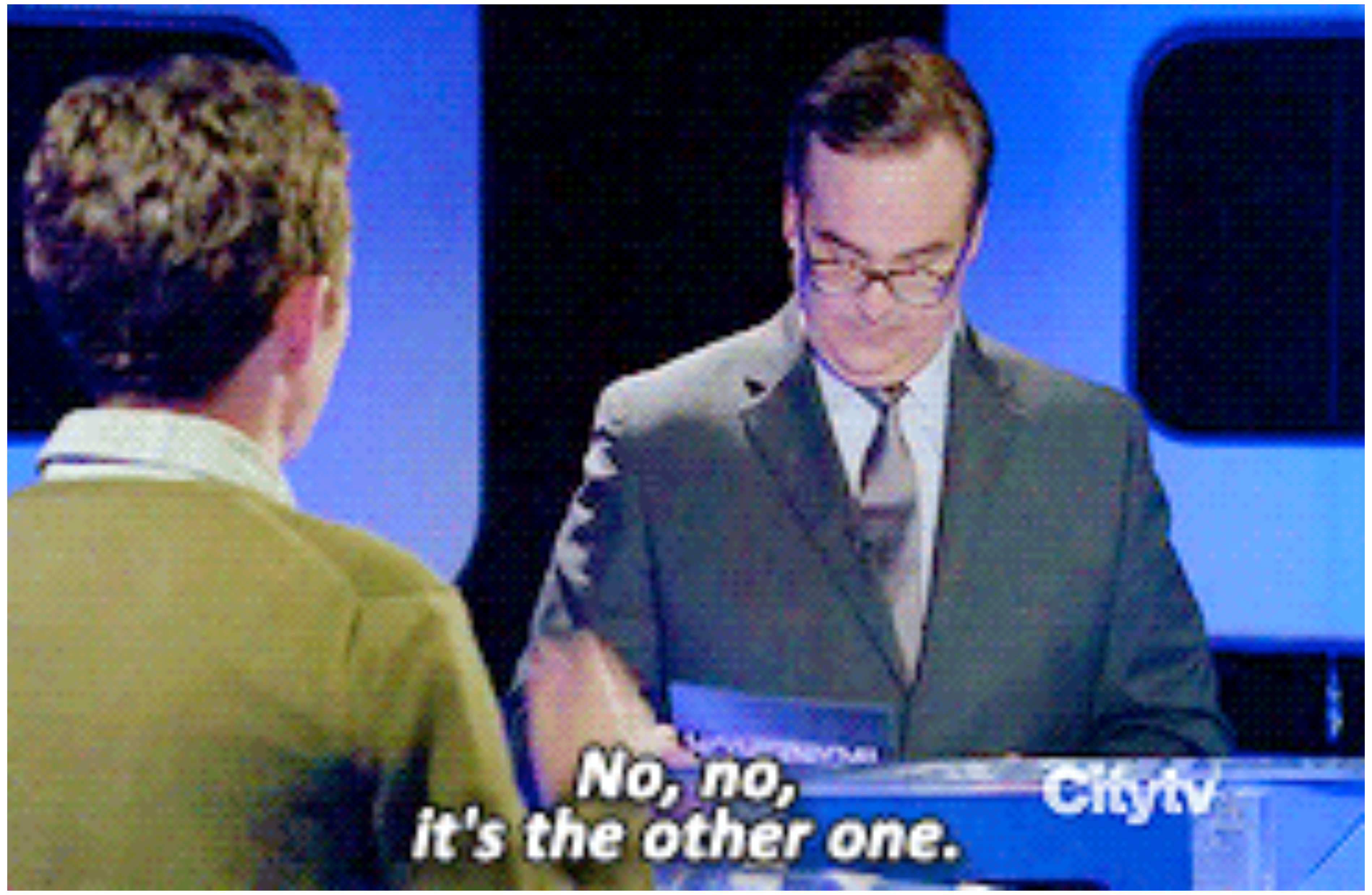
Regret: impossible to avoid

- Regret: “how much better it would have been to just play $h(x_t)$ every time”
- Consider $\mathscr{H} = \{x \mapsto 0, x \mapsto 1\}$
  - Adversary could always just say “no, you’re wrong” and get T mistakes
  - For any sequence of true y_t, either $x \mapsto 0$ or $x \mapsto 1$ has $\leq \frac{T}{2}$ mistakes



Regret: impossible to avoid

- Regret: "how much better it would have been to just play $h(x_t)$ every time"
- Consider $\mathscr{H} = \{x \mapsto 0, x \mapsto 1\}$
  - Adversary could always just say "no, you're wrong" and get T mistakes
  - For any sequence of true y_t, either $x \mapsto 0$ or $x \mapsto 1$ has $\leq \frac{T}{2}$ mistakes
  - So regret would be at least $T - \frac{T}{2} = \frac{T}{2}$



Regret: impossible to avoid

- Regret: "how much better it would have been to just play $h(x_t)$ every time"
- Consider $\mathscr{H} = \{x \mapsto 0, x \mapsto 1\}$
  - Adversary could always just say "no, you're wrong" and get T mistakes
  - For any sequence of true y_t, either $x \mapsto 0$ or $x \mapsto 1$ has $\leq \frac{T}{2}$ mistakes
  - So regret would be at least $T - \frac{T}{2} = \frac{T}{2}$
- To avoid this:

Regret: impossible to avoid

- Regret: "how much better it would have been to just play $h(x_t)$ every time"
- Consider $\mathscr{H} = \{x \mapsto 0, x \mapsto 1\}$
  - Adversary could always just say "no, you're wrong" and get T mistakes
  - For any sequence of true y_t, either $x \mapsto 0$ or $x \mapsto 1$ has $\leq \frac{T}{2}$ mistakes
  - So regret would be at least $T - \frac{T}{2} = \frac{T}{2}$
- To avoid this:
  - Learner has **random** prediction, $\Pr(\hat{y}_t = 1) = p_t$

Regret: impossible to avoid

- Regret: "how much better it would have been to just play $h(x_t)$ every time"
- Consider $\mathscr{H} = \{x \mapsto 0, x \mapsto 1\}$
  - Adversary could always just say "no, you're wrong" and get T mistakes
  - For any sequence of true y_t, either $x \mapsto 0$ or $x \mapsto 1$ has $\leq \frac{T}{2}$ mistakes
  - So regret would be at least $T - \frac{T}{2} = \frac{T}{2}$
- To avoid this:
  - Learner has **random** prediction, $\Pr(\hat{y}_t = 1) = p_t$
  - Adversary commits to y_t without knowing the roll

Regret: impossible to avoid

- Regret: "how much better it would have been to just play $h(x_t)$ every time"
- Consider $\mathscr{H} = \{x \mapsto 0, x \mapsto 1\}$
  - Adversary could always just say "no, you're wrong" and get T mistakes
  - For any sequence of true y_t, either $x \mapsto 0$ or $x \mapsto 1$ has $\leq \frac{T}{2}$ mistakes
  - So regret would be at least $T - \frac{T}{2} = \frac{T}{2}$
- To avoid this:
  - Learner has **random** prediction, $\Pr(\hat{y}_t = 1) = p_t$
  - Adversary commits to y_t without knowing the roll

Regret: impossible to avoid

- Regret: "how much better it would have been to just play $h(x_t)$ every time"
- Consider $\mathscr{H} = \{x \mapsto 0, x \mapsto 1\}$
  - Adversary could always just say "no, you're wrong" and get T mistakes
  - For any sequence of true y_t, either $x \mapsto 0$ or $x \mapsto 1$ has $\leq \frac{T}{2}$ mistakes
  - So regret would be at least $T - \frac{T}{2} = \frac{T}{2}$
- To avoid this:
  - Learner has **random** prediction, $\Pr(\hat{y}_t = 1) = p_t$
  - Adversary commits to y_t without knowing the roll
  - Measure **expected** loss $\Pr(\hat{y}_t \neq y_t) = |p_t - y_t|$

Low regret for online classification

- For every $\mathcal{H}$, there's an algorithm with
  $\text{Regret}_A(\mathcal{H}, T) \leq \sqrt{2 \min\left(\log|\mathcal{H}|, (1 + \log T)\ \text{Ldim}(\mathcal{H})\right) T}$
- Also a lower bound of $\Omega\left(\sqrt{\text{Ldim}(\mathcal{H})\ T}\right)$
- Based on Weighted-Majority algorithm for **learning with expert advice**

Learning from expert advice

- There are d available experts who make predictions

Learning from expert advice

- There are d available experts who make predictions
- At time t, learner chooses to follow expert i with probability $(w_t)_i$

Learning from expert advice

- There are d available experts who make predictions
- At time t, learner chooses to follow expert i with probability $(w_t)_i$
- Sees potential costs $v_t \in \mathbb{R}^d$; pays expectation $\langle w_t, v_t \rangle$

Learning from expert advice

- There are d available experts who make predictions
- At time t, learner chooses to follow expert i with probability $(w_t)_i$
- Sees potential costs $v_t \in \mathbb{R}^d$; pays expectation $\langle w_t, v_t \rangle$
- `Weighted-Majority` algorithm:

Learning from expert advice

- There are d available experts who make predictions
- At time t, learner chooses to follow expert i with probability $(w_t)_i$
- Sees potential costs $v_t \in \mathbb{R}^d$; pays expectation $\langle w_t, v_t \rangle$
- `Weighted-Majority` algorithm:
  - Start with $\tilde{w}_1 = (1,\ldots,1)$; $\eta = \sqrt{2\log(d)/T}$

Learning from expert advice

- There are d available experts who make predictions
- At time t, learner chooses to follow expert i with probability $(w_t)_i$
- Sees potential costs $v_t \in \mathbb{R}^d$; pays expectation $\langle w_t, v_t \rangle$
- `Weighted-Majority` algorithm:
  - Start with $\tilde{w}_1 = (1,\ldots,1)$; $\eta = \sqrt{2\log(d)/T}$
  - For $t = 1,2,\ldots$

Learning from expert advice

- There are d available experts who make predictions
- At time t, learner chooses to follow expert i with probability $(w_t)_i$
- Sees potential costs $v_t \in \mathbb{R}^d$; pays expectation $\langle w_t, v_t \rangle$
- `Weighted-Majority` algorithm:
  - Start with $\tilde{w}_1 = (1,\ldots,1)$; $\eta = \sqrt{2\log(d)/T}$
  - For $t = 1,2,\ldots$
    - Follow with probabilities $w_t = \tilde{w}_t / \|w_t\|_1$

Learning from expert advice

- There are d available experts who make predictions
- At time t, learner chooses to follow expert i with probability $(w_t)_i$
- Sees potential costs $v_t \in \mathbb{R}^d$; pays expectation $\langle w_t, v_t \rangle$
- `Weighted-Majority` algorithm:
  - Start with $\tilde{w}_1 = (1,\ldots,1)$; $\eta = \sqrt{2\log(d)/T}$
  - For $t = 1,2,\ldots$
    - Follow with probabilities $w_t = \tilde{w}_t / \|w_t\|_1$
    - Update based on costs v_t as $\tilde{w}_{t+1} = \tilde{w}_t \exp(-\eta v_t)$ (exp is elementwise)

Learning from expert advice

- There are d available experts who make predictions
- At time t, learner chooses to follow expert i with probability $(w_t)_i$
- Sees potential costs $v_t \in \mathbb{R}^d$; pays expectation $\langle w_t, v_t \rangle$
- `Weighted-Majority` algorithm:
  - Start with $\tilde{w}_1 = (1,\ldots,1)$; $\eta = \sqrt{2\log(d)/T}$
  - For $t = 1,2,\ldots$
    - Follow with probabilities $w_t = \tilde{w}_t / \|w_t\|_1$
    - Update based on costs v_t as $\tilde{w}_{t+1} = \tilde{w}_t \exp(-\eta v_t)$ (exp is elementwise)
- **Theorem** (SSBD 21.11): $\sum_{t=1}^{T} \langle w_t, v_t \rangle - \min_{i\in[d]} \sum_{t=1}^{T} (v_t)_i \leq \sqrt{2\log(d)\, T}$ if $T > 2\log d$

Learning from expert advice

- There are d available experts who make predictions
- At time t, learner chooses to follow expert i with probability $(w_t)_i$
- Sees potential costs $v_t \in \mathbb{R}^d$; pays expectation $\langle w_t, v_t \rangle$
- `Weighted-Majority` algorithm:
  - Start with $\tilde{w}_1 = (1,\ldots,1)$; $\eta = \sqrt{2\log(d)/T}$
  - For $t = 1,2,\ldots$
    - Follow with probabilities $w_t = \tilde{w}_t / \|w_t\|_1$
    - Update based on costs v_t as $\tilde{w}_{t+1} = \tilde{w}_t \exp(-\eta v_t)$ (exp is elementwise)
- **Theorem** (SSBD 21.11): $\sum_{t=1}^{T} \langle w_t, v_t \rangle - \min_{i\in[d]} \sum_{t=1}^{T} (v_t)_i \leq \sqrt{2\log(d)\, T}$ if $T > 2\log d$
- Can avoid knowing T by *doubling trick*: run for $T = 1$, $T = 2$, $T = 4$, … sequentially

Learning from expert advice

- There are d available experts who make predictions
- At time t, learner chooses to follow expert i with probability $(w_t)_i$
- Sees potential costs $v_t \in \mathbb{R}^d$; pays expectation $\langle w_t, v_t \rangle$
- `Weighted-Majority` algorithm:
  - Start with $\tilde{w}_1 = (1,\ldots,1)$; $\eta = \sqrt{2\log(d)/T}$
  - For $t = 1,2,\ldots$
    - Follow with probabilities $w_t = \tilde{w}_t / \|w_t\|_1$
    - Update based on costs v_t as $\tilde{w}_{t+1} = \tilde{w}_t \exp(-\eta v_t)$ (exp is elementwise)
- **Theorem** (SSBD 21.11): $\sum_{t=1}^{T} \langle w_t, v_t \rangle - \min_{i \in [d]} \sum_{t=1}^{T} (v_t)_i \leq \sqrt{2\log(d)\, T}$ if $T > 2\log d$
- Can avoid knowing T by *doubling trick*: run for $T = 1$, $T = 2$, $T = 4$, … sequentially
  - Only blows up regret by < 3.5x (SSBD exercise 21.4)

Low regret for online classification

- For finite $\mathscr{H}$, we can just run `Weighted-Majority` with each $h \in \mathscr{H}$

Low regret for online classification

- For finite $\mathscr{H}$, we can just run `Weighted-Majority` with each $h \in \mathscr{H}$
  - Plugging in previous theorem, $\mathrm{Regret}_{\mathtt{WM}}(\mathscr{H}, T) \leq \sqrt{2 \log|\mathscr{H}|\, T}$

Low regret for online classification

- For finite $\mathscr{H}$, we can just run `Weighted-Majority` with each $h \in \mathscr{H}$
  - Plugging in previous theorem, $\mathrm{Regret}_{\mathtt{WM}}(\mathscr{H}, T) \leq \sqrt{2\log|\mathscr{H}|\,T}$
- For infinite $\mathscr{H}$, we need a not-too-big set of experts where one is still good

Low regret for online classification

- For finite $\mathscr{H}$, we can just run `Weighted-Majority` with each $h \in \mathscr{H}$
  - Plugging in previous theorem, $\mathrm{Regret}_{\mathtt{WM}}(\mathscr{H}, T) \leq \sqrt{2 \log|\mathscr{H}|\, T}$
- For infinite $\mathscr{H}$, we need a not-too-big set of experts where one is still good
  - Expert($i_1, i_2, \ldots, i_L$) runs SOA on $x_1, \ldots, x_T$,
    but takes choice with smaller Ldim on indices $i_1, i_2, \ldots, i_L$

Low regret for online classification

- For finite $\mathscr{H}$, we can just run `Weighted-Majority` with each $h \in \mathscr{H}$
  - Plugging in previous theorem, $\mathrm{Regret}_{\mathtt{WM}}(\mathscr{H}, T) \leq \sqrt{2 \log |\mathscr{H}|\ T}$
- For infinite $\mathscr{H}$, we need a not-too-big set of experts where one is still good
  - Expert($i_1, i_2, \ldots, i_L$) runs SOA on $x_1, \ldots, x_T$,
    but takes choice with smaller Ldim on indices $i_1, i_2, \ldots, i_L$
  - Can show (21.13-14) that one expert is as good as the best $h \in \mathscr{H}$,
    and there aren't too many of them,
    giving $\mathrm{Regret}_A(\mathscr{H}, T) \leq \sqrt{2(1 + \log T)\ \mathrm{Ldim}(\mathscr{H})\ T}$

Online convex optimization

- **Online convex optimization** is
  - Convex hypothesis class $\mathscr{H}$
  - At each step: learner picks $w_t \in \mathscr{H}$, environment picks convex loss $\ell_t(w_t)$

Online convex optimization

- **Online convex optimization** is
  - Convex hypothesis class $\mathscr{H}$
  - At each step: learner picks $w_t \in \mathscr{H}$, environment picks convex loss $\ell_t(w_t)$
- $$\text{Regret}(w^*, T) = \sum_{t=1}^{T} \ell_t(w_t) - \sum_{t=1}^{T} \ell_t(w^*), \quad \text{Regret}(\mathscr{H}, T) = \sup_{w^* \in \mathscr{H}} \text{Regret}(w^*, T)$$

Online convex optimization

- **Online convex optimization** is
  - Convex hypothesis class $\mathscr{H}$
  - At each step: learner picks $w_t \in \mathscr{H}$, environment picks convex loss $\ell_t(w_t)$
- $$\operatorname{Regret}(w^*, T) = \sum_{t=1}^{T} \ell_t(w_t) - \sum_{t=1}^{T} \ell_t(w^*), \quad \operatorname{Regret}(\mathscr{H}, T) = \sup_{w^* \in \mathscr{H}} \operatorname{Regret}(w^*, T)$$
- **Online gradient descent** (exactly like SGD) has:

Online convex optimization

- **Online convex optimization** is
  - Convex hypothesis class $\mathcal{H}$
  - At each step: learner picks $w_t \in \mathcal{H}$, environment picks convex loss $\ell_t(w_t)$
- $$\mathrm{Regret}(w^*, T) = \sum_{t=1}^{T} \ell_t(w_t) - \sum_{t=1}^{T} \ell_t(w^*), \quad \mathrm{Regret}(\mathcal{H}, T) = \sup_{w^* \in \mathcal{H}} \mathrm{Regret}(w^*, T)$$
- **Online gradient descent** (exactly like SGD) has:
  - $\mathrm{Regret}(w^*, T) \le \dfrac{\|w^*\|^2}{2\eta} + \dfrac{\eta}{2} \sum_{t=1}^{T} \|v_t\|^2$ where $v_t \in \partial \ell_t(w_t)$ are step directions

Online convex optimization

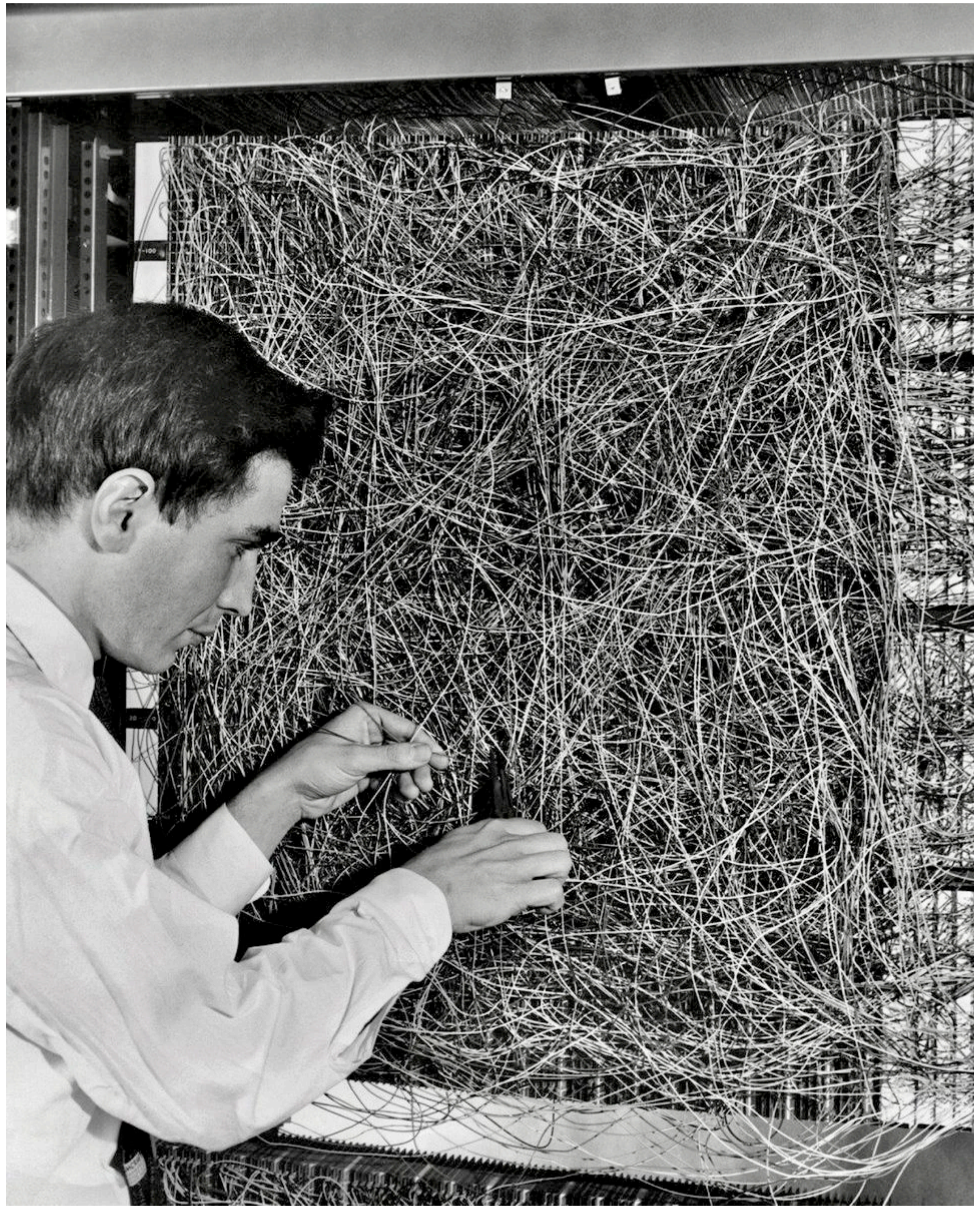
- **Online convex optimization** is
  - Convex hypothesis class $\mathcal{H}$
  - At each step: learner picks $w_t \in \mathcal{H}$, environment picks convex loss $\ell_t(w_t)$
- $$\mathrm{Regret}(w^*, T) = \sum_{t=1}^{T} \ell_t(w_t) - \sum_{t=1}^{T} \ell_t(w^*), \quad \mathrm{Regret}(\mathcal{H}, T) = \sup_{w^* \in \mathcal{H}} \mathrm{Regret}(w^*, T)$$
- **Online gradient descent** (exactly like SGD) has:
  - $\mathrm{Regret}(w^*, T) \leq \dfrac{\|w^*\|^2}{2\eta} + \dfrac{\eta}{2} \sum_{t=1}^{T} \|v_t\|^2$ where $v_t \in \partial \ell_t(w_t)$ are step directions
  - $\mathrm{Regret}(w^*, T) \leq \frac{1}{2}\left(\|w^*\|^2 + \rho^2\right)\sqrt{T}$ if ℓ_t are ρ-Lipschitz, $\eta = 1/\sqrt{T}$

Online convex optimization

- **Online convex optimization** is
  - Convex hypothesis class $\mathscr{H}$
  - At each step: learner picks $w_t \in \mathscr{H}$, environment picks convex loss $\ell_t(w_t)$
- $\mathrm{Regret}(w^*, T) = \sum_{t=1}^{T} \ell_t(w_t) - \sum_{t=1}^{T} \ell_t(w^*), \quad \mathrm{Regret}(\mathscr{H}, T) = \sup_{w^* \in \mathscr{H}} \mathrm{Regret}(w^*, T)$
- **Online gradient descent** (exactly like SGD) has:
  - $\mathrm{Regret}(w^*, T) \le \frac{\|w^*\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|v_t\|^2$ where $v_t \in \partial \ell_t(w_t)$ are step directions
  - $\mathrm{Regret}(w^*, T) \le \frac{1}{2}\left(\|w^*\|^2 + \rho^2\right)\sqrt{T}$ if ℓ_t are ρ-Lipschitz, $\eta = 1/\sqrt{T}$
  - $\mathrm{Regret}(w^*, T) \le B\rho\sqrt{T}$ if ℓ_t are ρ-Lipschitz, $\mathscr{H}$ is B-bounded, $\eta = B/\left(\rho\sqrt{T}\right)$

Online Perceptron

- You learned about Batch Perceptron in HW3
- Original algorithm is online
- Essentially identical, just only update on mistake
- Corresponds to online gradient descent on hinge loss
- Get same $(R/\gamma)^2$ margin-based mistake bound
  - Ldim = ∞ without the margin condition



Online-to-batch conversion

- If we have a good online algorithm, we have a good batch algorithm: just run it on the batch

Online-to-batch conversion

- If we have a good online algorithm, we have a good batch algorithm: just run it on the batch
- MRT **Lemma** 8.14: If $S \sim \mathscr{D}^T$ gives $h_1, \ldots, h_T$ for $0 \leq \ell(h, (x, y)) \leq M$,

$$\frac{1}{T}\sum_{t=1}^{T} L_{\mathscr{D}}(h_t) \leq \frac{1}{T}\sum_{t=1}^{T} \ell(h_t(x_t), y_t) + M\sqrt{\frac{2}{T}\log\frac{1}{\delta}}$$

Online-to-batch conversion

- If we have a good online algorithm, we have a good batch algorithm: just run it on the batch
- MRT **Lemma** 8.14: If $S \sim \mathscr{D}^T$ gives $h_1, \ldots, h_T$ for $0 \leq \ell(h, (x, y)) \leq M$,

$$\frac{1}{T}\sum_{t=1}^{T} L_{\mathscr{D}}(h_t) \leq \frac{1}{T}\sum_{t=1}^{T} \ell(h_t(x_t), y_t) + M\sqrt{\frac{2}{T}\log\frac{1}{\delta}}$$

- MRT **Theorem** 8.15: if $\ell(\,\cdot\,, z)$ is also convex,

$$L_{\mathscr{D}}\left(\frac{1}{T}\sum_{t=1}^{T} h_t\right) \leq \inf_{h \in \mathscr{H}} L_{\mathscr{D}}(h) + \frac{1}{T}\mathrm{Regret}_A(\mathscr{H}, T) + 2M\sqrt{\frac{2}{T}\log\frac{2}{\delta}}$$

(pause)

Differential privacy

- Randomized learning algorithm $A(S)$ is called **(ε, δ) differentially private** if

Differential privacy

- Randomized learning algorithm $A(S)$ is called **(ε, δ) differentially private** if
  - for all S_1, S_2 that differ on a single element (i.e. one person's data),

Differential privacy

- Randomized learning algorithm $A(S)$ is called **(ε, δ) differentially private** if
  - for all S_1, S_2 that differ on a single element (i.e. one person's data),
  - for all subsets $H \subseteq \mathscr{H}$, $\Pr(A(S_1) \in H) \leq \exp(\varepsilon) \Pr(A(S_2) \in H) + \delta$

Differential privacy

- Randomized learning algorithm $A(S)$ is called **(ε, δ) differentially private** if
  - for all S_1, S_2 that differ on a single element (i.e. one person's data),
  - for all subsets $H \subseteq \mathscr{H}$, $\Pr(A(S_1) \in H) \leq \exp(\varepsilon) \Pr(A(S_2) \in H) + \delta$
- Called **pure** DP if $\delta = 0$

Differential privacy

- Randomized learning algorithm $A(S)$ is called **(ε, δ) differentially private** if
  - for all S_1, S_2 that differ on a single element (i.e. one person's data),
  - for all subsets $H \subseteq \mathscr{H}$, $\Pr(A(S_1) \in H) \leq \exp(\varepsilon) \Pr(A(S_2) \in H) + \delta$
- Called **pure** DP if $\delta = 0$

- Used in practice (US Census, Apple, …), tons of work on algorithms

Differential privacy

- Randomized learning algorithm $A(S)$ is called **(ε, δ) differentially private** if
  - for all S_1, S_2 that differ on a single element (i.e. one person's data),
  - for all subsets $H \subseteq \mathscr{H}$, $\Pr(A(S_1) \in H) \leq \exp(\varepsilon) \Pr(A(S_2) \in H) + \delta$
- Called **pure** DP if $\delta = 0$

- Used in practice (US Census, Apple, …), tons of work on algorithms
  - Mijung Park and Mathias Lecuyer both work on this,
    teach courses (532P next fall, 538L now [but not next year])

Differential privacy

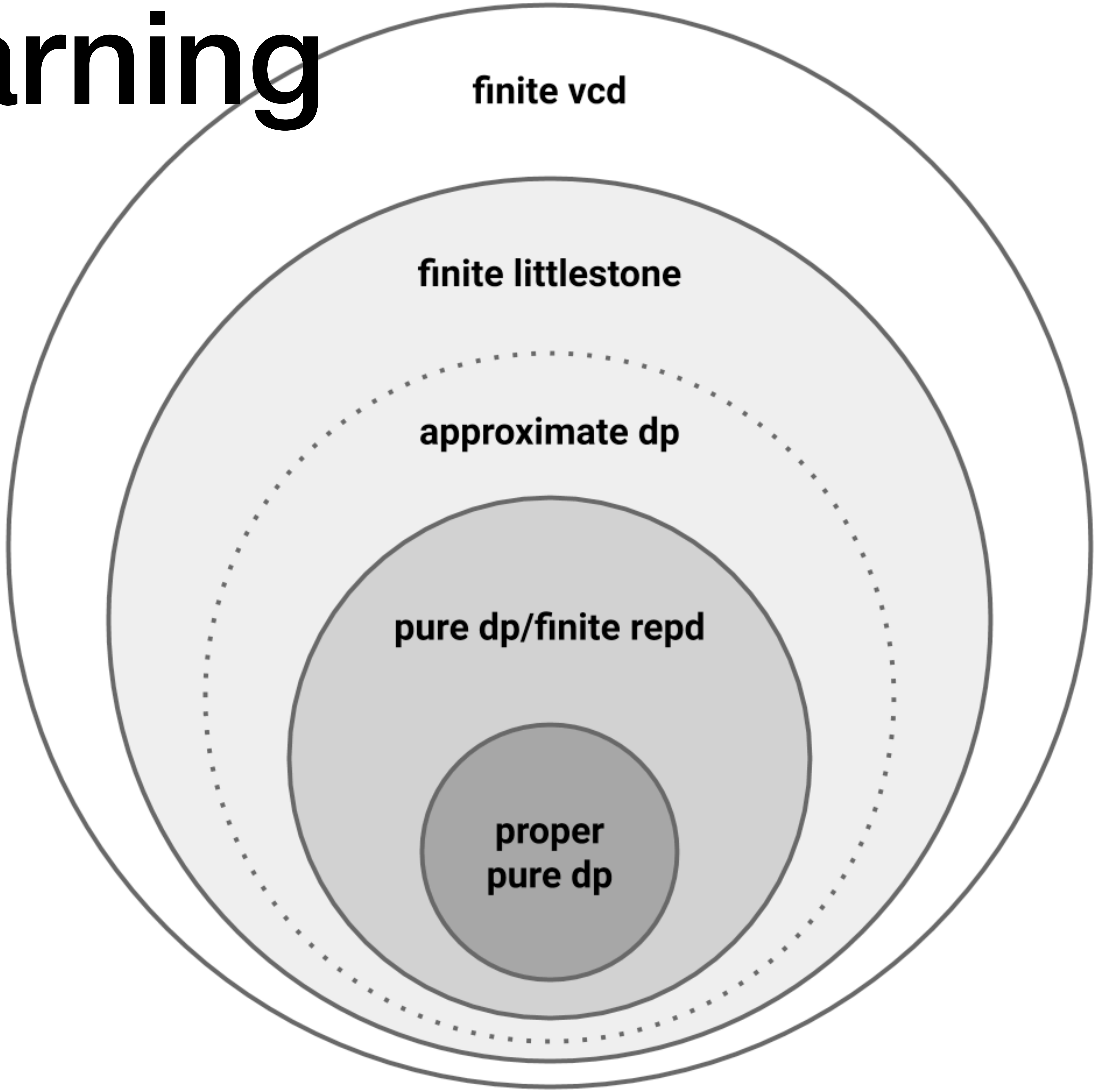
- Randomized learning algorithm $A(S)$ is called **(ε, δ) differentially private** if
  - for all S_1, S_2 that differ on a single element (i.e. one person's data),
  - for all subsets $H \subseteq \mathscr{H}$, $\Pr(A(S_1) \in H) \leq \exp(\varepsilon) \Pr(A(S_2) \in H) + \delta$
- Called **pure** DP if $\delta = 0$

- Used in practice (US Census, Apple, …), tons of work on algorithms
  - Mijung Park and Mathias Lecuyer both work on this,
    teach courses (532P next fall, 538L now [but not next year])

- Can be thought of as a particular **form of stability**

DP and online learning

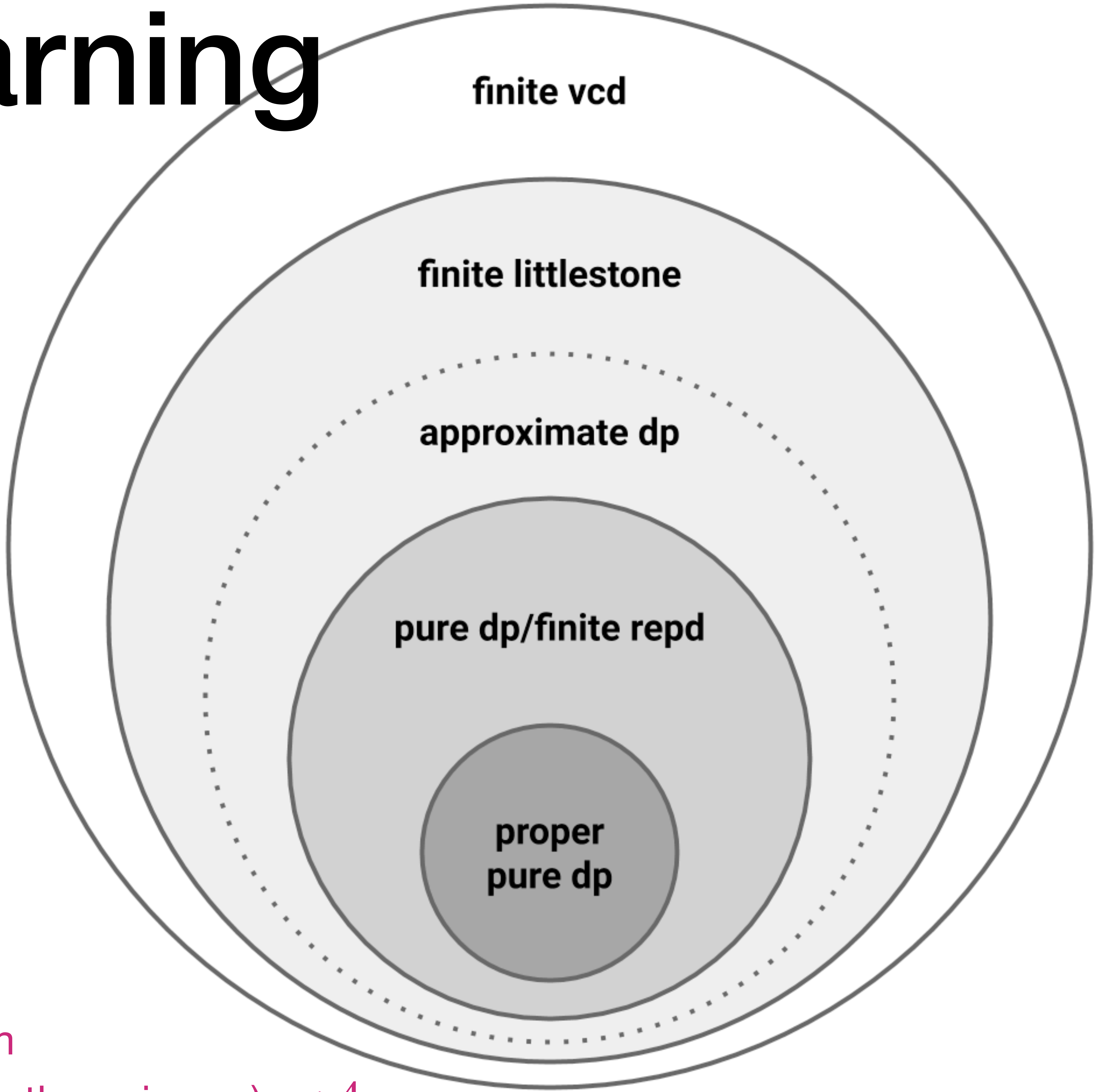
- Feldman and Xiao 2014:
  Pure private PAC learning takes
  $\Omega(\mathrm{Ldim}(\mathscr{H}))$ samples
  - Related to communication complexity



https://differentialprivacy.org/private-pac/

DP and online learning

- Feldman and Xiao 2014:
  Pure private PAC learning takes
  $\Omega(\mathrm{Ldim}(\mathscr{H}))$ samples
  - Related to communication complexity
- Alon, Livni, Malliaris, Moran 2019:
  Approximate private PAC learning takes
  $\Omega(\log^*(\mathrm{Ldim}(\mathscr{H})))$ samples



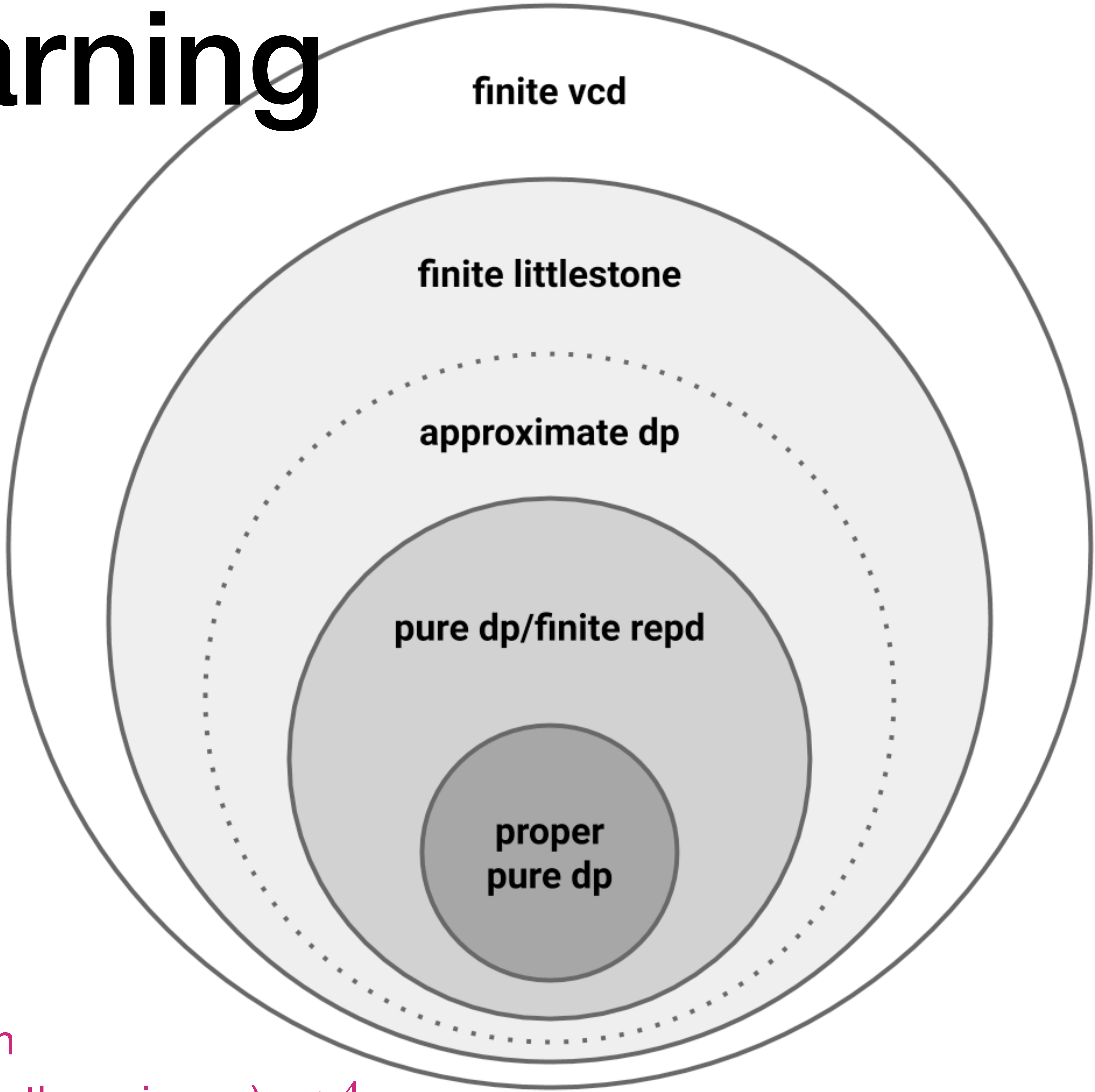
$\log^*$ = iterated logarithm

$\log^*$(number of atoms in the universe) ≈ 4

https://differentialprivacy.org/private-pac/

DP and online learning

- Feldman and Xiao 2014:
  Pure private PAC learning takes $\Omega(\mathrm{Ldim}(\mathscr{H}))$ samples
  - Related to communication complexity
- Alon, Livni, Malliaris, Moran 2019:
  Approximate private PAC learning takes $\Omega(\log^*(\mathrm{Ldim}(\mathscr{H})))$ samples
- Bun, Livni, Moran 2020:
  Approximate private PAC learning in $2^{\mathcal{O}(\mathrm{Ldim}(\mathscr{H}))}$ samples
  - analysis via "global stability"



$\log^*$ = iterated logarithm
$\log^*$(number of atoms in the universe) ≈ 4

https://differentialprivacy.org/private-pac/

DP and online learning

- Can learn differentially privately iff can learn online
  - Close connections via stability
  - But huge gap in sample and time complexity
  - Indications (Bun 2020) that converting one to the other isn't possible with polynomial time + sample complexity
  - Still a lot to understand here

Some of the stuff we didn't cover

- **Multiclass learning**: can use same techniques, need right loss
- **Ranking**: which search results are most relevant?
- **Boosting**: combine "weak learners" to a strong one (kind of like A3 Q3 b)
- **Transfer learning** / **out-of-domain generalization** / …: train on $\mathscr{D}$, test on $\mathscr{D}'$
- Do ImageNet Classifiers Generalize to ImageNet? / The Ladder mechanism
- **Robustness**: what if we have some adversarially-corrupted training data?
- **Unsupervised learning** (just the PCA question on A1)
  "How well can we 'understand' a data distribution?"
- **Semi-supervised learning** (just the algorithm from A4)
- **Active learning**: if xs are available but labeling them is expensive,
  can we choose which to label?
- **Multi-armed bandits**: which action should I take?
- **Reinforcement learning**: interacting with an environment with hidden state
- …